Project Number: JLS-0060

The Virtual Joust

An Interactive Qualifying Project

submitted to the faculty of the

Worcester Polytechnic Institute

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

Hyungjoon Kim Justin Liu Patrick Newell Steven Shidlovsky

Date: April 15, 2009

Approved:

Jeffrey L. Forgeng, Major Advisor

ABSTRACT

This project presents a sport of the past using current technologies to recreate the experience of jousting for visitors to the Higgins Armory Museum. Through collaboration with museum staff, intensive historical research, and a rigorous, iterative software development cycle, the project team developed a jousting simulation using technologies that incorporated Java, Flash, TCP/IP sockets, Bluetooth and XML. Nintendo® Wii remotes, embedded in a lance stub and to horse reins, were also used to further simulate realism in the user-application interface.

ACKNOWLEDGEMENTS

The project group would like to thank the following people for their support in the successful completion of this project:

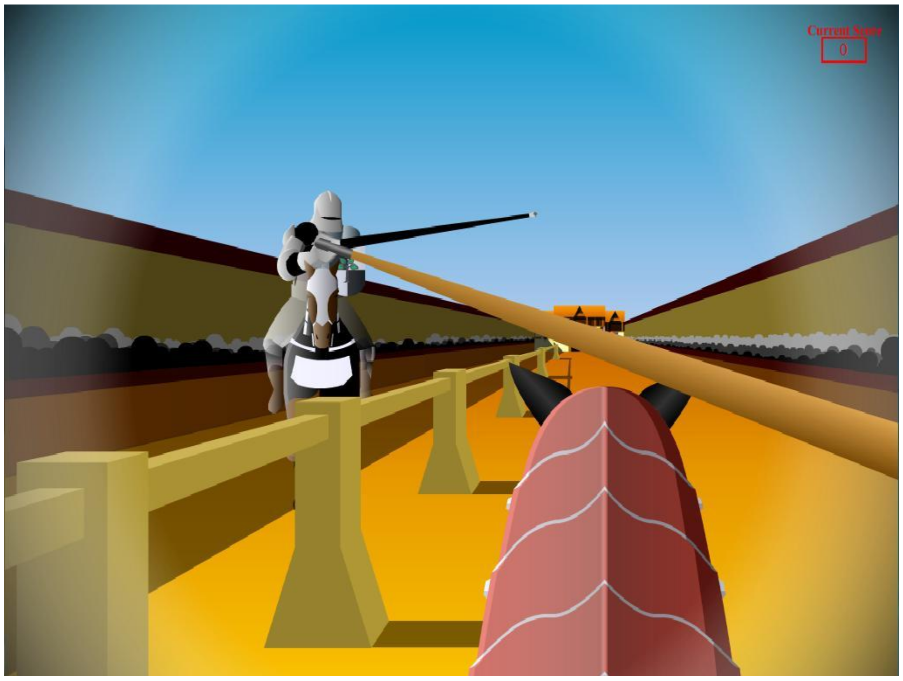
- Elizabeth Beinke for her assistance in generating graphic samples for the initial design
- Michael Calabro for his lending of hardware resources and time commitment to voluntary beta testing and technical photography modeling.
- Peter Keller for his providing of in-depth knowledge and insight in the areas of heraldry and professional software development.
- Patricia Martini for her professional assistance and editing with technical documentation.
- Emilia Martini for her assistance in acquiring hardware resources, revising technical documentation, and voluntary beta testing.
- Noel Naczi for lending his voice acting talent.
- The Higgins Armory Museum and its staff for providing access to both printed resources and museum pieces for our research.
- Nikki Andersen and Devon Kurtz for their assistance in coordinating our meetings with museum staff as well as providing invaluable advice.
- Professor Jeffrey L. Forgeng, WPI Project Advisor, for his guidance and help in obtaining access to the necessary resources to conduct our research.

TABLE OF CONTENTS

INTRODUCTION

THE JOUST

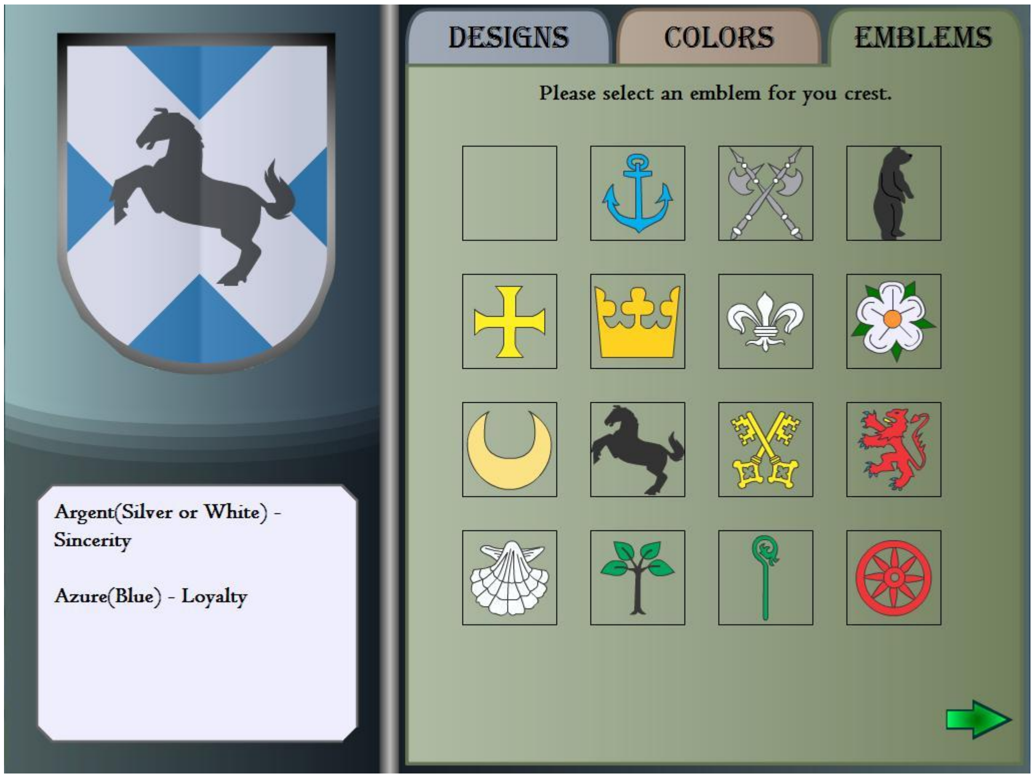
The scene stands still as the contenders glare at each other, separated by a distance shy of a hundred yards. The crowd grows restless as the final adjustments are made to each knight's mount. Suddenly, one of the knights begins to advance. In response to his rival's challenge, the second knight commands his horse forward. The horses gain speed as the distance between the two knights shrinks, the rhythmic galloping drowning out the cheers and jeers from the animated spectators. With less than a dozen yards remaining, both knights drop their lances and brace for impact. The outcome of this joust lies in the horsemanship and lance mastery of each knight, and perhaps a bit on the preference of Lady Luck.



A virtual jouster, moments before impact... (Virtual Jousting Simulator screen-shot)

The scenarios that were painted just now are descriptive of a highly celebrated sport of the past, the joust. Although tournaments existed long before the jousts, shortly after its introduction in the 11th century, it became the main attraction of subsequent tournaments. There were many variations of the joust, each with their own sets of rules and equipment. The most common type involved two knights wielding lances, charging at each other with the intent of shattering their own lance. Despite the numerous variations, all had their roots in war training. As times of peace lowered the necessity of war training, the joust evolved into a sport through the introduction of safety measure such as the tilt and rebated lances.

Accompanying the technical evolution of the joust was the redirection of ambitions and dispositions. Wars were infrequent, and there were few occasions more fitting for a knight to prove his worth than during tournaments featuring the joust. Commoners, who formed the majority of the spectators, were also drawn to tournaments, which provided a recess from the pains and monotony of everyday life. However, organization of jousts required significant funding and management, a responsibility that only the royalty can carry out. As such, the joust became one of the first true popular culture items, bringing the royalty and commoners together through entertainment.



Please select an emblem for you crest.

Argent(Silver or White) - Sincerity

Azure(Blue) - Loyalty

Heraldic Crest Designer (Virtual Jousting Simulators screen-shot)

Unfortunately, enthusiasm for the joust has faded for the most part. There are still societies that regularly host and participate in jousts, but the general populace is unable to indulge in this practice. Therefore, the goal of this project team was to introduce the joust in a format that would be most convenient and exciting: a simulation employing numerous innovative technologies.

THE MUSEUM

Due to the specificity of this topic, reliable information and artifacts are difficult to uncover. Fortunately, a local museum provided the project group with sufficient guidance and resources to develop a simulation of acceptable quality.

The Higgins Armory Museum, the interior designed in a Gothic castle style, carries a collection of arms and armors from various ages. The majority of the collection stems from European origins, with over 3000 armors and components and 1000 weapons and accessories, but the museum also features collections of African, Islamic, Indian, and Japanese origins. Aside from hosting a vast collection of arms and armors, the museum also hosts numerous other features for audiences of all ages. Performances and demonstrations are held regularly at the museum's auditorium. There is a special exhibition room where artifacts of a specific theme are displayed. Finally, there is a quest gallery where children can dress up as knights, explore exhibits that are geared toward them, or just relax with other children.

The Process

Worcester Polytechnic Institute's (WPI) Interactive Qualifying Project (IQP) introduces students to problems that involve consideration of parameters beyond those found in typical textbook problems. It requires the students to interact with the community of interest as they proceed to deliver solutions to various problems. With this mindset at the onset of the project, the member of this project team set out to the Higgins Armory Museum (HAM) with one core objective: to determine the type of virtual exhibit that would benefit the patrons of the museum most.

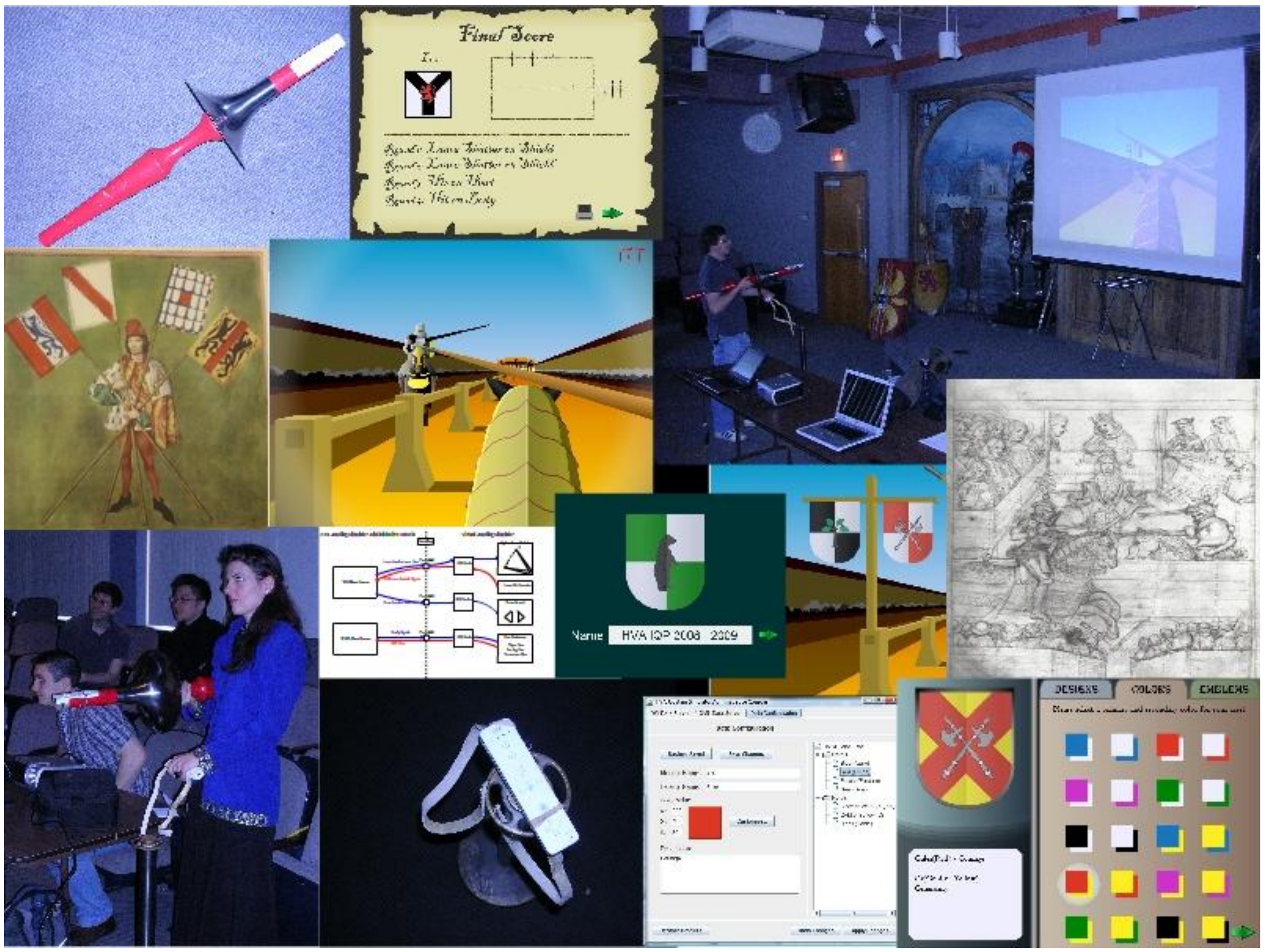
For the first term of the project, research in various areas was conducted. The four major areas of research include: museum profile and research materials, museum visitorship, design concepts, and technical concepts. During this phase of the project, interviews and meetings were conducted with HAM staff in order to brainstorm ideas for a virtual exhibit. Museum context and visitorship research was done to get an idea of the target audience and context for the exhibit. Design and technical research was done to gather tools that will facilitate the completion of this project. By the end of the term, the list of potential exhibits was narrowed down to one: a jousting simulator.

The second term of this project was again, split into four main branches: research on the historical context of the joust, research on the mechanics of the joust, sketching out of interfaces and graphics to be used in the simulator, and development of an underlying framework for the simulator. During this phase, Nintendo Wii remotes were used to emulate the jousting lance movements and a simplistic form of horse control, and an extensive graphics library was compiled from the ground up with the assistance of a student majoring in Interactive Media and Game Development (IMGD) fulfilling an independent study program at WPI. A storyboard drawing from the historical context research was drafted, and game mechanics were modeled after the research on the actual mechanics of the joust.

During the third term of the project, all four of the team members worked on developing and testing the simulator to a level of software maturity where it could be introduced to the public. In addition to the continuous implementation of new functionality, the simulator was constantly revised and changed on a weekly basis to better coincide with the derived game mechanics and historical accuracy. Ease of use and ergonomics were also major issues during the development process, causing the input device mechanisms to be modified until they reached a point of acceptable usability.

The Product

After two terms of research and development, the first beta version was completed. It consisted of several features including a historically accurate heraldry customizer, where the user would generate his own heraldic crest that would be associated with his virtual avatar, a real-time jousting simulator, which allowed a user to joust against a computer-controlled enemy jouster using an actual jousting lance stub with an embedded Nintendo Wii Remote and a simplistic form of horse control employing a second Nintendo Wii Remote attached to horse reins, and a unique scoring system that was modeled after authentic scoring systems of historical jousts. The full application was developed to be compatible with standard personal computer setups using a keyboard and mouse as input devices, as well as more advanced setups that include touch screens and projectors.



History of the Joust

The tournament of the Middle Ages – an event that evokes images of heroic knights, fair ladies, and above all else, the joust. These events played an important role in everyone's lives. Ordinary citizens would attend these tournaments as a way to get away from the daily struggle of life. Meanwhile, royalty would use them to display the power of their kingdom through their knights. To the knights, and later on to some princes, this was a way to relax and hone skills necessary for war. Like any sport, it underwent changes until it reached its peak in the sixteenth century. Beginning as nothing more than a brawl, slowly rules were created and the tournament became an organized affair. The joust, also known as the hastilude (Barber & Barker, 1989, p. 3), got introduced to the tournament and became the main attraction. In many respects, the tournament played a central role in the history of the middle ages.

The Origins of the Tournament

Information about the early origins of the tournament is rare. Most of the information that still remains comes from unreliable sources such as romances and novels. Due to this, no one can be sure of when the first tournament was held nor does anyone know when it officially became a sport. It is believed that the first tournament were the horsemanship games between Louis the German and Charles the Bald in 842 (Barber & Barker, 1989). Since no earlier accounts of other tournament-like games have been found, these games are thus recognized as the first tournament that occurred. As a sport, evidence points to having its roots in northern France during the end of the 11th century (Barber & Barker, 1989, p. 14). This idea is reinforced by how many of the terms used for the tournament, such as "tourneamentum" and "tournoi", are French (Clephan, 1919, p. 1). Further evidence comes from information that in this same area of France, a new type of fighting style had been created. This new style of fighting allowed cavalry to keep their lances after they had attacked the enemy. Before this, after attacking, most cavalry would be defenseless as their lances would be destroyed on impact with the opponent. While this advantage was useful, it was balanced by the disadvantage of requiring a lot of training in order to execute it effectively. The tournament was the method developed to provide the necessary training.

The earliest tournaments were primitive affairs. More akin to an actual battle than a formal event, tournaments often occurred between townships and did not contain jousts. This resulted in the common practice of naming the tournament after the two townships that were participating in it. There were no rules during these fights and even if there were, no one was around to enforce them. Due to this, dirty tactics were common during tournaments. Another disadvantage to this style of tournament was that participants rarely cared about the area in which they were tourneying. This often resulted in the complete destruction of the area the tournament occurred in, making the people that lived in the eareas where the tournaments occurred hate them. Formal areas for tournaments were later established to prevent this problem.

The Rise of the Tournament and Objections

Between 1125 and 1130, tournaments started to pop up all around Europe. Due to the increasing popularity, the Catholic Church took notice. They did not approve of this new sport. They viewed it as brutal and bloodthirsty even as rules emerged and it became more civilized. To them, there was no difference between a real battle and a tournament. In an attempt to stop people from participating, they threatened knights with excommunication and denial of Christian burial if they were killed in a tournament. This threat was not strictly enforced however; there were times it was used on knights that were killed during tournaments. Even though most knights knew about the threat, many chose to ignore it. In the event that it was enforced, the family would plead with the church to get their dead family member a proper burial. Any reasonable excuse the family could think of would be used. For example, the dead knight had rejected the tournament before his death. The church would usually then provide the Christian burial as long as the rest of the family swore never to participate in tournaments again.

As the tournament underwent changes during the 1170's, it gained attention from another group – writers. They would often include the tournament inside of the romances and novels they wrote for royal courts they served. This resulted in the tournament taking a central role in chivalric literature. As this trend continued, the stories slowly leaked outside into the hands of ordinary citizens. Common people that heard the stories got fascinated into wanting to go and see a tournament to compare it to what they had heard about in the story, increasing their popularity. By the end of the 12th century, the tournament had become a major part of chivalrous mythology and part of knightly life (Barber & Barker, 1989). As more people sought to attend the tournaments, spectatorship slowly became an important aspect of the tournament.

In England, the tournament first appeared during the reign of King Stephen. Stephen, who reigned between 1135 and 1154, had a very weak government compared to his predecessors. During the reigns of Henry I and Henry II, the tournament had been banned (Barber & Barker, 1989). This forced those that wanted to participate in a tournament to go overseas to France. When Stephen took control of the government, he had little control of the government and no true power. This allowed knights to have tournaments in England despite the government ban. The first casualty in these tournaments was Hugh Mortimer of Worcester. The bans against the tournament would continue until Edward I decided to revoke the ban and place tournament organization under the government's control.

Edward I enacted a law in 1194 that gave the government control over the tournament. His primary goal in doing this was to make money. His government was having financial problems and he saw this as an easy way to raise a large amount of funds. To help make it so he could have as many people as possible participate, the fee to participate was based on your rank in society. The higher ranking you were, the higher the cost became. In order to regulate tournaments better, he created five places for them to officially occur and banned them from occurring anywhere else (Clephan, 1919). This prevented people from avoiding paying the necessary fees. This new regulation allowed the tournament to flourish inside of England.

William Marshal

A jouster that gained fame for his unique method of tourneying was William Marshal. He participated in tournaments during the 1170's and 1180's. His most unique quality was that he was not of noble birth. Normally, anyone not of noble birth was prohibited from participating in the tournament. However, this rule was not enforced as much in England as it was in France and Germany thus allowing William to participate. Unlike most knights, he treated the tournament as a business. He was in it to win and acquire more gear so he could continue to be in tournaments. Through jousting, William was also able to raise his social status. Unlike other ways of doing this, the tournament did not have a social stigma attached to it. His skill at the joust was held in high enough regard that he had other kings offering to pay him if he jousted for them. Despite these offers, he continued to stay loyal to his king Henry III.

Jousting Equipment and the Playing Field

The equipment used for the tournament originally was the same as the gear used on the battlefield. Of the implements used, the most basic piece of equipment was the lance. Originally, the lance was nothing except a piece of straight, smooth wood that was pointed at the end. It could be made out of any type of wood but soft woods were common so it would be easier to break and score points. To increase participant's safety, a vamplate was added in the 14th century to protect the user's arm. For armor, early jousters used standard battlefield armor, which was a chainmail shirt. It was not until the beginning of the 13th century that plate armor started to be used for jousting. Horses themselves were not armored until the third quarter of the 13th century, but trappings for the horse were common earlier (Clephan, 1919).

The 14th century brought specialization to jousting equipment. The first mention of armor created specifically for jousting was made in 1391 by Sir Bartholomew Burghersh (Cripps-Day, 1918, p. 49). This armor would be more protective than battlefield armor. Unfortunately it would also hamper movement much more than battlefield armor would. This was not an issue for armor made just for the joust as movement was not a major ability needed to joust. By 1330, jousters no longer were using just blunted lances. Instead, lances were tipped with coronals, blunt metal tips used to increase safety. Shields were now flat and triangular with the knight's heraldry on the front. The knight themselves would wear surcoats with their heraldry on them and made sure that their horse had similar trappings. Their belts would also be elaborately decorated (Clephan, 1919, p. 29). The lances would sometimes have pieces of cloth tied to the end of them. This came from the knight's lady. She would rip a piece of her clothing and hand it to her knight for luck. This image of a fully armored knight and horse with flowing surcoats is what most modern people think of when they think of a knight at a tournament.

The jousting arena itself underwent changes as the tournament evolved. The first jousting arenas had few defined boundaries and were often just an open field. Over time, official areas for the joust were created. This was in part due to the increased preparation and buildings that were becoming necessary. Spectator seating, galleries for the lords and ladies, and feasting halls were some of the necessary buildings and these would be expensive to continuously rebuild. The ground would have a thick layer of sand and tanning refuse to soften the fall of knights if they were dismounted. To protect the spectators, the enclosure for the jousting arena would be a double row of palisading that would be high enough to prevent horses from jumping over it and into the audience. A barrier, called a tilt, was added between the two jousters in the 1400s to prevent the horses from colliding into each other when the knights were charging at each other (Clephan, 1919, p. 39).

Training for the Joust

In order to train for the joust, there were two earlier games that were used. These games were the quintain and running at the ring. Each focused on improving skills the knight needed in order to win a tournament. The quintain focused on perfecting the knight's aim, ability to stay steady in the saddle after impact, and discarding broken lances properly. This game had existed long before the joust and because of the significantly reduced amount of danger involved, more people where able to play. The objective would be to hit the target, which would usually be either a post or a stuffed doll with a shield. More advanced devices actually would punish the player if they did not hit the target correctly to train them not to repeat the bad habit. This game remained popular until the 17th century.

Running at the ring was a later sport that was created to perfect other essential jousting skills. The objective of this game was to run at rings and grab them onto the end of the lance. This helped the knight practice their precision with hitting targets. The ability to aim accurately was an important skill as it allowed a knight to break their lance on the target easier, which meant they were able to get more points.

The Joust in the 14th Century

The 14th century was a period of evolution for the joust and the tournament. The transition from chain mail to plate armor that began in the 13th century was complete by the end of the century. The tournament, and thus the joust, was having its view within society change. It was becoming a way to celebrate or greet important government officials into a town. Royalty would have a tournament to commemorate any special event such as a coronation or wedding. These types of tournaments were often announced well in advance, allowing any foreign knights ample time to come (Clephan, 1919, p. 24). The tournament had begun its transformation into an event in which pageantry took on an ever increasing role.

Setting up a Tournament

There was a standard procedure on how to announce, set up, and run a tournament. It would begin with a herald from the opposing side coming to inform the opponents of the challenge. The heralds were selected for this job since if the opposing countries were not on friendly terms, the herald would be able to deliver the message without issue. It was common procedure not to attack heralds so that they could relay messages. If they accepted the challenge, the details such as the time and location would be discussed and agreed to (Cripps-Day, 1918). With this information, it was possible to announce the upcoming tournament to other countries so that their knights would be able to come and participate in the tournament if they wished. On the arrival day, the knights and their escorts would come to the tournament grounds.

The day after arrival was a day of preparation. Knights would need to line up their helmets on a special location, usually named the hotel, so that the judges and ladies could examine them. This gave the judges and ladies time to decide if the knights would be allowed to participate in the tournament. Knights would be expelled from the tournament for a variety of reasons, the most common being that the knight was not chivalrous. The third day of the tournament began with the swearing in of the knights and the choosing of the chevalier d'honneur. The chevalier d'honneur was a knight whose lance became known as the couvre-chef de mercy (Cripps-Day, 1918). When this lance was used to touch another knight, no other attacks could occur on that knight.

The fourth day brought the actual jousting. Jousting would often start at about noon time. This was due to the large amount of time it took for knights to put on their armor and prepare everything for the upcoming joust. After all the knights were ready, they would go to the lists where they would joust against their assigned partners or those knights that they had challenged earlier. By doing well, they would gain fame and recognition from their lords and ladies. After all of the jousting was finished and a winner determined, a feast would be put on by the sponsoring lords. During the feast, prizes would be given out to the knights that had done well in the tournament. Prices were often expensive and extravagant items such as rare gems or gold items made by highly skilled craftsman (Cripps-Day, 1918). The day after the feast, everyone would head back to their homelands.

While this procedure seems simple, it does not express the high price tag that came with tournaments. Holding a tournament became a way for young princes to show off their wealth and power. However, many of these tournaments failed to produce the results that they were after. Many of them did not have any foreign knights attend them, meant that only local knights were participants. This caused a lack of spectatorship since the tournament itself would be anticlimactic. One of the reasons for the high price tag was the large amount of preparation necessary to run a tournament. If a tournament was not properly planned for, major issues could erupt.

An example of a tournament that was poorly planned became an event known as Black Shrove Tuesday (Barber & Barker, 1989, p. 61). In 1376 in the town of Basle, Germany, a tournament was held that did not have the proper planning. During this time, tournaments were illegal in Germany but with the proper preparation, it was possible to put them on safely without worrying about the law. Proper preparation meant ensuring the jousting schedule was well organized, extra guards were hired for security, and the safety of the spectators was considered. During this tournament, none of these things had happened. The tournament was poorly marshaled. Horses were running into the crowds. Discarded lances from the jousters were landing inside of the spectator area causing injury to the spectators. As the number of issues mounted, the townspeople and spectators were unable to contain their rage and attacked the nobles that were running the tournament. This resulted in the death of a few of the nobles and the town falling into anarchy for a few days. In the end, the town ended up suffering more due to the outbreak. They got condemned by Leopold of Austria and had to sue for peace. This example demonstrates some of the consequences of a badly planned tournament. Without proper planning, chaos would occur and result in a catastrophic event for the participants and the hosting lords.

Individual Challenges

While pageants and large scale tournaments were becoming the norm for royalty to hold, another form of challenge was rising. This type of challenge was a duel that consisted of three parts. The three most common parts were jousting, battle-axe swings, and dagger swings. Each participant would be given a certain number of swings to injury or kill their opponent. The number of swings with the battle-axe and dagger would be the same as the number of passes with the lance. Usually duelers would have three uses of each weapon. As the century progressed, higher numbers became common. Near the end of the 14th century, five had become the most common number of swings, though sometimes 10 or 20 attacks of each were used (Clephan, 1919).

Some famous examples of this type of challenge are seen during the 14th century. Before the walls of Rennes in 1357, Bertrand du Guesclin and Sir Nicholas Dagworth participated in this type of duel (Clephan, 1919). The duel ended with no clear winner and neither party ended up getting hurt. Another example would be from France. Shortly before the death of King Charles V in 1380, Frenchman Gauvain Micaille and Englishman Joachim Cator dueled. During the duel, Cator managed to inflict a serious wound to Micaille's thigh during their jousting. However, as the blow occurred during jousting, it was considered dishonorable.

Cator managed to explain his lack of chivalry by stating that his horse had gone out of control and that he had been unable to aim properly. This reasoning was accepted and no further issue was raised about it.

De Boucicaut

The jouster of the 14th century that everyone wanted to mimic was the Frenchman Jean Le Maingre, also known as De Boucicaut. He gained his fame by extending a challenge to all knights that he would joust against all comers for 30 days straight (Clephan, 1919). The opposing knight had the choice between using blunted lance tips or pointed tips by banging on the correct gong that had been sent up outside of the list. If a knight that came to challenge him did not have the proper equipment, he provided the challenger with the necessary equipment so they could joust. The list itself was highly decorated to show off his wealth and power. After each joust, he attempted to get a small break in order to recover from the previous battle. These breaks were short and few however as there were many knights that wished to challenge him. After the end of the 30 day challenge, he had managed to win against all comers, demonstrating his strength and power as a jouster.

15th Century

During the 15th century, the tournament and joust underwent more changes. New technologies and improved armor specialization further reduced the risk of injury that was associated with jousting. Rules were more firmly established and strictly adhered to in order to make the tournament safe and orderly. In addition, pageantry was becoming an increasingly important part of the tournament. While the eyes of the spectators were still primarily on the joust, the amount of ritual and decoration that was included was increasing. Any time royal officials came into town, a tournament was held in order to greet them. This further advanced the idea of the tournament being a type of party instead of a demonstration of fighting. The melee, which was once a common practice at tournaments, had been all but eliminated at this time. Many saw it relying too much on chance instead of individual skill. This century is the last century in which the joust and the tournament were vital aspects of life for many.

The form of jousting that was most common during the 15th century was the Kolbenturnier or baston course (Clephan, 1919, p. 41). The main objective of this type of jousting was not to injure the opponent. Instead, the riders were attempting to hit the crest off the helmet of the opponent. The more times a knight managed to hit the crest off, the more points they scored. The weapon that was used was called the kolben, which was a heavy polgonally-cut baton or mace that was crafted from hard wood and 80 centimeters long (Clephan, 1919, p. 42). Since the target of the game was located on the head, helmets were heavily padded and armored in order to prevent injuries.

The development of the tilt during the 15th century was an important addition to the jousting field. The tilt was a wooden barrier that was placed between the two jousters which prevented the horses from hitting each other and prevented the jousters from hitting each other head on with the lances. These two points further reduced the risk associated with jousting. By preventing the horses from seeing each other, it gave the knights' greater control over them. By hitting at an angle, the chance the lance would pierce though the armor was greatly reduced since it would reflect off the opponent easier. Despite the increased safety the tilt brought to jousting, there were some that did not approve of the tilt. These objectors thought that by eliminating the risk associated with jousting, you were making into a child's game. These knights still preferred the old style of jousting where sharp-tipped lances were used and padding was placed on the front of the horses to reduce the impact in the event that they crashed into each other. They also disliked the cantle as it made it harder to unhorse an opponent. They felt that this was another item that was taking away the soul of jousting. While there where knights from all around Europe that liked this hard core form of jousting, it was especially popular in Germany.

Armor also advanced technologically during the 15th century in order to provide more protection to the participants of the joust. Areas that were usually hit during the joust were reinforced with additional plating to provide greater protection. In addition, since most attacks hit the left side of a knight's body, the left side of the armor was made thicker than the right side. The vamplate was also enlarged to provide increased protection. An issue with all this new armor was that it was making it impossible for the knight to move around in his armor. This lack of mobility could prove to be either a disadvantage or an advantage. While it was hard to get around, it made it easier to aim with the lance as that was the only part you could move. Overall, the advances to the armor in this century made it so when properly used, there was little risk of injury to the participants.

Downfall of the Joust

The downfall of the joust began in the 16th century. There are many reasons that attributed to this downfall, though the most common one is the rise of firearms. While this did play a role in the downfall of the joust, it was not the only factor. The increasing effectiveness of firearms was making plate armor much less effective. Firearms were able to break through the armor rendering the protection that it used to give ineffective. An armored knight was similar to a tank. A ground solider had few options for destroying a tank with just a gun. However, when given a rocket launcher, the tank suddenly does not provide the protection it used to. The armored knight also had a lack of mobility due to the restrictiveness of the armor. As tactics changed and the protection that armor gave became obsolete, the use of knights started to be discontinued.

As stated above, changing tactics were another reason that uses of the knight were declining. Mobility was becoming a vital part of any strategists' plans to win a battle during a war. This presented a problem as wartime tactics changed to rely more on mobility than on line battles. Knights in armor were not very mobile and could not be deployed quickly. They required people to dress them in their armor, prepare their horses, and undress them after the battle. This amount of setup time was not available with the new emerging type of warfare. Quick movement on the battlefield was also impossible as armor by nature was hampering to the movement of the knight. As warfare moved towards increased mobility, the use of armor continued to disappear.

Armor itself had another factor that was reducing its usage - cost. Creating a custom suit of armor that would allow for maximum mobility and comfort was out of reach for most of society. Even in the Middle Ages, only the rich were able to afford plate armor. To help reduce the cost of armor, mass produced pieces were created. However, this type of armor rarely fit well and caused discomfort when marching to battle. Many soldiers threw these pieces of armor onto the side of the road despite the consequences because it was just too uncomfortable to wear. They knew it offered little protection and was not worth the discomfort. As fewer people owned or used armor, less people were able to participate in the joust as plate armor was a necessity.

The last pieces of the puzzle that were bringing down the joust were the loss of the values that surrounded the joust and the lack of royal participation. The joust slowly moved away from a fight between two highly skilled warriors to more of a theatrical performance. Chivalry, which was a major part of the tournament, was itself fading. Individual power and greed were becoming a much greater part of a person's life. Fighting for a lord and lady's for just honor and glory were gone. Royalty, which provided a major source of funding for these expensive events, stopped participating after Henry II of France died while jousting in 1559. He had forgotten to lower his visor, which was a common mistake. However, the person he was jousting against, the Constable of France, did not lower his broken spear in time. With his visor up, the broken spear was able to hit his face and kill him. Without funding, it was much harder to hold tournaments.

The Joust as Theater

Perhaps the best demonstration of how the joust was changing was its degradation into a theater show. It might be expected that this would occur as the tournament had pageantry and decoration taking greater significance in the tournament over the joust as the 15th century progressed. Tournaments were beginning to only include jousts as a formality, not because it was the main source of attention. By making the joust into a theater show, the last remnants of the original joust had disappeared.

An example of the joust being used as theater was seen at the tournament at Tempio d'Amore (Barber & Barker, 1989). Each of the battles had been prearranged to ensure that the correct people would get to the final round. The joust at this tournament can definitely be classified as theater when it becomes known that on top of these prearranged fights, they also included a storyline to make it more interesting. This makes it have absolutely no resemblance to the original tournament where knights would try their best in order to win and the outcomes of tournaments were never guaranteed and predestined. Having the joust turn into an item for theater demonstrates how far the joust had fallen.

Modern Attempts to Revitalize the Joust

Years after the tournament ceased to exist there were some modern attempts to revitalize it. The people that sought to bring back the tournament wanted to show the old glory and values that existed in tournaments. After the slaughter seen in the world wars, chivalry and honor in battle were ideals that seemed to have disappeared. By bringing back the tournament from extinction, these old ideals could be seen again as a stark difference to the impersonality of modern warfare.

The first attempt to bring back the tournament occurred in Ayrshire in 1839 (Clephan, 1919). Known as the Eglington tournament, it encountered many problems when it tried to get organized. Many people did not see a reason for attempting to revive the tournament. It was an ancient practice that had no useful context in society in the 1800's. Despite these protests, Eglington managed to collect the necessary funds and run the tournament. Even with his careful planning, the tournament was considered a failure. The day the tournament was suppose to occur, rainfall ruined most of the pageantry and flooded the meal hall. Although the tournament was run successfully a few days later, this did not overshadow the failure of the first attempt. They had tried to mimic the historical details as closely as possible. This made the event have little connection with the current audience. Another tournament that was held about 60 years later that did not push to have complete historical accuracy was more successful.

The tournament that was more successful was held at Brussels in 1905 (Clephan, 1919). They decided to set this tournament to mimic the event when the joust was the most spectacular. To appeal to a modern audience, they added more modern twists to it such as selling postcards. These changes allowed this recreation to have much greater success than the Eglington tournament. As the 1900's continued, the fascination with the middle ages increased. This made these tournament recreations a much more compelling event to attend.

In conclusion, the tournament played a vital role during the Middle Ages. It filled several niches such as entertainment, celebration, and war skill practice for knights. It also helped to advance technology as jousters sought the best equipment to gain an edge in the tournament and to stay safer. Overall, the tournament during its height promoted good values, including respecting women, preserving the land you traveled on, and treating others with respect. Even today in the 21st century, we emulate these events to remember the values of this time. The tournament will continue to fascinate and provide us with a means of getting a glimpse of this time period.

Mechanics of the Joust

It is believed that medieval tournaments were inspired from the Roman Troy Games; these games were essentially mock battles amongst various warriors (Blair, 1958, p. 156). Consequently, the earliest jousts were fought using typical armors and weapons of the time; there were no special equipment designed for safety so tournaments did not differ much from actual warfare (Blair, 1958, p. 156). Eventually, two broad categories of jousts emerged: jousts of peace and jousts of war. Jousts of peace were fought using rebated lances (sharp ends blunted), and the objective was to shatter the lance or dismount the opponent. Jousts of war were fought with real arms and armor, usually to the death (Blair, 1958, p. 157).

Despite efforts to reduce the danger in jousting by including a barrier between jousters and using more specialized arms and armor, jousting was still a dangerous sport. Many of the participants had several years of apprenticeship or experience in actual warfare (Barber & Barker, 1989, p. 6). However, despite the dangers of the jousts, there were always knights seeking fame and fortune. As a matter of fact, in the 15th century, individual challenges grew in popularity because they were one of the few ways where a new knight could gain recognition (Barber & Barker, 1989, p. 107).

Joust Styles

Starting from the early 15th century, jousting was developing into a sport practiced for its own sake; it was originally practiced for war training (Blair, 1958, p. 158). Stricter rules were devised and different styles developed. Generally, there is very little information on jousting styles outside of Germany. This is because 15th century Germany was the location for jousting during its golden age (Barber & Barker, 1989, p. 62). Although many types of jousts were practiced in Germany, each of these styles was basically a variation of two main styles: the Gestech and the Scharfrennen. The differences between these two styles stemmed from the type of lance used and the objectives of the run (Blair, 1958, p. 160).

The Gestech

The Gestech was originally used to describe jousts fought with rebated lances (Blair, 1958, p. 160). In the late 14th century, a special form of the Gestech, known as the Hohenzeuggestech, developed. Like the Gestech, the main objective of this course was to splinter the lance. However, the most distinctive element of this style stemmed from the saddle. For this style, the saddle was raised about 10 inches above the horse's back so that the knight assumed a standing posture when mounted (Blair, 1958, p. 161). The front of the saddle was designed like a wooden shield to protect the knight from feet to waist, and two wooden bars curved around his thighs to prevent him from being thrown out (Blair, 1958, p. 161). The typical equipment used for this style was field armor without the leg harness, a frog mouthed helm, and a small shield covering the left side of the body (Blair, 1958, p. 161).

After c. 1450, the Hohenzeuggestech was replaced by the Gestech (Blair, 1958, p. 161). In addition to the splintering of the Hohenzeuggestech, an additional objective was included: unhorsing (Blair, 1958, p. 161). Compared to the Hohenzeuggestech, the saddle of the Gestech was less elaborate; it was of normal size and lacked the bars surrounding the thighs. Since the leg protection from the Hohenzeuggestech was lost, a new defense known as the Stechsack was implemented; this was a thickly padded bumper that hung around the horse's neck in such a way as to protect the front of the horse and the rider's legs (Blair, 1958, p. 161).

The Scharfrennen

In the Scharfrennen, the main objective was unhorsing, but points were also gained for splintering lances (Blair, 1958, p. 162). The earliest reference to this style of joust by name was found in the Archduke Friedrich's 1436 inventory of armor (Blair, 1958, p. 163). From illustrations dating before the end of the 15th century, it was suggested that the Scharfrennen was originally fought in a light half-armor, with a sallet, bevor, and a rectangular shield covering the left side of the body (Blair, 1958, p. 163). By c. 1480, a special armor was designed for the Scharfrennen. This armor did not have leg harnesses, vambraces, pauldrons or gauntlets; this was because the large Reetartsche and the vamplate of the lance was considered to be sufficient protection for the arms and hands (Blair, 1958, p. 163).

Rules of the Joust

There are various rules specific to different types of jousts. A few examples of rules follow:

Rules by Edward IV in 1466

An example of jousting rules follows. These rules were decided by Edward IV in 1466 (Cripps-Day, The history of the tournament in England and in France, 1918).

- Each knight to run six courses with each other knight at slow canter.
- Points Scored
  - Lance struck and broken between waist and bottom of helm – ADD 1
  - Lance struck and broken on helm – ADD 2
  - Lance struck and broken on shield – ADD 3
- Penalty Points
  - Lance broken on saddle or below waist – DEDUCT 1
  - Lance hitting barrier, first offense – DEDUCT 2
  - Lance hitting barrier, second offense – DEDUCT 3
  - Lance hitting barrier, third offense – FORFEIT PRIZE
  - Knight killing adversary's horse must forfeit price of this horse or exchange horse at judge's discretion
  - Lance striking horse forfeits prize
  - Lance hitting barrier three times forfeits prize
- Prizes Awarded to

    - Lance striking adversary out of his saddle – HIGHEST AWARD
    - Knight upsetting both man and horse – NEXT HIGHEST AWARD
    - Knight lancing adversary’s helm three times – NEXT AWARD
    - Knight who breaks most lances – NEXT AWARD
- In a tie the knight wins who ran the fairest course, gave the greatest number of lance strokes and bore himself best.

EQUIPMENT OF THE JOUST

ARMOR OF THE JOUST

The following figure illustrates various equipment that are used in the joust.

LANCE OF THE JOUST

History of the Lance

Early lances were very short; they measured roughly 6 feet and 6 inches (Edge & Paddock, 1988, p. 30). The arrival of the 13th century brought increases in lance length; after this period, lances were rarely below 10 feet in length (Edge & Paddock, 1988, p. 30). The construction of the lance at this time consisted of one piece of solid wood (usually ash) forming the shaft and a reduction in the head size with an increase in sharpness for penetration (Edge & Paddock, 1988, p. 46). By the 14th century, the average lance length increased to 12 feet, and the lance was fitted with a vamplate and grapper (Edge & Paddock, 1988, p. 88). As time passed, the lance became larger and heavier, and by the end of the 15th century, it became unfeasible to increase the size of the lance anymore (Edge & Paddock, 1988, p. 146).

About the Lance

Although lance dimensions differ amongst individuals, there are some suggestions for general lance dimensions (Anglo, 2000, p. 245). First off, the distance between the butt of the lance and the grip should not exceed 15 inches; anything over 15 inches will cause difficulty in holding or raising the lance (Anglo, 2000, p. 245). With respect to the grip, it should have a circumference around six and a half inches (Anglo, 2000, p. 245). As for the vamplate, it should be eight and a half inches in diameter, two and a half pounds in weight, and about four inches above the grip. If the vamplate is placed too high, the shoulder is exposed to the opposing lance. If it is placed too close, the vamplate may touch the cuirass when making an encounter and may get in the way of shattering the lance (Anglo, 2000, p. 245). A final note is that in most tournaments, lances were designed to be jointed or with a hollow interior in order to embellish shattering (Edge & Paddock, 1988, p. 160).

Horse Equipment

Horse equipment generally consists of three main components: saddle, bitting, and armor. War saddles started off simple, but evolved over time to include plating ("saddle steels") that protected the thigh of the rider and a cantle that encased the hips, to prevent the rider from being forcibly dismounted (Hyland, 1998, p. 6).

Bits in the medieval period were divided into two categories: snaffles and curbs (Hyland, 1998, p. 7). Snaffles were milder on the horse than curbs, because they were jointed and acted on the tongue and bars of the horse's mouth (Hyland, 1998, p. 7). The wider the mouthpiece, the less pressure the horse will feel at any one point, and some even feature external branches to prevent the bit from being pulled through the mouth (Hyland, 1998, p. 7). Due to the harshness of curbs on the horse, they were used with indirect reining (Hyland, 1998, p. 7). Several use a double mouthpiece so metal was in contact with the roof of the mouth from the front teeth to the molars (Hyland, 1998, p. 7). Horses were ridden loose-reined, and were conditioned to follow the rider's movements, since any resistance will result in pain (Hyland, 1998, p. 8).

Skills of the Joust

Lance Play

The lance play demonstrated in tournaments of the medieval period differs from the lance play applied in actual combat. In combat, the lance is aimed at the opponent's horse instead of the knight, because the horse is not as heavily armored as the rider (Anglo, 2000, p. 229). Another difference, which does not apply to all styles of the joust, is the use of the tilt. Without a tilt, knights are able to pass much closer, which means that lance attacks are more penetrating and horse collisions are possible (Anglo, 2000, p. 227). However, with the introduction of the tilt, the angle of attack was increased, which led to a higher probability of shattering the lance. One main difference lies in the innate differences in objective between war and tournaments. In tournaments, emphasis was placed more on showiness than raw skill. Therefore, in general, the appearance of the knights was much more impressive than their results in the tournaments (Anglo, 2000, p. 229).

Training with the Lance

Individuals seeking to participate in jousts begin lance training with a light lance on foot (Anglo, 2000, p. 230). There is a gradual transition from light to heavy lance; this is to reduce the risk of injuries (Anglo, 2000, p. 231). After mastering the lance on foot, the beginner may then proceed to training with the lance on horseback (Anglo, 2000, p. 231).

Using the Lance

There are many procedures to keep in mind when handling a lance. The lance is initially supported on the thigh or in a pouch on the saddle bow (Anglo, 2000, p. 235). After moving the lance from rest, it should be lowered gradually (Anglo, 2000, p. 235). In holding the lance, the weight of the lance should be supported in the palm of the hand and not the fingers (Anglo, 2000, p. 230). Once the lance is in striking position, the hand should be directed with the eye, and the eyes should remain open until contact is made (Anglo, 2000, p. 231). Meanwhile, in striking position, the lance should not be gripped tightly, for that will result in excess vibrations (Anglo, 2000, p. 235).

Factors Affecting Lance Play

Duarte outlines four main weaknesses that must be overcome in order to succeed in the joust: not being able to see, lack of control of the lance, lack of control of the horse, and the lack of will to win (Anglo, 2000, p. 231). In each of these subcategories, he further illustrates specific scenarios.

There are several possible reasons that result in a knight not being able to see properly. Some knights do not realize that their eyes are not open at the moment of impact (Anglo, 2000, p. 231). Others are unable to keep their eyes open (Anglo, 2000, p. 231). A possible, but less common scenario is poor helm placement or poor helm design (Anglo, 2000, p. 231).

In the lance control domain, the reasons for failure are more diverse. One possible scenario is that the knight is not used to supporting the weight of tournament armor while handling the lance (Anglo, 2000, p. 231). Another possibility is that the lance is too heavy for the knight to control effectively (Anglo, 2000, p. 231). A third option stems from lack of comfort in the saddle, caused by poor saddle construction (Anglo, 2000, p. 231). A final reason is a restless horse, usually resulting from poor horsemanship (Anglo, 2000, p. 231).

Finally, the lack of will to win builds off of various psychological factors. These factors include fear, inability to meet opponents in an encounter, and being overly anxious to engage the opponent (Anglo, 2000, p. 231).

Horsemanship

Reasons for Skilled Horsemanship (General)

According to Duarte, there are many advantages to being a skill horseman. If, in combat, all other factors are equivalent (same weapons, same armor, same fighting ability), then the victor will be the one with better horsemanship (Duarte, Bem Cavalgar, 1438, p. 5). Setting aside combat, horsemanship also provides various advantages in tournaments. Compared to an average horseman, a good horseman will be able to have a better idea on coping with collisions, on positioning to wound the opponent, and on knowing the limits of the horse and how to take advantage of it (Duarte, Bem Cavalgar, 1438, p. 5).

Reasons for Skilled Horsemanship (in Jousts)

Duarte also mentions that there are four reasons why men do not correctly handle their horses during jousts: poor rein control, poor reins, other worries, and incompetence (Duarte, Bem Cavalgar, 1438, p. 40). In poor rein control, the jouster does not handle the reins properly, which leads to the bridles not being controlled properly, which means that the horse is left without instruction in movement (Duarte, Bem Cavalgar, 1438, p. 40). In poor reins, the reins are broken, so that any moving of the reins does not result in movement of the bridle (Duarte, Bem Cavalgar, 1438, p. 40). As for other occupation, some jousters use additional ropes to bind themselves to the horse in order to prevent dismounting, and some focus so much on these auxiliary ropes that they forget to use the reins (Duarte, Bem Cavalgar, 1438, p. 40). Finally, for incompetence, the rider's lack of knowledge in jousts result in him not directing the horse close enough to the tilt to initiate a collision (Duarte, Bem Cavalgar, 1438, p. 40).

Qualities of a Good Horseman

Duarte lists numerous qualities that a good horseman emanates, and this list can essentially be broken down into three main subcategories. The first is composure; a good horseman will always radiate confidence in countenance and posture, show elegance when riding in a saddle by disguising any difficulties, and stay strongly mounted in the horse in everything he does and faces (Duarte, Bem Cavalgar, 1438, p. 8). The second is knowledge; a good horseman knows how to handle all horse equipment, understands the horse's shortcomings and is able to reduce or correct them, and knows how to handle various tasks while mounted (Duarte, Bem Cavalgar, 1438, p. 8). The third is determination; a good horseman constantly seeks improvement, rides in various terrains, and does not fear falling from or with the horse (Duarte, Bem Cavalgar, 1438, p. 8).

Technical Research and Development

Summary

In order to make the idea of a virtual jousting simulator feasible, a significant amount of research and preliminary development to the technical aspects of the project was essential. For the extent of B-term, research and the development of primitive prototypes for various aspects of the jousting simulator occurred. These aspects primarily encompassed the user input functionality for the simulator, but also included some preliminary development in data persistence and communication techniques, visuals, collision detection, and player customization. As the prototypes and techniques involved in these technical aspects of the simulator matured, additional research and analysis was performed on the potential hardware requirements and their respective costs for the project, which was later used in the construction of a funding proposal. The culmination of all of the work completed over the course of B-term resulted in a fully functional jousting simulator prototype that utilized Nintendo Wii remote technology to control user input, a complete heraldry customizer for player customization, and a complete framework and development plan.

Technical Reports

The following is a collection of reports developed throughout the course of the technical research and development phase during B-term. Each report details the culmination of one or more weeks of research and development on a particular aspect of the jousting simulator.

Week 2 Report – Input Devices – 11/03/2008

Summary

The virtual jousting simulator to be developed for the 2008 Higgins Virtual Armory IQP will require some means by which to control the varying aspects of a virtual jouster. Several methods to capture various types of human input have been analyzed, where factors such as the level of immersion and the cost of input devices were taken into consideration. For this iteration of the project, three types of input devices were taken into consideration for the aspect of lance control: a standard optical mouse, a Nintendo Wii Remote, and a mouse that sends input based on accelerometer readings. Due to constraining resources, only the Wii remote and the standard optical mouse were tested in this iteration, although the mouse that utilizes accelerometer input should be a working component in theory. The best means by which to capture human input for controlling more auxiliary functions, such as controlling the speed of the jouster in a single pass and entering in text was determined to be by mapping keys or buttons from either the lance controller or a standard keyboard. As mapping keys and buttons to various functions is a trivial task, the aspect of lance control was highly focused on for this iteration.

Design

The initial design for the input component of this project was to create a high-level human input capturing class that would be extendable to various types of input devices. However, upon initial implementations, it was determined that the environment in which this project was to be developed exposed some complications and inefficiencies for this approach. Thus, a new approach was taken that utilizes the wrapper façade pattern, a common design pattern used in enterprise-level software engineering. This approach essentially allows for encapsulation of low-level human input by having the built-in Flash API handle the various types of mouse input, without actually knowing how it is implemented at a low level.

For the standard optical mouse configuration, the x and y coordinates of the lance are derived from the relative x and y coordinates of the mouse. For example, assuming that high-level y-axis modifications were not made, moving a standard optical mouse up on the y-axis will move the lance upwards. Similarly, moving the mouse down on the y-axis, right on the x-axis and left on the x-axis will move the lance down, right, and left, respectively.

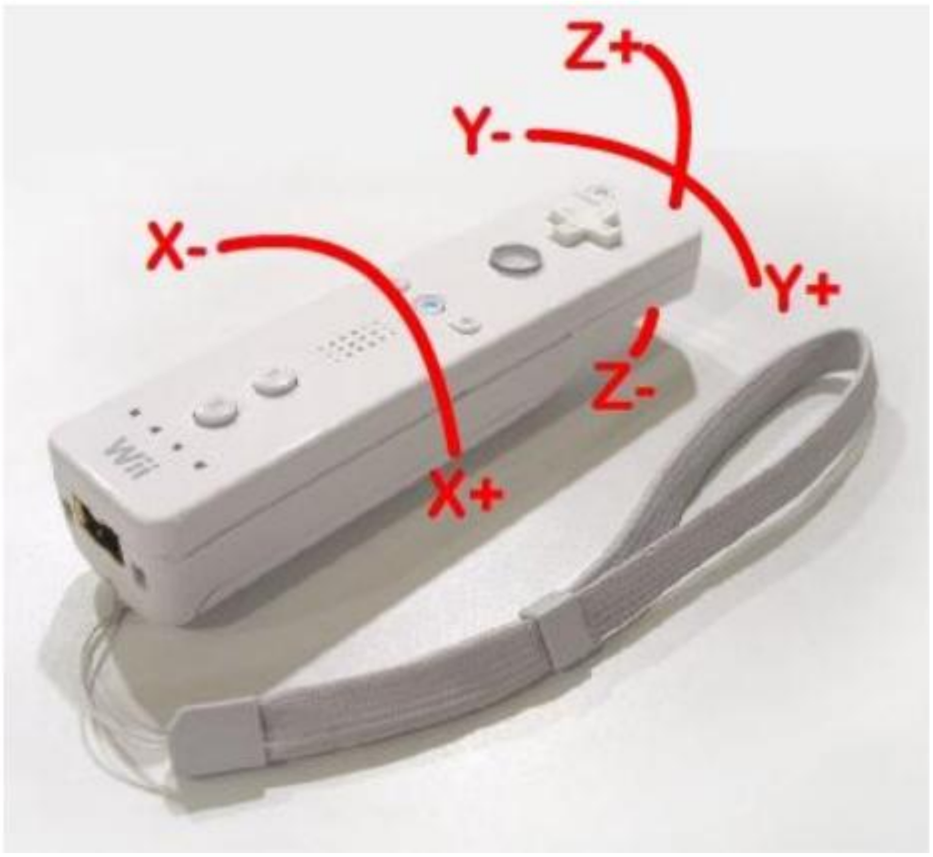
For the Nintendo Wii remote, a slightly more in-depth approach had to be taken to achieve maximum functionality and efficiency. The method used to control the lance was to read input from the accelerometer on the Wii remote by translating those values to low-level mouse coordinates through the use of the GlovePIE human interface device scripting engine. These virtual mouse movements are captured by the high-level Flash API, which interfaces directly with virtual lance movements.

Implementation

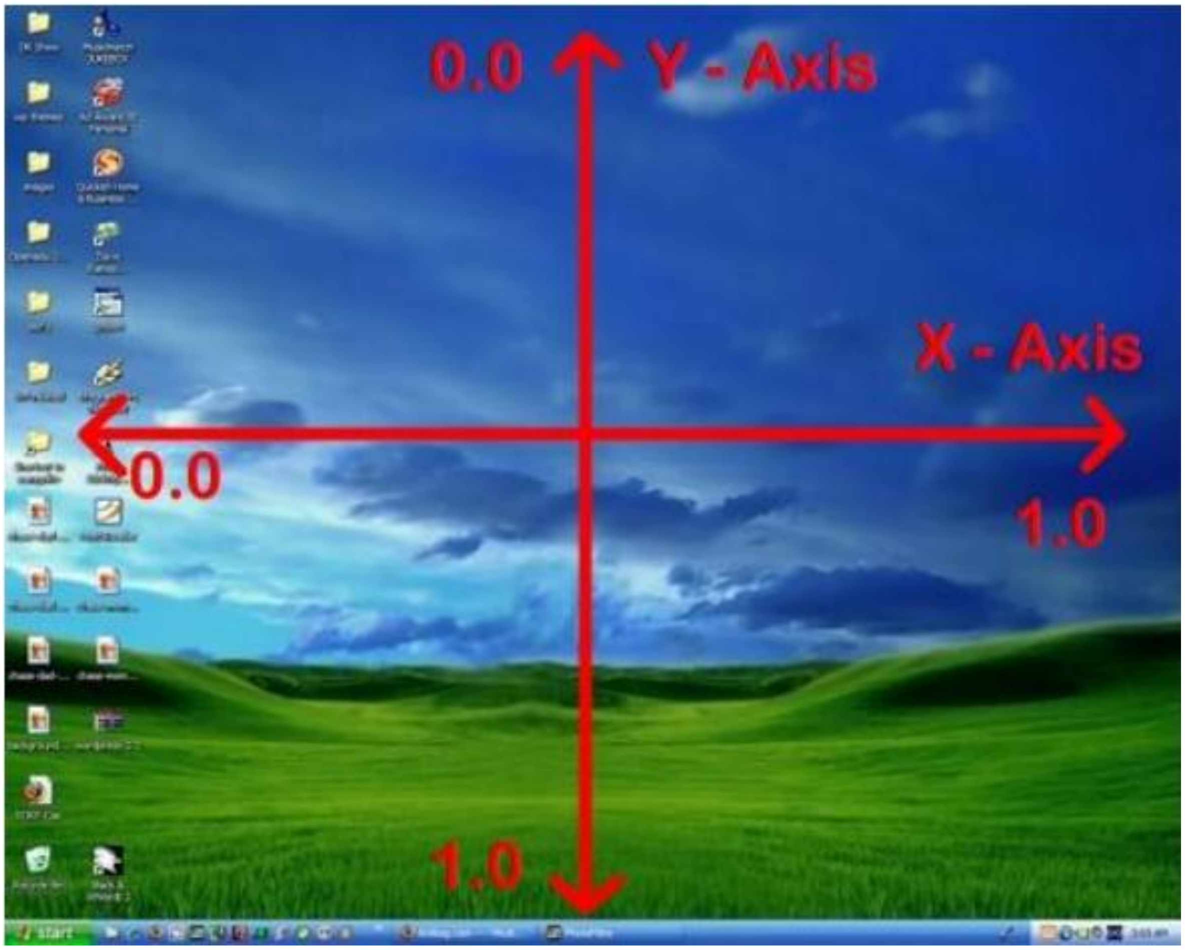
The lance controlling mechanisms were implemented by utilizing the built-in functionality of the Flash API to capture either mouse or virtual mouse movements. Capturing mouse coordinates was relatively trivial from a standard optical mouse. Mapping Wii remote movements to virtual mouse movements was more complicated, but proved to be successful in optimal hardware-related scenarios.

The standard optical mouse was connected to a laptop through a standard universal serial bus (USB) port. The Wii remote was connected to a laptop through a Bluetooth connection, using an external USB Bluetooth adapter operating on the WIDCOMM Bluetooth stack. By modifying an existing script that mapped Wii remote movements to mouse movements, a relative motion capturing method on the z-axis was scripted, so that rotating the remote on the z-axis in a specific position would translate directly to a discrete y-axis value for the mouse location. For example, tilting the Wii remote upward would tilt the lance upwards, and tilting the Wii remote downwards would tilt the lance downwards.

This relative mouse movement was accomplished by analyzing the extreme values outputted by the Wii remote when it was rotated along the z-axis and mapping these values to the y-coordinate system implemented by the GlovePIE engine. The axes of the Wii Remote have been identified in the following diagram:



By monitoring output from the Wii remote, it was determined that rotating the remote upwards on the z-axis produces values between 0.0, which is the neutral state, to 30.0, which is the approximate maximum value that the accelerometer outputs when it is in a theoretically perfect upright position. Similarly, rotating the remote downward on the z-axis produces values between 0.0 and approximately -30.0. By monitoring the value of the mouse coordinate on the y-axis when the mouse was placed at the two extremes of the screen, it was determined that the top of the screen had a value of 0.0 and the bottom of the screen had a value of 1.0. Below is a diagram detailing the coordinate system determined by these readings on a typical desktop:



Using the data gathered from monitoring the output of the Wii remote and mouse movements, a simple equation was then created to map a Wii remote position directly to a cursor position on the z-axis:

```
(Relative Mouse Position) = (((Wii Accelerometer Reading * -1) + 30) \ 60)
```

Multiplying the accelerometer reading by -1 allows a negative value to correspond to the top of the screen and a positive value to correspond to the bottom of the screen. Adding 30 to this value offsets the value so that it will always be a positive value, as needed by the mouse coordinate system. Dividing this value by 60 will then give a value between 0.0 and 1.0 that will correspond directly to a mouse coordinate. There are two methods to determine the y-position of the cursor through the Wii remote, one of which has been tested. The method that has been tested is to read values from the accelerometer on the x-axis, so whenever the remote is tilted left or right, the cursor will move left or right. The advantage of this method is that it does not require for a sensor bar to be in place, as all of the readings are taken only from the accelerometer. However, the downside to this method is that the feel of tilting the remote left and right is inconsistent with moving a lance, and therefore can feel less immersive.

The other method is to take readings from the infrared (IR) sensor in order to calculate the y-axis position. The advantage of this method is that it gives the user a greater sense of immersion, as moving the remote would be very similar to moving a lance if a relative motion capturing system was developed for the x-axis as has been developed for the y-axis. The disadvantage to this method is that a sensor bar would have to be build in order to accommodate this functionality. A sensor bar can be built by emulating a standard Nintendo Wii sensor bar by placing two IR LEDs at a specific distance from each other and keeping them lit up, so that the Wii remote can determine its relative position to the lights.

Having mapped a Wii remote to a mouse coordinate system, implementing this control scheme is relatively easy to accomplish in Flash using its built-in scripting language, ActionScript. The ActionScript code to map a visual object to the mouse coordinates is as follows:

```
Mouse.hide() // Hides system mouse cursor

<<Object>>.onMouseMove = function() // Map mouse move event handler
                                // to this custom function
{
  this._x = _xmouse; // Set x-pos of object to mouse-x-coord
  this._y = _ymouse; // Set y-pos of object to mouse-y-coord

  updateAfterEvent(); // Update object with coordinate data
}
```

As stated in the comments (un-executable text in the code preceded by "//"), the default system cursor is hidden from the screen, and a custom visual object in flash is dragged along the screen wherever the mouse cursor should be by retrieving the x and y coordinates from the mouse every time the mouse moves.

Detecting keystrokes from a standard keyboard is a fairly trivial task. The only information required in order to accomplish this is the set of key codes for each key on the keyboard, which is identical to the ASCII decimal value of a specified character. The following ActionScript code will map the "A" key to a function that produces a debug message using the trace() function:

```
eventListener = new Object(); // listener for keystroke

eventListener.onKeyDown = function() // Map the key-down event to this
                                     // custom function
{
    var keyValue = Key.getCode(); // Get code of key that was pressed

    var aKey = 65; // Key code for capital „A‟

    if(keyValue == aKey) // If the A key was pressed, show message
    {
     trace("You pressed the A key!");
    }
}
```

Demonstration

The mouse movements using both a standard optical mouse and a Nintendo Wii remote can be tested using the Flash demo provided with this week' s report. The first draft of this demo tracks the movements of either a mouse or a Wii remote in order to aim a "lance point", which is a low-tech version of what the actual virtual lance might behave like. The user can currently move the lance point over various parts of the "target dummy", to simulate possible points of impact of the lance. The x and y coordinates of the lance point are tracked in a data box in the lower right corner of the application.

** Note **

The version of the demo provided with this documentation is currently a rough draft and some functionality has not yet been implemented. All of the functionality listed above should be in working order for this draft.

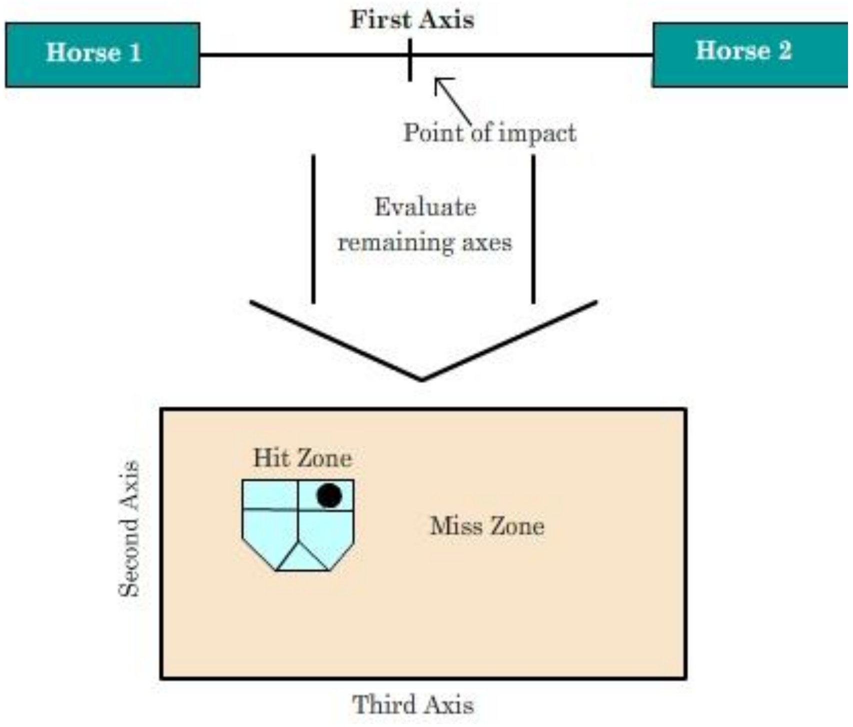
Week 3 Report – Collision Detection – 11/10/2008

Summary

The virtual jousting simulator being developed for the 2008-2009 Higgins Virtual Armory IQP will require a collision detection system in order to determine points of impact on both a micro and macro level. The macro level of this aspect involves being able to calculate the point in time in which two large objects, in this case two jousters, come into contact with each other. The micro level is more involved in the points of impact aspect of the system, where the actual point at which a jouster's lance meets his opponent must be calculated upon the "collision" of the two jousters. This collision system is intended to be implemented as a part of the overall framework for this project and is compatible with differing variables in the mechanics of the joust.

Design

The design of the collision detection system for this project is fairly simple. Although the concept of determining specific impact locations on a three-dimensional scale may seem daunting, the design approach of this system helps to break the problem down into more simplified concepts. As stated above, the system solves the problem of determining jousting collisions by diving it into two sub-problems. This first sub-problem is concerned with the point at which the two jousters come to a point of collision along a single axis, regardless of the precise point of impact. The second sub-problem deals with the remaining two axes in this three-dimensional problem, where it must determine the point on a two-dimensional plan in which the jouster hits his target. The diagram below gives a visual representation of this concept:



The sub-problem regarding a collision along a single axis is fairly simple. The overall concept of this issue is that two objects are approaching each other at their own respective speeds. In order to simplify the design of this system, a set of incremental, arbitrary speeds are defined for each jouster, ranging from 1 to 10. The unit of measure for these speed values is irrelevant, which will be explained in the implementation section of this document. This set of speeds is equivalent for each jouster, where a speed of X for one jouster will be equivalent to a speed of X for another jouster. Similarly, a speed of 2 for Jouster A will be slower than a speed of 5 for Jouster B. These speed values are then used to determine how quickly each of the jousters will be able to travel from one location to another. Since both of these jousters are traveling along the same axis towards each other, they will be destined to meet at some point, which is where the next sub-problem comes into play.

Having solved the single-axis problem, the second component of the collision detection system determines the point along the remaining two axes upon which a single jouster strikes another jouster. Having a standardized plane with regions specifically assigned to the various strike points of another jouster allows for a comparison to take place between the location of the virtual lance at the time of impact and an assigned hit point. In other words, when the collision on the first axis is detected, the point of impact is subsequently determined on the remaining two axes. When the two-axis point of impact has been solved, the combination of the two sub-problems produces an evaluated three-dimensional point of impact.

Implementation

The first iteration of the collision detection system was implemented in Flash by utilizing the built-in scripting language, ActionScript. The various speed constants that were defined for the two jousters were used to modify the time it would take for two abstract data objects to collide with each other. When these two objects create a virtual collision, an event is triggered to retrieve all data from the first person perspective, such as where the lance was positioned in relation to the opposing jouster. This approach not only allows for a three-dimensional point of impact to be evaluated, but also for other useful information to be evaluated as well, such as how well the user controlled the virtual horse.

It was noted earlier that the speed constants used on the first axis for the abstract data objects did not have a unit of measure and were in fact, irrelevant. This is due to the fact that the variable of distance in this setting is unneeded. Since the only perspective that the user has during the actual joust is in the first person, the concept of distance is essentially faked in order to give the illusion of moving forward. Therefore, a simple timed event can be used, which determines how fast an opposing jouster approaches the jouster and likewise, how fast the jouster approaches the opposing jouster. In this case, the speed constants were used to modify a set amount of time that it would take for each jouster to reach each other's starting positions.

Using a constant speed and modifying it also allows for Flash's innate tweening functionality to be used to simulate acceleration. Motion tweens are used in Flash in order to automatically animate an object to move from one location to another. Various modifiers can be incorporated into these tweens that allow for acceleration effects and elasticity if needed.

Having a timed event to represent the collision on the first axis allows for the data from the second and third axes to be retrieved fairly easily. When an event is triggered after the first collision, the position of the virtual lance can be retrieved, as well as any objects that it is in contact with. This allows for a precise point of impact to be determined at the point of collision. Other data can also be collected at this point in time as well as described above. Having all of this data collected at the point of collision will allow for a plethora of game mechanics to take form, such as run-time probabilistic evaluations to determine if a user has knocked the opponent off his horse, or vice versa.

Demonstration

The demonstration that shows the functionality of the collision detection system in this iteration makes use of a new development interface that allows the user to actually see the abstract data objects approaching each other at given speeds. When the objects are given the instruction to approach each other, the user can freely use the game controls to simulate a joust on a dummy target. Once the two data objects collide, data is retrieved from the lance and from the horse controlling mechanism. This data is then reported in a message box. Ideally, this data would later be used to determine game mechanics as described above.

Week 4 Report – Infrared-Based Virtual Lance Control Implementation – 11/17/2008

Summary

The virtual jousting simulator being developed for the 2008-2009 Higgins Virtual Armory IQP will require a means by which to control the player's virtual lance. One of the method proposed involved using a Nintendo Wii Remote to capture human input and subsequently manipulate the virtual lance in order to best emulate controlling a real lance. As stated in the "Input Devices" document, there are two means by which to read user input from a Wii Remote, which are by taking accelerometer readings and by taking infrared readings. This document will discuss the latter of the two, as these user input readings appeared to be the most stable from initial experiments.

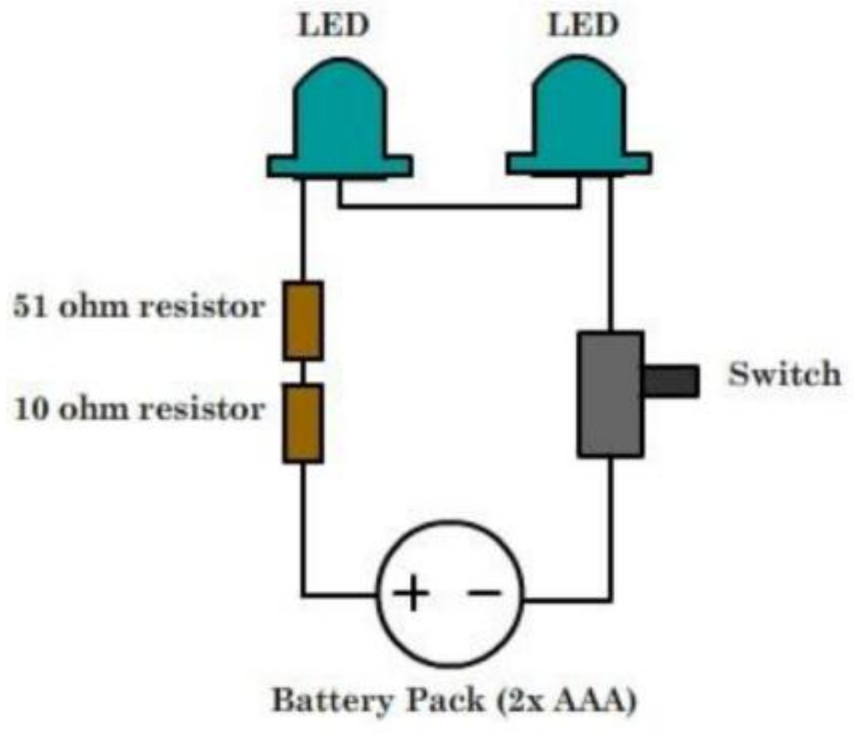
Design

The design approach to the infrared controller method utilizes the field of vision aspect of the virtual jousting simulation. In order for infrared readings to be taken in from the Wii remote to be converted to mouse movements, the infrared sensor must be in visual contact with two sources of infrared light at all times. The positioning of the infrared lights relative to the Wii Remote can allow for a fairly realistic control scheme to be implemented. If a user were actually holding a lance, there would be a certain range in which the user would be able to move the lance in front of him and still be able to see the lance point. This range can be emulated by the infrared configuration by positioning the two infrared sources close enough to the Wii Remote to the point where if the user were to rotate the Wii Remote to a point where it could no longer find the infrared sources, then virtual lance point should be at a relative position on the sides of the screen. By doing so, the user should be able to feel as though the virtual lance is in a position where the user would expect to find a lance point if he were in fact holding a lance.

This design approach could also be taken a step further by allowing the virtual lance position to be relative to where the virtual jouster's head is looking. Whenever the user causes the virtual jouster to look in a particular direction, the two infrared sources could be moved in the opposite position, so that the virtual lance position is not always dependent on where the virtual jouster is looking. By using this method, the user would essentially be able to look in different directions without the lance being moved as well.

Implementation

This control scheme was implemented using the human input scripting engine described in the "Input Devices" document and by constructing two sources of infrared light for the Wii Remote to be able to take in infrared readings. Each source was created using two high-output 5mm infrared LEDs, a 51 ohm resistor, a 10 ohm resistor, a standard slide switch, a AAA battery holder with a two-battery capacity, and 2 AAA batteries. These components were soldered together in the configuration detailed in the diagram below:



Each source has the two LEDs in very close proximity to each other in order to intensify the infrared light output and to not allow for the Wii Remote to be able to be read the two LEDs as two separate sources. Having more than two infrared sources would essentially "confuse" the Wii Remote infrared detector and would cause unexpected data output to occur. These two sources can be placed in any configuration by simply moving them into various locations. This allows for experimenting with different configurations to be as efficient as possible.

Demonstration

The latest demonstration that was developed for this project is fully compatible with the infrared reading from the Wii Remote. In order to move the lance with the Wii Remote, the remote must be able to detect both sources of infrared light. The infrared sources must also be emitting infrared light. Moving the switch on the source to the position closest to the LEDs will ensure that the lights are on. It should be noted that the human eye cannot detect infrared light as well. To determine if there is power to the two LEDs, a person can look through a digital viewfinder on either a camera or a phone that is pointed at directly at the LEDs. Most digital viewfinders will able to make the infrared light visible.

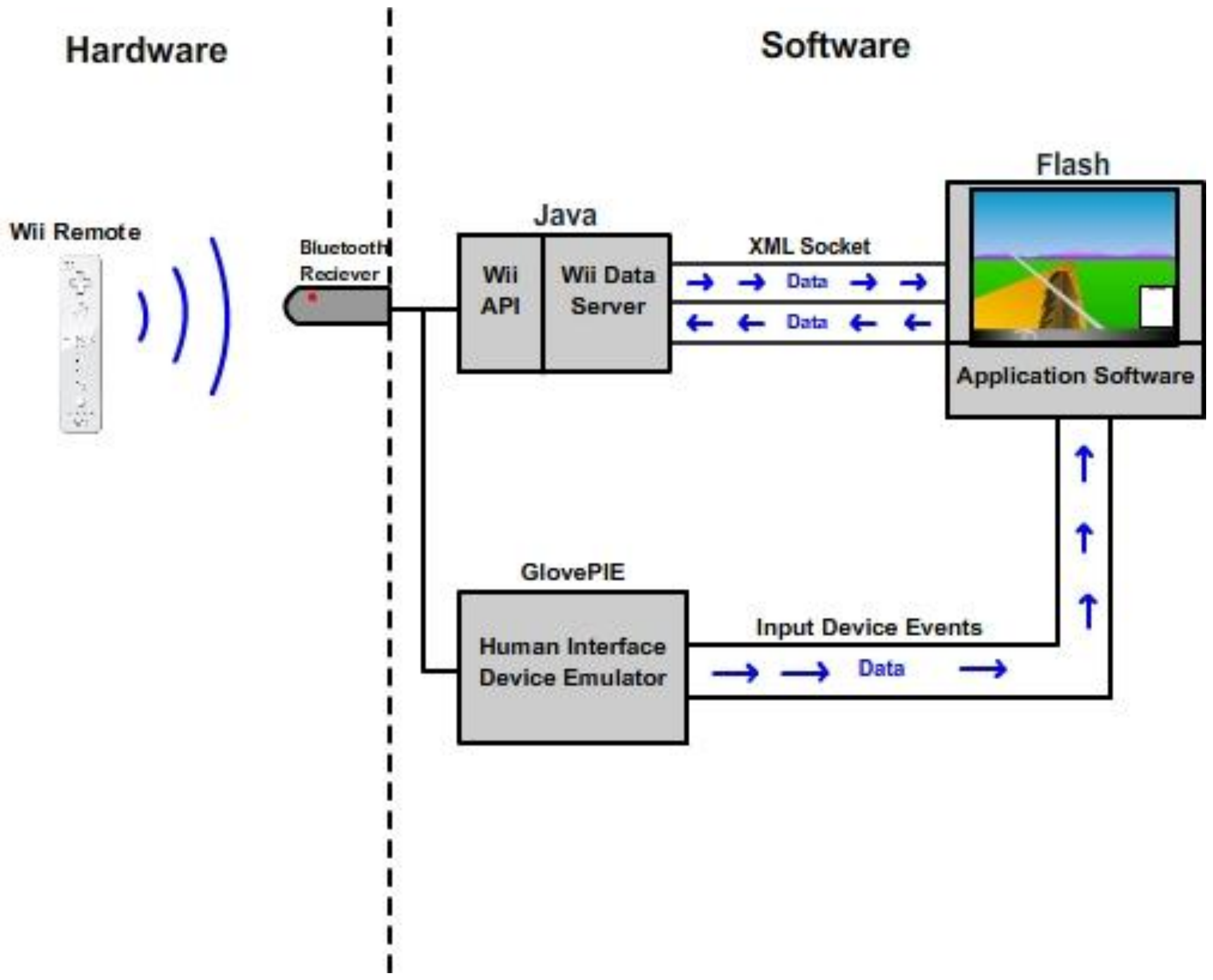
Week 7 Report – Wii Remote Data Communication over an XML Socket – 12/16/2008

Summary

The current approach to capturing Wii Remote data uses third-party software to map Wii Remote data to input device events (i.e. mouse movements and keystrokes). Although sufficient for most of the requirements for the Higgins Virtual Armory IQP, this approach has some limitations. One limitation is that the application software only has a one-way interface with the Wii Remote. The application software has the ability perform actions based on data received through emulated input device events, but it cannot communicate back to the input device, which rules out some abilities such as telling the Wii Remote to rumble or to play a certain sound using its speaker system. Another limitation is that there are limited possibilities by which Wii Remote data can be processed. A simple example of this is that since there are only two axes on a mouse, only two axes can be monitored on the Wii Remote at a single point in time. The solution to these two problems is to establish a means of communication between the Wii Remote and the application software so that all Wii Remote data can be seen at all times and so the application software can interact with the Wii Remote if necessary. One way that this solution is implemented is using a client-server setup that utilizes the XMLSocket object in Flash and the networking capabilities of Java.

Design

The design approach to this concept involves creating a client-server setup that allows for the application software to have access to the Wii Remote data without having to directly interface with the Wii Remote. The benefit to this approach is that the application software does not have to worry about varying hardware changes and configurations with the Wii Remote and Bluetooth setup and can expect the same data at all times. This allows for the application software to be loosely coupled from its system-level backend , which makes code maintenance a much easier task in the long-run. The diagram below gives a basic outline of how this system is laid out from a more abstract perspective and includes a comparison between sending data to the application software through input device events and over a socket in a client server setup:



The server in this setup basically has two jobs when dealing with sending data to the client, which in this case would be the application software. The first job is to notify its client if any new data has been received from the Wii Remote over the Bluetooth connection. The second job is to send the latest copy of any Wii Remote data that is requested by the client software. This approach ensures that the client is always aware of data updates, but only receives data when it requests it. The server also acts as a middle ground between the client and the Wii Remote, where the client can send data to the server that can be sent to the Wii Remote to perform actions such as activate the onboard LEDs.

The job of the client is relatively simple in this case. It only needs to register itself with the server to receive updates whenever the Wii Remote data has changed and must be able to receive data from the server when it makes a request for it.

Implementation

The server for this system was developed using Java. The Wii Remote interfacing side of the server was developed using the WiiremoteJ application programming interface (API). The networking component was developed using Java's networking components. The WiiremoteJ API simplifies the interface between the server and the Wii Remote by being able to map data change events from the Wii Remote to custom events specified by the server. Essentially, all of these event handlers are mapped directly to update signals that will be sent to the client. The server will also keep an updated copy of all the data that has been received from the Wii Remote so that it is readily available for any clients that request it.

The networking side of the server is slightly more complicated. This component is basically driven by sockets, which are the end points to a connection between the client and the server. Initially, a certain type of Java socket, called a ServerSocket, is instantiated to listen for connection requests by any clients that want to connect. Whenever a client decides to connect, a second socket is created that sets up a means of communication between the client and the server through data streams. Data can then be sent both ways over the data streams, which can be parsed and acted upon by both the client and the server.

Since the client in this case is a Flash-based application that uses the ActionScript 2.0 scripting language, the only means of communication to outside processes that the client can utilize is a specific type of socket, known as an XMLSocket. This type of socket basically works the same way as normal sockets would, with the exception that data must be sent using XML formatting. All messages must also be terminated by a 0 byte. An example valid message sent to an XMLSocket signifying acceleration in the X and Y directions on the Wii Remote accelerometer would be as follows:

```
<XAcceleration>304.2</XAcceleration><YAcceleration>20.2</YAcceleration>\0
```

Using this messaging format, the server can send any Wii Remote data and update signals to the client whenever it is requested.

Project Proposal

Overview

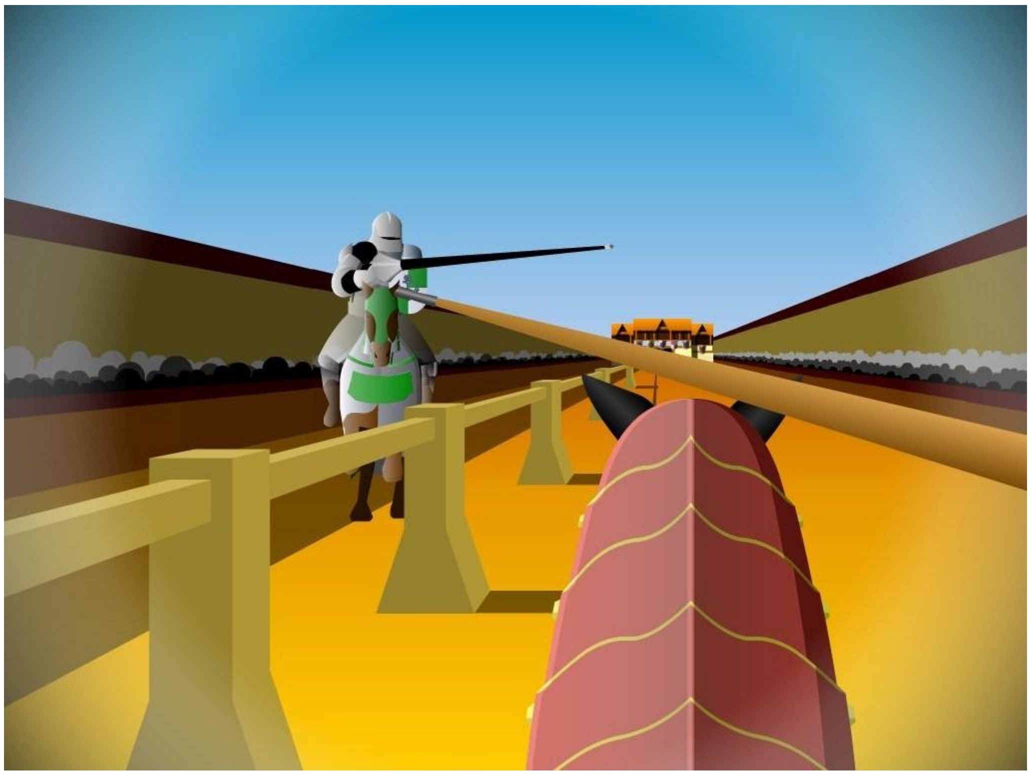
The following is the result of the research put into the possible hardware requirements and costs of the virtual jousting simulator. This document was developed by Patrick Newell and Professor Jeffrey L. Forgeng.

<u>Higgins Armory "Virtual Joust" Project 2008-09</u>

A collaboration between
Worcester Polytechnic Institute
and
the Higgins Armory Museum

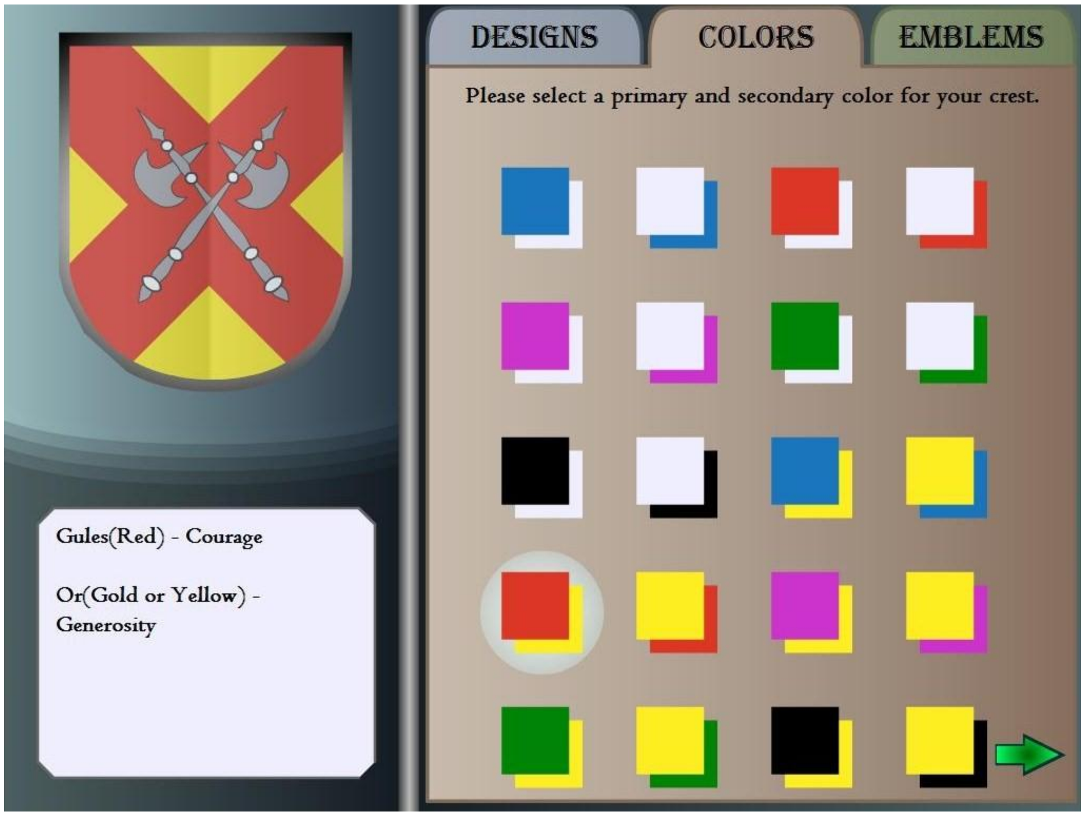
Project Description

Beginning in Fall 2008, a team of four Worcester Polytechnic Institute students has been developing a "virtual joust" experience for installation in the public galleries at the Higgins Armory Museum. Visitors will interact with a Flash-based game through a Nintendo Wii Remote embedded in a shortened medieval-style lance, using a second Nintendo Wii Remote to represent interaction with the jouster's horse.



Virtual Joust (Full demonstration build Apr. 09)

The game's introductory section allows visitors to select their own heraldic colors and coat-of-arms. The visitor's heraldic design will be procedurally incorporated into the game's visuals. Internet access will allow visitors to develop their heraldic designs offsite before and after their onsite experience. Long-term development of this game will allow for building a full online community of Higgins "virtual jousters."


DESIGNS
COLORS
EMBLEMS
Please select a primary and secondary color for your crest.
Gules(Red) - Courage
Or(Gold or Yellow) - Generosity

Heraldry Selector (Full demonstration build Apr. 09)

This project was completed in May 2009, which includes a fully-functional Jousting Simulator that can utilize an interface consisting of two Nintendo Wii Remotes. Further projects will be planned in future academic years to build on the success of this final pilot-version.

Technical Requirements

The project plan calls for duplicates of all essential hardware to provide backup as well as allowing offsite demonstrations.

Total Costs:
Minimum Requirements: approx. $7326
Maximum Requirements: approx. $8952

Server and Client Platform

Two main desktop PCs are called for in the plan, one to run the joust game itself, the other functioning both as a separate kiosk for the introductory (heraldry) section of the game, and as a server to support Internet access to this part of the game.

Processor	Multi-Core: 3.0 GHz (Server) 2.4 GHz (Client)
	Single-Core: 1.8 GHz (Server) 1.8 (Client)

RAM	2 GB
Hard Drive	40 GB
Video Card (Video RAM)	256MB (Server) 128MB(Client)
Sound Card	2 Channel Audio
Networking	High-Speed Ethernet Port
Optical Drive	DVD-ROM

Device	**Specs**	**Price (Minimum)**
Laptop (Demo System)		$550
Client PC		$400
Server PC	Single dedicated tower server running open-source server software	$900
Windows Vista Home Basic*	One for each unit (3 total)	$200**
Modem		$40
Router		$40
Networking supplies		$50
Total		*$2580 ($1330)****

* This project has been primarily tested on the Windows Vista Business and Windows Vista Ultimate platforms. Windows XP should also be able to support this project, although it has not been formally tested and is no longer easily available for purchase.

** This is the price directly from Microsoft. Having the operating system pre-loaded on a pre-manufactured system tends to be lower in price.

*** Desired configuration is Server + Client, plus Laptop for offsite demonstration. Minimum is 2 client-level PCs.

Note:
Pre-manufactured desktop platforms from companies such as Dell, HP, or Compaq tend to have good prices for systems similar to that needed for this project. An example desktop customized on the Dell website is detailed in the print-out below: (Price: $369)

DELL™

Print this page Close

Dell recommends Windows Vista® Business. SHARE

Print Summary

Vostro 200n Mini Tower

Starting Price **$369**

As low as **$10**/mo.[1]

Learn More | Apply

Preliminary Ship Date: **12/6/2008**[2]

Great Vostro Desktops starting at $319.
Shop Now

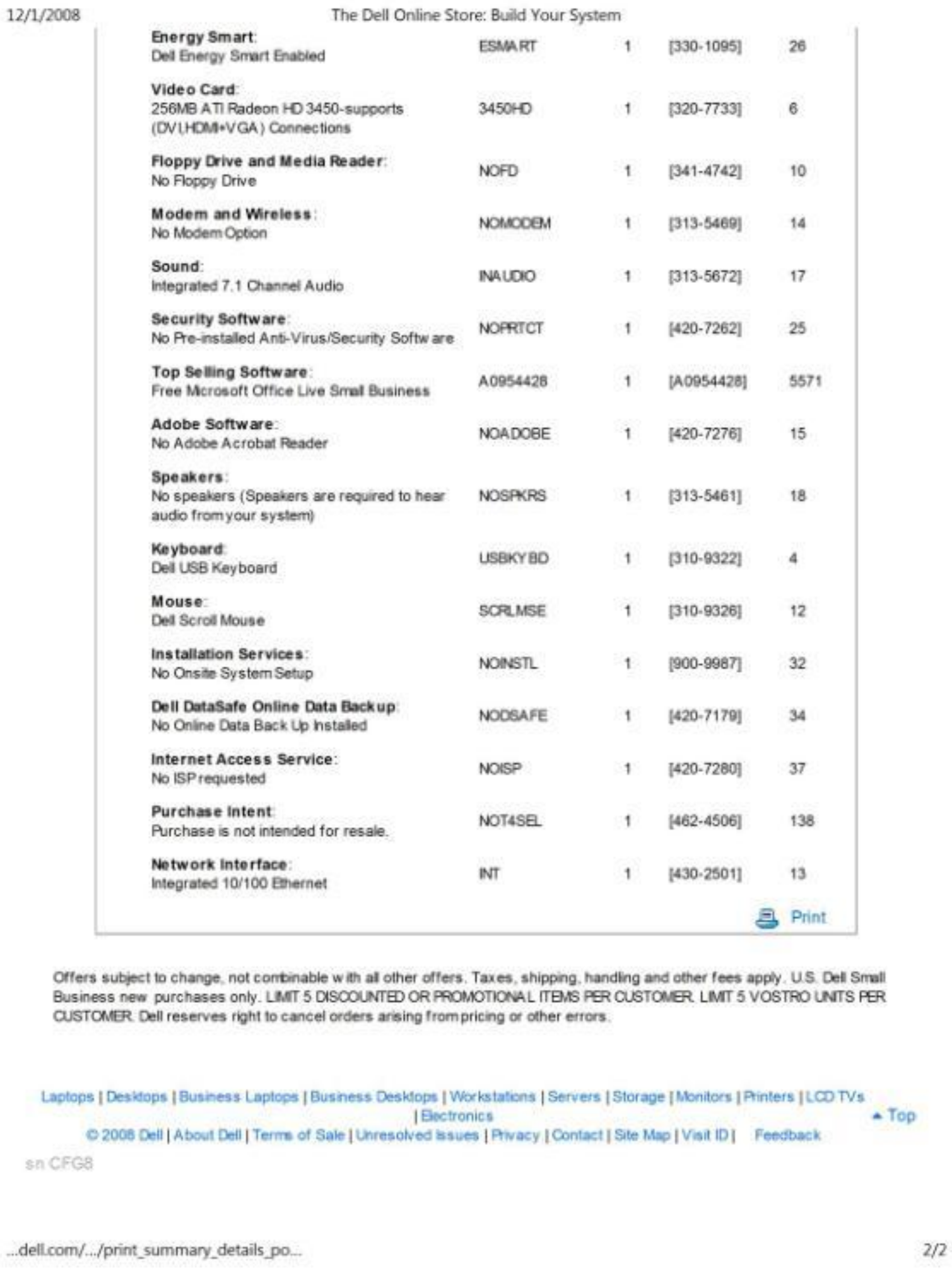
Color Laser only $199 with system purchase!
Limited time offer.
More Details

My Selections All Options

• Vostro 200n Mini Tower

Date	12/1/2008 12:01:23 PM Central Standard Time
Catalog Number	4 Retail 04

Catalog Number / Description	Product Code	Qty	SKU	Id
Vostro 200: Intel® Pentium® Dual-Core E2200 (2.20GHz,1MB L2Cache,800FSB)	2200N	1	[223-6791]	1
Operating System: FreeDOS™ included in the box, ready to install	FREEDOS	1	[313-5468][420-7235]	11
Office Productivity Software (Pre-installed): No Pre-installed Productivity Software	NOSFW	1	[420-7281]	22
Warranty & Service: 1 Year Basic Limited Warranty and 1 Year NBD On-Site Service	Q1YOS	1	[983-4250][986-5280][988-7347][989-5897][991-2878]	29
Monitor: No Monitor	NOMON	1	[320-5671]	5
Memory: 2GB Dual Channel DDR2 SDRAM 800MHz - 2DIMMs	2GB800	1	[311-7366]	3
Optical Drives: Single Drive: 16X DVD-ROM Drive	16XDVD	1	[313-5454]	16
Primary Hard Drive: 80GB Serial ATA Hard Drive (7200RPM) w/DataBurst Cache™	80GB72K	1	[341-4988]	8



Energy Smart: Dell Energy Smart Enabled	ESMART	1	[330-1095]	26
Video Card: 256MB ATI Radeon HD 3450-supports (DVI,HDMI+VGA) Connections	3450HD	1	[320-7733]	6
Floppy Drive and Media Reader: No Floppy Drive	NOFD	1	[341-4742]	10
Modem and Wireless: No Modem Option	NOMODEM	1	[313-5469]	14
Sound: Integrated 7.1 Channel Audio	INAUDIO	1	[313-5672]	17
Security Software: No Pre-installed Anti-Virus/Security Software	NOPRTCT	1	[420-7262]	25
Top Selling Software: Free Microsoft Office Live Small Business	A0954428	1	[A0954428]	5571
Adobe Software: No Adobe Acrobat Reader	NOADOBE	1	[420-7276]	15
Speakers: No speakers (Speakers are required to hear audio from your system)	NOSPKRS	1	[313-5461]	18
Keyboard: Dell USB Keyboard	USBKYBD	1	[310-9322]	4
Mouse: Dell Scroll Mouse	SCRLMSE	1	[310-9326]	12
Installation Services: No Onsite System Setup	NOINSTL	1	[900-9987]	32
Dell DataSafe Online Data Backup: No Online Data Back Up Installed	NODSAFE	1	[420-7179]	34
Internet Access Service: No ISP requested	NOISP	1	[420-7280]	37
Purchase Intent: Purchase is not intended for resale.	NOT4SEL	1	[462-4506]	138
Network Interface: Integrated 10/100 Ethernet	INT	1	[430-2501]	13

Print

Offers subject to change, not combinable with all other offers. Taxes, shipping, handling and other fees apply. U.S. Dell Small Business new purchases only. LIMIT 5 DISCOUNTED OR PROMOTIONAL ITEMS PER CUSTOMER. LIMIT 5 VOSTRO UNITS PER CUSTOMER. Dell reserves right to cancel orders arising from pricing or other errors.

Laptops | Desktops | Business Laptops | Business Desktops | Workstations | Servers | Storage | Monitors | Printers | LCD TVs | Electronics
▲ Top
© 2008 Dell | About Dell | Terms of Sale | Unresolved Issues | Privacy | Contact | Site Map | Visit ID | Feedback

sn CFG8

Input Devices

Device	**Specs**	**Price (Minimum)**
Optical Mouse	2 Buttons, 1 Wheel, USB or PS/2	$10
Keyboard	US Layout, USB or PS/2	$20
USB Bluetooth Adapter*	Targus ACB10US	$40
Nintendo Wii Remote		$50
Wii Remote Rechargeable Power Pack		$20
4 High-Output Infrared LEDs		$8
2 DPDT Slide Switches	Rated 0.3A at 125V AC	$7
2 60 Ohm Resistors		$1
AC/DC converter for LEDs		$20
Nintendo Wii Balance Board**		$120
Total		*$602 ($326)****

* Wii Remote functionality has only been tested with this particular USB Bluetooth adapter. Other adapters that are compatible with the software and hardware required for Wii Remote functionality have been tested by various third parties. The specific devices and their respective prices can be found at the following website: http://www.wiili.org/index.php/Compatible_Bluetooth_Devices

** Appears to only be able to be purchased with Wii Fit.

*** Minimum is optical mouse and keyboard for all units, plus 1 Wii interface system. Preferred is 2 Wii interface systems.

Display

Device	**Specs**	**Price**
Large Touch Screen Monitor	20”, 1366x768 px resolution	$900
Small Touch Screen Monitor	15”, 1024x768 px resolution	$300
Total		$1200

Sound

Device	**Specs**	**Price (Minimum)**
2.1 Speaker Configuration	2 Speakers, 1 Subwoofer	$50
Sound Dome	20” Diameter	$600
Total		*$100 ($1200)**

* Desired is 2 sound domes. Minimum is 2 speaker systems.

Software (Development and Maintenance)

Item	**Specs**	**Price (Minimum)**
Flash CS4 Professional		$700
Total		*$700 ($700)*

Kiosk

Item	**Specs**	**Price (Minimum)**
Jousting helmet	Great basinet style from Jeff Wasson	$2500
Lance stub	From Historic Enterprises	$100
Vamplate	From Historic Enterprises	$70
Materials for 2 kiosks; backup power source; security cable		$1100
Preparator/building mgr work on 2 kiosks	1 week of work required to produce each kiosk	NEED THIS FIGURE
Total		*$3870 ($3770)**

* Desired figure includes a 2nd lance stub in case of breakage.

Conclusion

Review of Interactive Qualifying Project

After determining our project path through research on the Higgins Armory Museum and numerous meetings with museum staff, significant research was done using the museum's resources and outside sources in order to acquire the necessary background to complete the project. For seven weeks, two team members worked on acquiring the information needed on the historical aspects and mechanics of the joust. During the same seven weeks, the other two team members investigated implementation routes and design details to develop a simple simulation of the joust. Following this initial research phase, work was completed through weekly iterations. Each of these iterations focused on at least one of the following major areas: user interface, game content, or game balancing. After delivering a beta version of the game, significant testing and revising was carried out within the team in order to find bugs and check the usability of the user interface. Numerous changes occurred due to this internal testing resulting in our released product.

Analysis of Development Process

The project was broken down into three stages. These stages were the requirements gathering, research and initial development for the product, and the creation of the product itself. During each stage, we used the appropriate techniques in order to successfully complete that stage with what we needed in order to move onto the next stage. Throughout the entire project, we kept open channels of communication with our sponsor to ensure that we were going to provide them with the product that they wanted.

Future Work

The released version of our product provides the user with a good feel of what the joust was like. As such, we completed the primary goal of the project, which was to create a game that would simulate the feel of jousting. While the base product is playable and fun, there are areas in which the program could be expanded. These enhancements would further increase the realism of the game or make it more widely available to others.

The first area that future work could be performed in is in the game itself. Suggestions for future projects that build off of this one include using a 3D graphics engine to actually implement 3D graphics for game play, giving the game multiplayer capabilities, adding a role-playing game container, or porting the game for online play. We feel that any of these would be good continuations of the product that would enhance the experience of the users. Items such as the 3D graphics engine and multiplayer support would make the joust itself more realistic and interesting for the players. Online play, along with a role-playing game container would make a great future project as it would allow more people to play the game, generate traffic on the museum website, and interest people into visiting the museum to play the game using the lance. The other aspect that could have future work done on it is the hardware.

Currently, the game is played using a lance and a set of reins on a pole. This will give the user a small taste of what it is like to be a knight but it can be better. In this area, future projects are making the horse control more realistic and making a station for the game to be played in. Both of these projects would be using the hardware to enhance the realism of the current simulation.

References

Anglo, S. (2000). *The Martial Arts of Renaissance Europe.* New Haven: Yale University Press.

Anglo, S., & Wagner, A. (1968). *The Great Tournament Roll of Westminster.* Oxford: Clarendon Press.

Barber, R., & Barker, J. (1989). *Tournaments Jousts, Chivalry and Pageants in the Middle Ages.* New York: Weidenfeld & Nicolson.

Blair, C. (1958). *European Armour: Circa 1066 to Circa 1700.* London: B.T. Batsford LTD.

Clephan, R. C. (1919). *The Tournament Its Periods and Phases.* London: Methuen & Co. LTD.

Couch, D. (2005). *Tournament.* New York: Palgrave Macmillan.

Cripps-Day, F. H. (1918). *The History of the Tournament in England and in France.* London: Quaritch.

Cripps-Day, F. (1912). *The Triumph holden at Shakespeare's England on the eleventh day of July in the third year of the reign of King George the Fifth.* London: W.H. Smith and Son.

D'Anjou, R. (1946). *Le Livre Des Tovrnois.* Paris.

Duarte. (1438). *Bem Cavalgar.* Portugal.

Edge, D., & Paddock, J. M. (1988). *Arms & Armor of the Medieval Knight.* Greenwich: Brompton Books Corp.

Emmet, W., Biernacki, B., Nicoll, K., Kozlowski, A., Jennings, C., & Westcott, P. (1979). *The Tournament (Worcester Polytechnic Institute).* Worcester: Worcester Polytechnic Institute.

Hyland, A. (1998). *The Warhorse 1250 - 1600.* Great Britain: Sutton Publishing Limited.

Kottenkamp, D. F. (1857). *History of Chivalry and Ancient Armour.* London: Willis & Sotheran.

Program of Tourney at Avon. (1912). England.

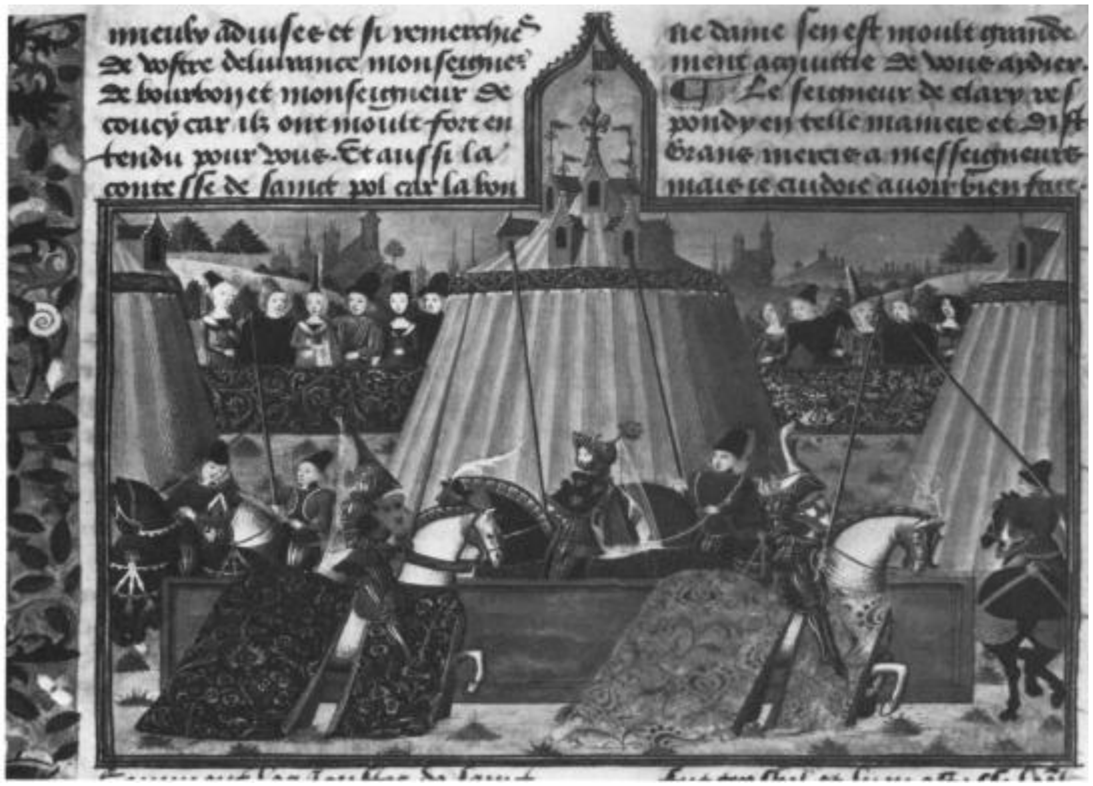
Randolph, A. (2002). *Engaging Symbols: gender, politics, and public art in fifteenth-century Florence.*

Trentsensky, M. *Exercises in Coloring The Tournament.* London: Joseph Myers & Co.

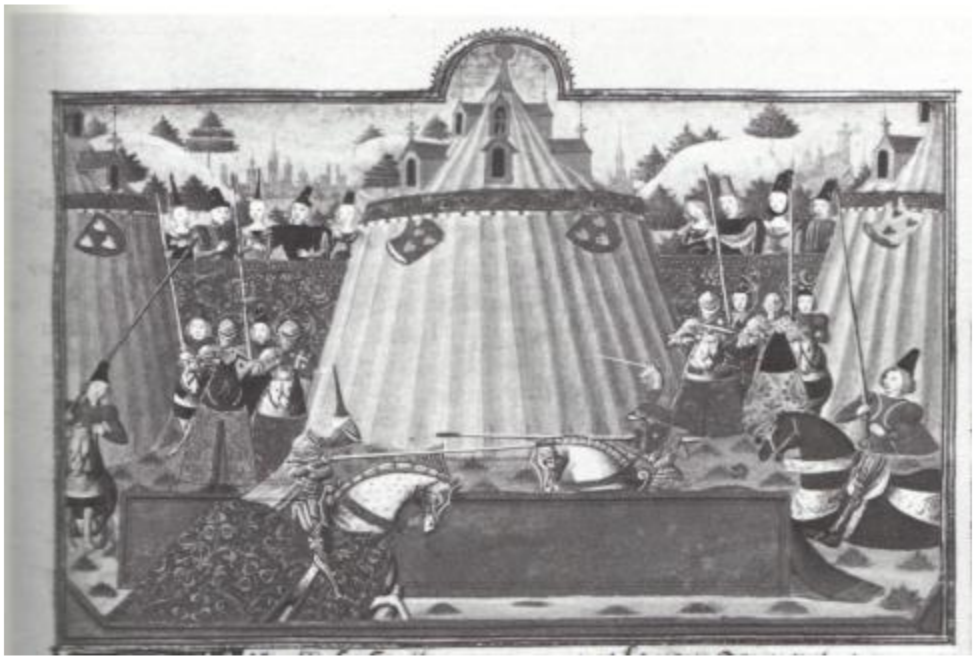
Young, A. (1987). *Tudor and Jocobean tournaments.* New York: Dobbs Ferry.

Appendices

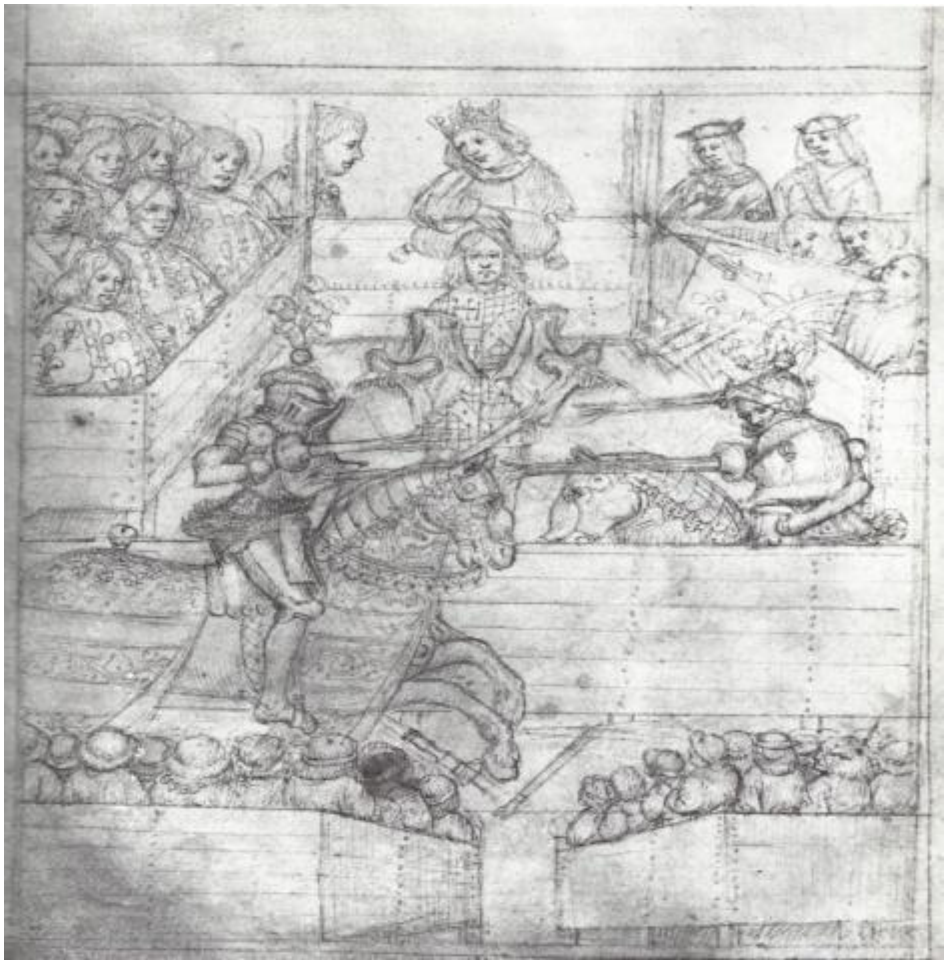
Appendix I: Pictures of the Tournament


mieulx adviſee et ſi remerchiés
de voſtre deliurance monſeigneur
de bourbon et monſeigneur de
coucy car ilz ont moult fort en
tendu pour vous. Et auſſi la
conteſſe de ſainct pol car la bon
ne dame ſen eſt moult grande
ment acquittie de vous aydier.
Le ſeigneur de clary reſ
pondy en telle maniere et diſt
grans mercis a meſſeigneurs
mais je cuidoie avoir bien fait

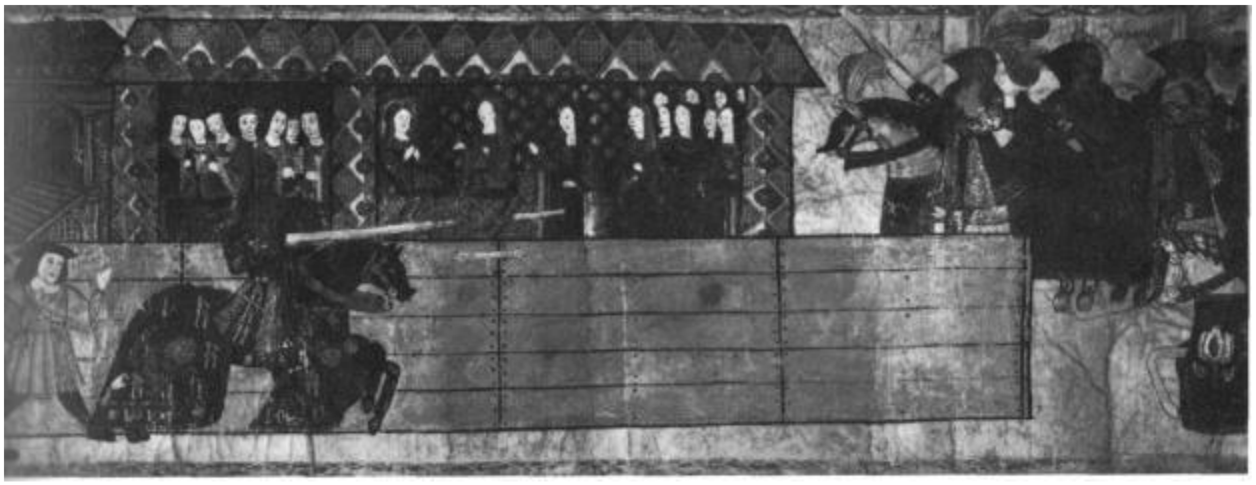
Knights parading around the lists before a tournament that occurred between 1450 and 1475



Knights jousting during a tournament that occurred between 1450 and1475

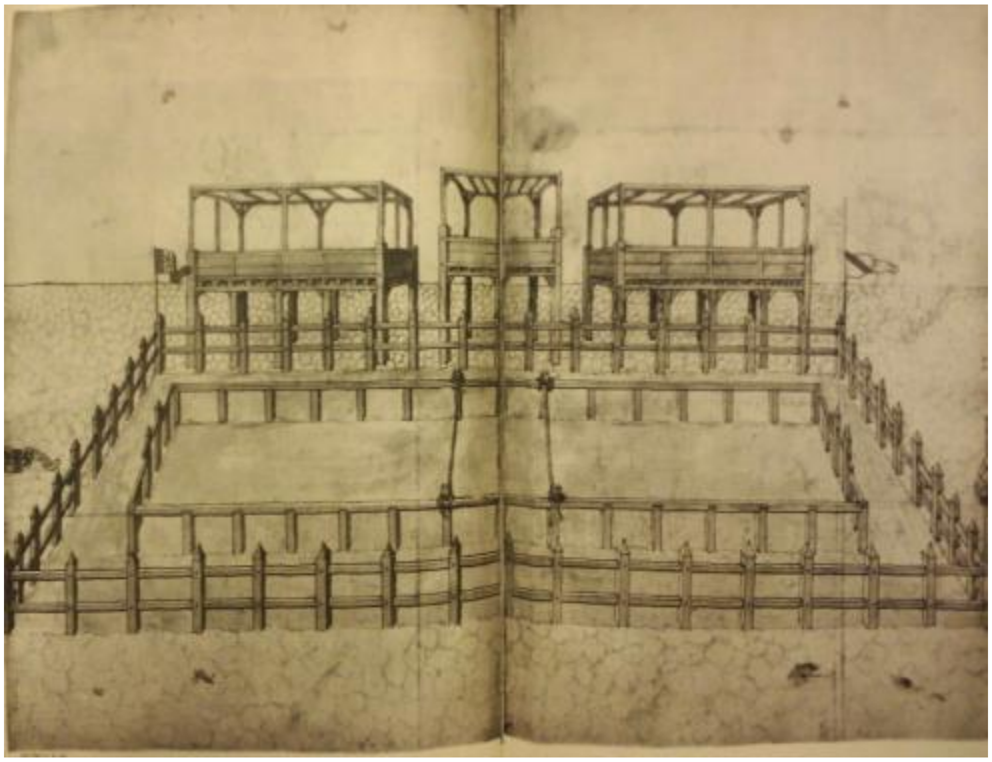


Two knights jousting in the lists during the 1450-1475. This image also shows us the placement of the stands that surrounded the lists.



Ladies watch a joust during 1515

Example of heraldry used during tournaments



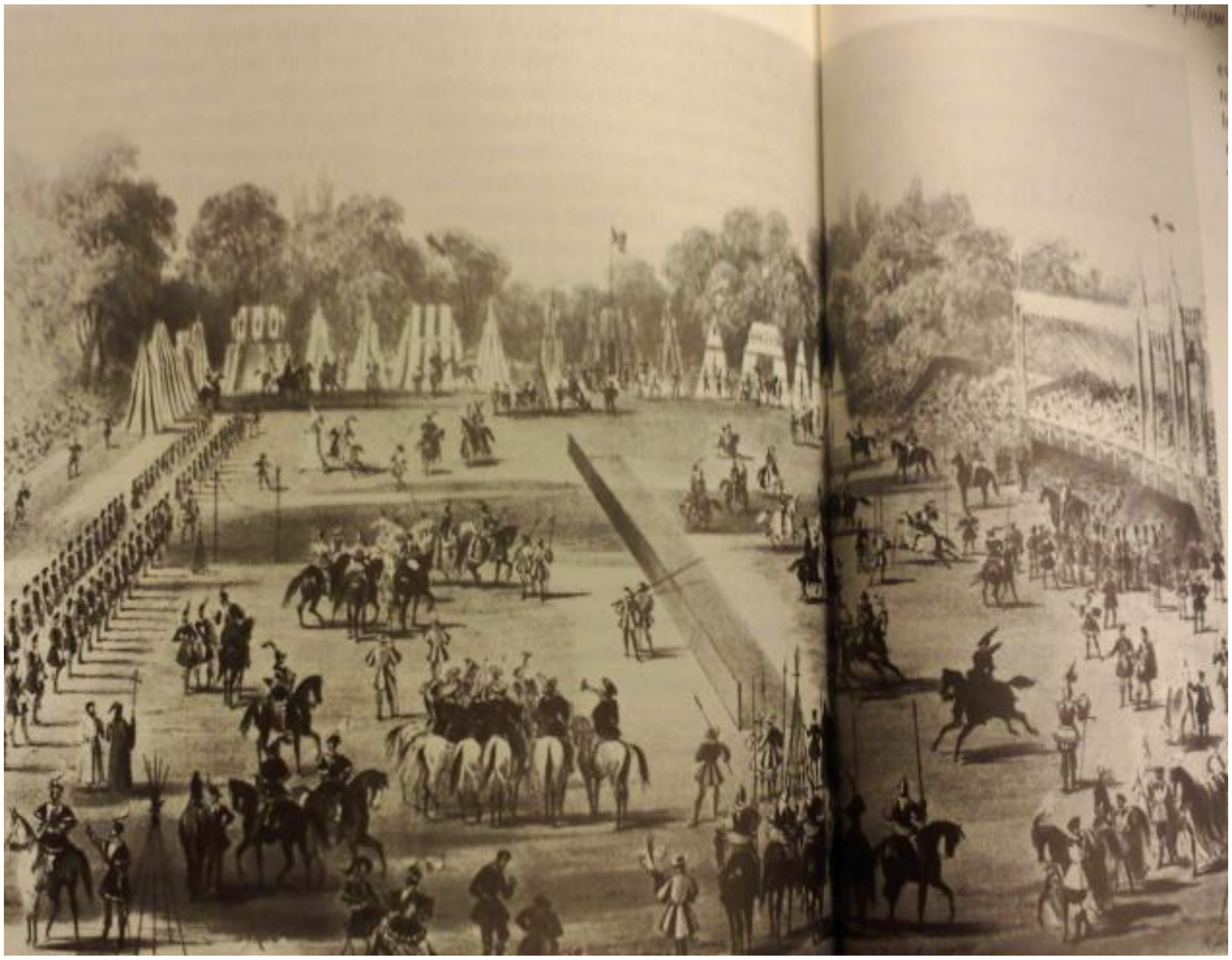
A basic jousting list. This one is not ornate but provides the basic setup needed for a joust.

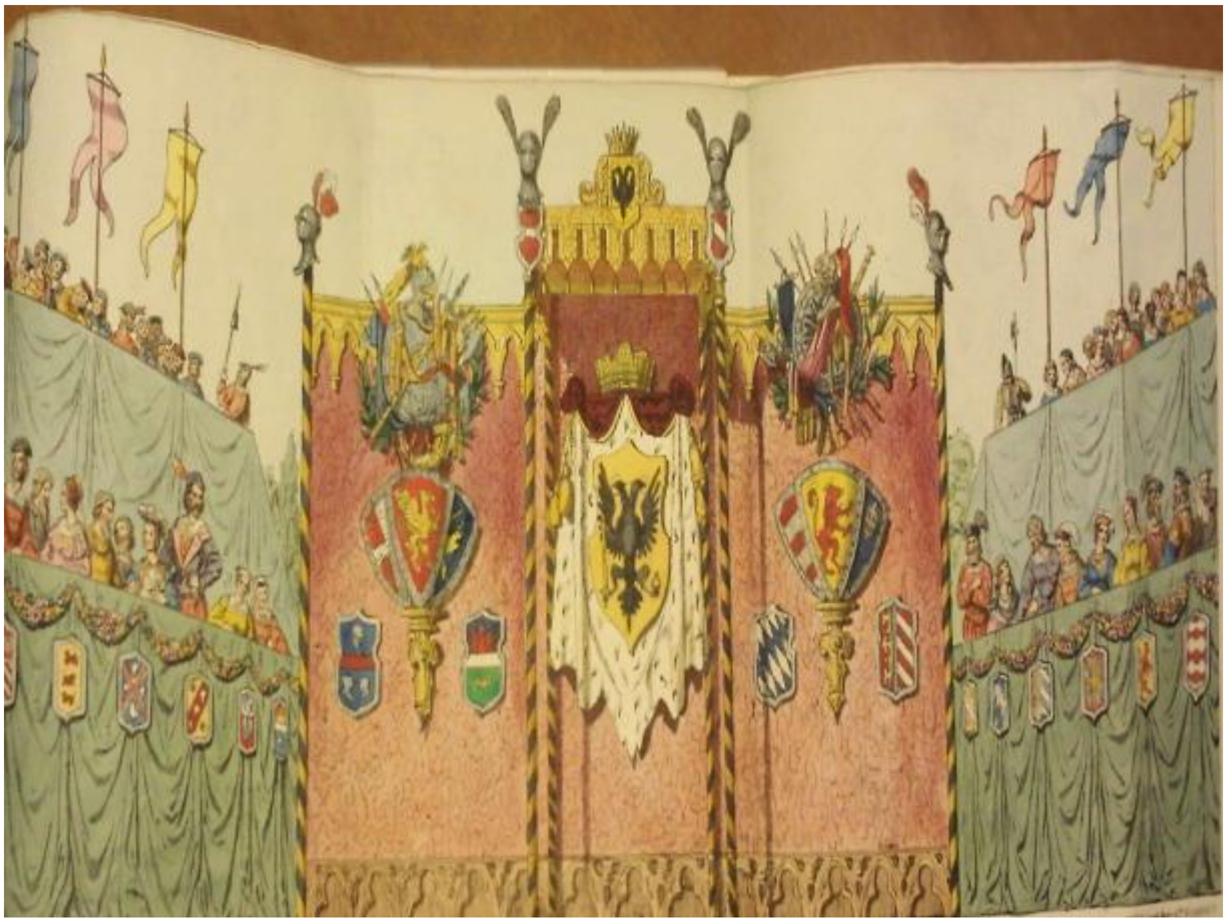
A town decorated for a tournament. Hanging from the windows is the heraldry of the knights that will be participating in the tournament.

Pretournament preparation. The knight's helmets are lined up on display for the judges to view. At this time, the judges were able to expel knights from the tournament if they felt they were not chivalrous.

The awarding of the prizes. The lady of the tournament usually gave out the prizes in the tournament.

Herald issuing a challenge between two people.


A modern example of a tournament field.



Another modern view of the tournament stand with pageantry. Shows basic view of how the stands were set up and what the royalty stands would look like compared to the normal stands.

Appendix II: Interview Question Outline

Interview Question Outline For Higgins Armory Museum Interviews

For this project, which group would you prefer to benefit? Would a project that is centered on making the researchers and curators work easier be useful for you or do you prefer that the project be centered on the public?

Depending on the answer given, use the correct path. Path 1 revolves around questions that would help with the curators and researchers while path 2 is centered around the public.

1. *Researcher and Curator questions*

- 1.1. Which areas of research/work are most active here? Is there any type of software product that you could use to make this work easier?
  - 1.1.1. What type of tool would you be looking for? This project could create an something that assists with data storage, data gathering, or something that makes collaboration easier for you. Would any of these be of interest for the researchers?
  - 1.1.2. Based upon your current tools, do they have any limitations or issues that you feel could be fixed by this project?
- 1.2. What is the average day for a researcher here? By knowing this, it will help us to brainstorm ideas for projects that we feel might benefit the researchers.
  - 1.2.1. Do researchers have any repetitive tasks that they must complete on a daily bases?
  - 1.2.2. Have the researchers complained about any problems they have encountered that our project might be able to solve? If so, can you describe the problem to me or tell me a way to contact one of these researches so I can ask them about the problem?

1.3. How does the museum currently manage recording its visitor data? Would a program that helps makes this information easier to view and understand be useful to the museum? If so, what would the museum need in it to ensure that it met all of your needs?

1.4. Do you have other suggestions as to projects that we could do to assist the museum's researchers?

(Ask 1.4 last. Build upon the answers that are given. Addition questions to obtain extra detail and clarity will assist with determining what the best course for the project is.

2. ***Public Questions***

2.1. What virtual exhibits do you currently have in the museum? Is there any information about how the visitors respond to them? If there is information, would it be possible for us to get it so that we can analyze it for our project to help make it better suited for your visitors?

2.2. Have you seen a virtual exhibit at another institution that you would like to see implemented here? Do you know if there is a website or other resource so we can see this project in case we decide to implement something similar for the Armory?

2.3. Are there any types of visitor feedback that we could use to help determine where visitors are having issues?

2.4. Which demographic of visitor would you prefer to have targeted? Children and families, schools, or adults?

Quest Gallery

2.4.1. What types of interactive are currently available in the quest gallery?

2.4.2. What type of interactive do you feel would be the best addition? A game, simulation, day in the life?

2.4.3. If such a game was implemented and only one target age group could be created, which would you select? (Very young children 2-4, kids 5-7, or 8-10)

2.4.4. Would you like the game to be a stand lone exhibit or be part of a circle of inclusion? (Circle where they play the game and something is sent to an email or stored on the server for them to access at home so they return to the Higgins website)

Great Hall

2.4.5. Are there any areas of the Great hall that have generated a large number of questions? Would an interactive exhibit in this location help the visitors to comprehend the exhibit?

2.4.6. Would the interactive be better focused on people with a small knowledge of arms and armor or would it be better to focus on the experts?

2.4.7. Which section of the Great Hall do you think would benefit from having a virtual exhibit placed in it?

2.4.8. Are there any upcoming exhibits that could benefit with the addition of an interactive? (Might have a special exhibit on the creation of arms. Could add an exhibit that was guiding the user through the process or had images along with the steps to further understanding)

General Closing Questions

2.5. Are there any resources that you feel would be useful for us? Do you feel that additional interviews with other Higgins Armory Staff members would help us get a better idea of what would be the best route for this project to take? (Resources including books, web pages, etc.)

2.6. Do you have anything else might be useful for us?

Finish by thanking them for their time and information. Also stress during the interview that you want to do something that will benefit the museum. This is shows that the group wants to do something that will help them and leaves a good impression. Any important points or items they bring up, talk about them to show even further interest so they know that the group truly wants to make something great.

Appendix III: Interview with Nikki

- Museum has three major types of visitors: children and families, schools, and adults. Within each group are the experts with prior knowledge of arms and armor and the new viewers.
- Reasons for coming
    - Experts come to look at the items in the museum and learn more about the subject.
    - Preschoolers/young kids are another major group. They are heavy into knights, dragons, and princesses which makes coming to the museum interesting for them. Age of the children ages 2-7.
- Two major sections of the museum. The quest gallery that focuses on hands on activates. The great hall is more tailored for the expert.
- The museum labels for the artifacts are made for an expert to read. Provide little information to those that do not have a large knowledge of arms and armor.
- For an exhibit Nikki is seeking something that will give them the "most bang for our buck".
- If its computer based, it might be also possible to put on internet.
- Ideally, want the project to help set of a circle of involvement. Either get people into the museum or having something that will bring them back. Example: Make a program that creates postcards. The emailed postcard has a link back to the museum site.
- The target age group needs to be discussed internally with the staff. Would be useful to have the team there to assist with the discussion after more information has been gathered.
- Currently they do not have a very large number of teen visitors. However, it would be best to work with an audience they already have as this would not require them to do post project work in drawing in a completely new audience.
- A project that targets children and families that also has a connection with the school curriculum would be good.
- Possible projects they have running to link to: interrupt stuff in the great all. They are working on rethinking the quest galley. Other then these there are no real projects to enhance currently.
- The museum has been in the process of collecting visitor questions and concerns. There is also a small survey of what the customers thought.
- If possible the interactive could be used in the museum and as an internet application.
- The previous database projects performed by IQP teams were very useful to the museum since it put all of their collection online.
- *Her ideal interactive would be some sort of learning game*
- Suggested that instead of interviewing steve, which is no longer possible as he has left the museum, try getting interview with Linda Woodland who is the project manager of exhibits.
- Doesn't know about the number of hits on the website. However, is more interested in how long people that visit the website stay.
- Currently there are 2 interactive display. It might be useful to observe these two interactive and get some data about the audience. Best times for this are Sunday 1pm to 3pm and Saturday 11am to 3pm.
- Areas to highlight would be either the great hall or the quest gallery.
- Does not know of any previous projects that need to be expanded
- Other resources

    - www.eduweb.com has soft copies of the papers she gave me as hard copy
    - American Association of Museums aam-us.org web address. Has a committee that targets media and technology. Also they have a competition for projects. Will provide useful link as to what the winners looked like
- Network administration is taken care of through subcontractor. To get information about network will need to work with the museum to contact him.

Background information on Nikki

Has a career with museums, with most of it as Indianapolis Children's museum. Was able to bring staff up to 250 from 10 and increased attendance from 200k to 1.5 million. Her specialty is with the public relations section. Has experience with setting up teams and how to ensure that teams work well.

Appendix IV: Interview with Operand

Operand: explore interactivity interview on September 17, 2008.

Present: 3 Higgins armory museum members. And the 4 WPI group members.

Operand: How they create their own interactive projects. Hopefully gain insight in the project. Will help us to enable our own interact experiences.

New York city based interactive agency that develops interactive experiences for many different clients. Primary market is museums, retail establishments. Large list of different clients. 14 people in company.

Strategy phase: initial phase of design. Gather information, determine what is going to be done.

Design and then production. Have one manager to ensure things get done.

Engagement of the audience is important since if they don't have interaction then the exhibit was a fail.

Closer to 8 is the better age for interactive exhibits. Might be able to get a bit lower but if to young it might become impossible to keep attention

First level of interaction:

Observation: look at it only but can't touch. Often occur in a museum. Not really much of an interactive just really a short movie clip.

Case studies mentions: A sports museum. In each of the galleries there is an interactive that they can use.

Make the design to help show the artifacts off. They attempted to show the story of the artifacts. Appears that they are mentions mostly observing movies and such. These are not the interactive types that we are after. Makes use of powerful wording to catch people's eyes.

Linear 1: at this point, it is more of an interactive to the since that we need. Similar to a website. In museum, observation is the most popular "interactive" while linear is the next most popular type.

Database interactive: just filled with data. Mostly just a searchable database with information. Don't allow the interactive to become boring.

Form factor: the medium in which the interactive is delivered (touch screen, screen and rollerball, buttons

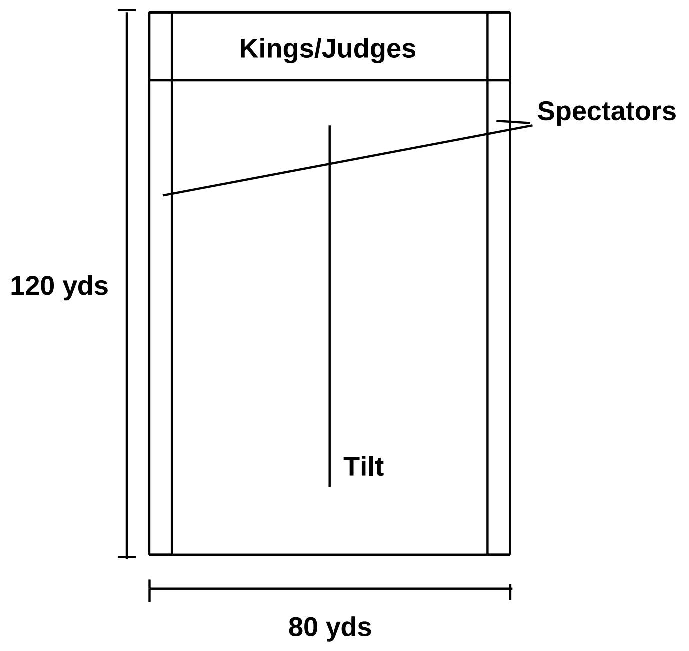
Lists will not get audience attention. Do not use a list cause that will very very quickly cause that people to become board. Do not use these two types of exhibits. It will accomplish nothing except boring the audience to death.

Immersive: attempting to make them an active character into the environment.

Appendix V: Joust Mechanics

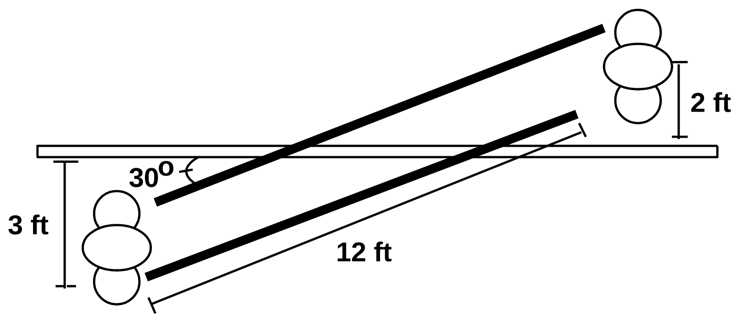
About the Field

- The tilt is approximately 6 feet high
- These are dimensions taken from the lists for Lord Scale's combat in 1467.
- Kings/Judges arranged on the north side like shown
- Tents for servicing of jousters may be located on south
- Noted that the tree for challenges in the north as well (perhaps beyond the kings/judges seats)



Positioning during the Moment of Impact

- Generally pass left arm to left arm, holding lances on the right
- 30 degree angle of impact ensures best chance of shattering



Possible Outcomes of a Run (Player)

- Three primary targets: helm, between waist and helm, and shield
- Possible Scenarios
    - 1st Outcome: Completely miss
    - 2nd Outcome: Glance one of the three primary targets (possible unhorse)
    - 3rd Outcome: Shatter lance on one of the three primary targets (possible unhorse)
    - 4th Outcome: Lance hit the barrier
    - 5th Outcome: Lose control of horse – go off course
- Outcomes are determined by a hidden point system shown later
- Hit across the knight is not taken into account; lance mobility will be limited so this scenario will not occur

Possible Outcomes of a Run (Computer)

- Computer opponent will not be able to unhorse the player
- Hits from the computer will be random based on statistics of actual jousts of the past (possibly toned down to make it easier for the player); Strikes are beyond player control for the most part (i.e. players cannot do anything to prevent the strikes; they can only aim to strike the opponent in a better spot)
  - For example, from the Westminster Tournament of Nov 1501, out of 57 courses, 1 lance broken, 6 head attaints, 10 spears broken on the body.
- Possible Scenarios
  - 1st Outcome: Completely miss
  - 2nd Outcome: Glance one of the three primary targets (possible unhorse)
  - 3rd Outcome: Shatter lance on one of the three primary targets (possible unhorse)
- Computer players cannot fault like the player; this is to ensure the player gets the opportunity for a full run each round
- After actual testing, computer hit statistics will be modeled from the data gathered from players

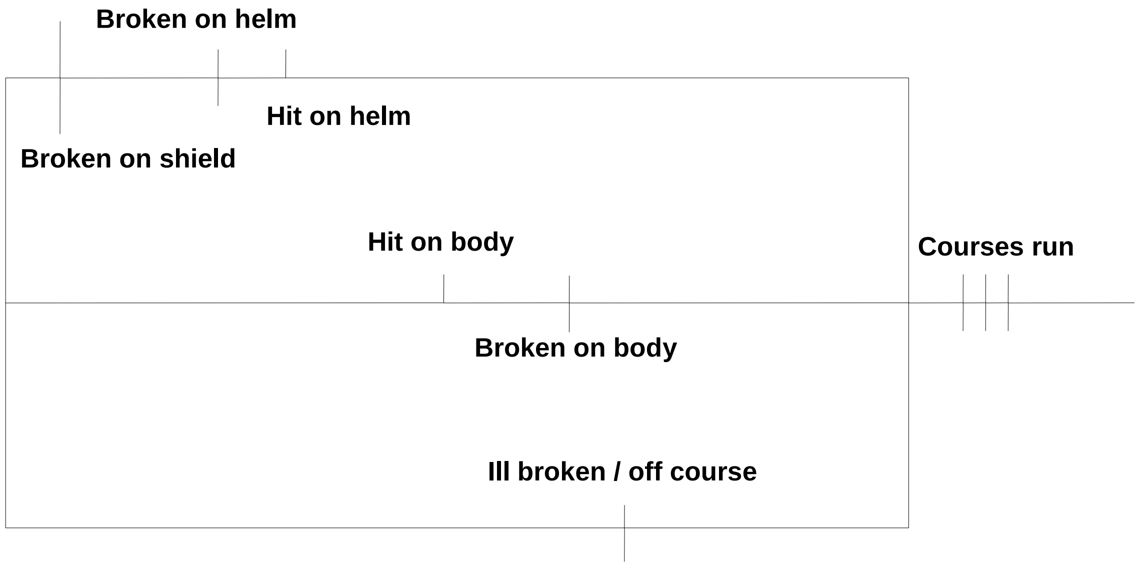
Point System

- Point system will be comprised of explicit and implicit bonuses/penalties
- Explicit Scoring – Players will be able to actually see these points awarded for the actions they take

Action	**Points**
Lance hit between waist and bottom of helm	+0.5
Lance shattered between waist and bottom of helm	+1.0
Lance hit on helm	+1.0
Lance shattered on helm	+2.0
Lance hit on shield	+1.5
Lance shattered on shield	+3.0
Losing control of horse – to the point of going off course	-2.0
Lance hitting the barrier	-1.0
Unhorsing the opponent	+5.0

  - Points will also be awarded for achievements. Such achievements include consecutive strikes to the same area, shattering the lance every round, and unhorsing the opponent every round (very rare!). Exact values and award titles still to be determined.
- Implicit Scoring – These values are added to the final score without the player actively being alerted
  - Showiness – Heralds just made will begin with a default rating of 5 out of 10. On a regular basis, other players can rate heralds. The higher rating the herald, the more hidden points awarded. Points cannot be subtracted. Maximum point gain in this area will be 5. Min rating is 5.

    - Horse Control – Every half a second or so, depending on the current location of the target in the horse engine, points will be awarded (more points for better horse control). Lowest award is 0, highest is 5.
    - Difficulty Levels – There will be varying levels of difficulty to be determined after an initial game engine has been developed. Higher difficulty will result in higher point awards. Again, max will be 5 points.
- Scores will be added/deducted from a base of 15. This is to ensure no scores below 0, and a more impressive looking high score (number inflation)
- Scoring will be done on the scorecard proposed previously:

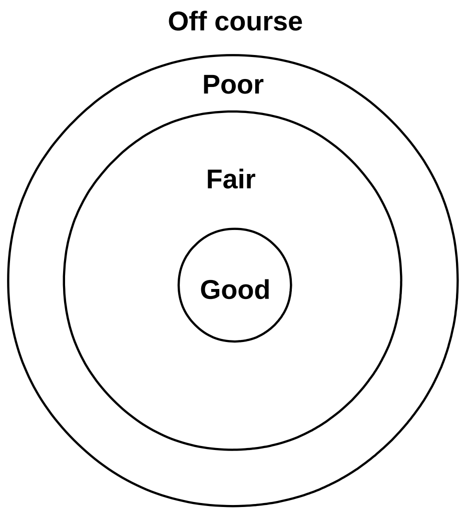


Time of an Actual Run

- An actual run will take 7 seconds
- 0 – 2s, the horse begins to move and the player can start to lower the lance
- 2s – 6s, the player will gain control of the horse, and can continue lowering the lance (if already lowered, they must position the lance. The sooner the lance is lowered, the sooner the counter for the intensity of random jitter begins. This means lowering the lance later will result in better accuracy for the final hit)
- 6s – 7s, player given 1 second for any last minute adjustments for the final contact (perhaps we should remove horse control from this point on, and only focus on the lance positioning?)

Horse Control

- Involve the use of the board
- Try to keep a target (circle or some icon) inside a range
- The circles of ranges will be randomly moving inside an area.
- If we want to get elaborate with the GUI, can use a more visually appealing shape
- When in the off course area, the player will be permitted about 0.5 – 1 seconds to correct before running off course
- Screen moves accordingly (if target to right, screen environ shifts to the left)

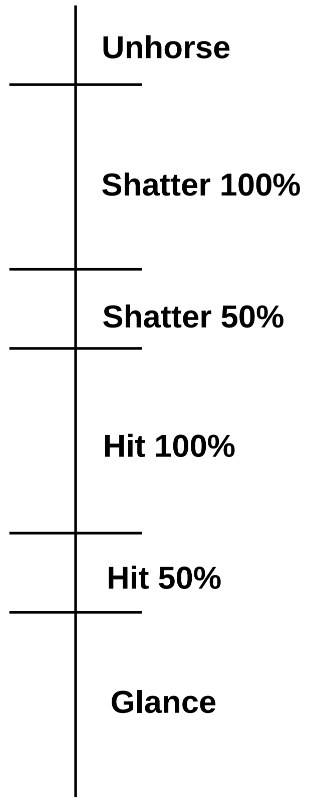


Lance Control

- As shown previously, use of a lance stub embedded with a Wii-mote
- Try to aim for the center of helm/body/shield
- Can be fully lowered any time in the 6 seconds of the run prior to the last second
- The sooner it is lowered, the harder to control in the last second
- Possible to implement hidden points in this area

Shattering Determination

- This will be based on a hidden point system (not to be confused with end game result scores)
- This point system will be applied when a target is actually hit in the collision
- Horse will basically be in 3 possible lanes, which will affect the outcome
    - Good lane = factor of 1.5
    - Fair lane = factor of 1
    - Poor lane = factor of 0
- The factors obtained from the horse will be multiplied by the points awarded based on location of hit
    - For example, the shield will consist of 3 regions
        - A good region = 100 pts
        - Fair region = 40 pts
        - Poor region = 10 pts
    - Therefore, a good lane and a good region will result in 100pts * 1.5 = 150 pts
- Finally, this product will be added to points awarded based on how well the lance was lowered (ideal is for lance to be lowered near the end)
- Exact numbers have yet to be determined
- This is to ensure that proper jousting procedures are followed, and a person cannot dismount with just a lucky last minute movement of the lance/horse



Appendix VI: Asset List

Hardware

Bluetooth Adapter
Computer
Lance Stub
Monitor
Protective Cage
Speakers
Wii Balance Board
Wii Remote
Wireless Wii Sensor Bar

Software

Adobe Flash Player
Java Runtime Environment (6.0 or later)
Web Browser
Windows OS (XP/VISTA)

Audio

Cheering crowds
Horse clopping
Horse neighing
Intro Music
Lance shattering
Lance striking armor
Trumpets
Voice-overs for all text

Visual

Backdrop image
Crowd image
Tilt image
Enemy knight image
Enemy horse image
Enemy lance image
Enemy shield image
Historical context image
Horse head image
Lance image
Title screen slideshow
Scoreboard image
Scorecard image
Crest charge images
Crest field images
Sky image

Appendix VII: Storyboard

Underlined text is what we have voiceovers for.

Start Scene

Have they played before?

<u>Honorable knight, have you participated in our tournaments before? If so, we can use the information that we already have stored about you.</u>

Use the lance to move the mouse and scroll over the correct answer.

Then use the back trigger button on the reins to select your answer

Have played before, input name

<u>Noble knight, if you will tell me your name, I will go and retrieve your information</u>

Enter your name. After you are finished, use the lance to scroll over the green button and click it using the back trigger on the reins

Actually, I would like to register as a new knight

Score to win message

New player

<u>"You, a noble knight of the realm, have been invited to participate in an upcoming tournament. Confident in your skills with your horse and lance, you accept the invitation. In order to win the tournament, you must score at least 14 points. You will score 3 points for a hit on your opponent's body, 4 points for hitting his head, and 5 points for hitting his shield. If you control your horse well and lower your lance at the right time, you will shatter your lance or possibly even unhorse your opponent. When the time arrives, you set out with your convoy to the tournament grounds to test your skill against other knights.</u>

Old player, hasn't beat first high score

Knight (playerName), in order to win the tournament you must beat the current high score of 14 points over four rounds of jousting. Good luck (playerName), joust with honor and chivalry."

Old player, has own high score

Knight (playerName) in order to win this tournament you must beat or equal your previous score of (the player's highscore) over four rounds of jousting. Good luck (playerName) joust with honor and chivalry."

Game opening

The tournament of (Name of tournament)[1]. When you first arrive, you see that the town has been decorated for the occasion. Hanging from the windows is the heraldry of the knights that will participating in the upcoming event. Setting off for the judges gallery, you see a herald holding the knights' banners that will decorate the jousting grounds.

New player

In order for the spectators to recognize you on the field, you need to register your heraldry with the judges. After registering, you leave your helmet in the gallery for the other knights and ladies to see.

Old Player

As a veteran of this tournament, the judges and ladies already know you. As you pass through the judge's galley, you notice them nodding approvingly in your direction after they see your heraldry.

The first step is for the judges to identify you by your heraldic coat of arms.

[1] See appendix for list of tournament names.

Transition to the Field

Now that you registered for the tournament, you can proceed to the jousting field to prove your skill as a knight. When you are ready, point your lance upwards, and shake your reins to urge your horse forward.

Results

Win

Congratulations! You have won the tournament! (onscreen)

<u>Today you have demonstrated honor and chivalry on the field of battle. In recognition of your outstanding skill, from here forward this tournament shall be named in your honor!</u> (voiceover)

Lose

<u>After a long day of jousting, you were unable to win this tournament. Yet your practice today has given you valuable experience as a knight. Knowing that there are other tournaments for you to attend, you head back to your homeland to continue practicing for your next try.</u>

Thank you for playing!

Resources

Tournament names

Blue Anchor

Silver Axes

Black Bear

Golden Cross

Golden Crown

White Fleur-de-lis

White Rose

Golden Crescent

Black Horse

Golden Keys

Red Lion

Silver Shell

Green Tree

Green Crozier

Red Wheel

Humble Knight

The Virtual Joust

Patrick Newell

One of the more unique events during medieval tournaments was the joust. In this event, mounted knights would charge at their enemies using various weapons, with the intention of dismounting them. In an effort to replicate this event, a virtual jousting game will be produced. Four main areas of research will support this project: historical context of the joust, mechanics of the joust, design concepts, and technical concepts. Background research on the joust will be required to set up the historical context and game mechanics, and design and technical research will be required in order to construct the actual game.

TABLE OF CONTENTS

Installation and Setup

This section details the resources and procedures required to install and operate the full and demonstration versions of the Virtual Jousting Simulator. The full version of the Virtual Jousting Simulator includes Nintendo Wii Remotes for handling user input for the lance and horse. The demonstration version of the Virtual Jousting Simulator uses a standard keyboard and mouse to handle user input.

Prerequisites

The Virtual Jousting Simulator requires the following hardware and software installed in order to complete the subsequent installation and setup steps in this section:

Computer System Requirements

The Virtual Jousting Simulator is designed to run on a personal computer or a laptop computer. Regardless of the type, the computer should have the following minimum specifications:

Processor:	1.0 GHz or higher
RAM:	1 GB or higher
Video Card:	256 MB of video RAM or higher
Available Disk Space:	1 GB
Operating System:	Windows Vista or Windows XP
Optical Drive:	Must be able to read CD-ROM's.

Additional Hardware Requirements

The following hardware is required in order to operate the Virtual Jousting Simulator with respect to user input and graphical displays:

- **LCD or CRT Monitor** that supports a resolution of 1024x768 pixels and is compatible with the computer platform described in section 1.1.1.
- **LCD Projector (Full Version Only)** that supports a resolution of 1024x768 pixels and is compatible with the computer platform described in section 1.1.1.
- **Projector Screen (Full Version Only)** that can fit the display of the LCD projector described above.
- **Mouse** with at least two buttons that is compatible with the computer platform described in section 0.
- **Standard keyboard** (either external or onboard to laptop) that is compatible with the computer platform described in section 0.
- **2 Nintendo Wii Remotes (Full Version Only)**
- **Bluetooth Receiver (Full Version Only)**, either internal or external that is listed on the following website that contains known compatible Bluetooth devices: http://www.wiili.org/index.php/Compatible_Bluetooth_Devices. Using an external Bluetooth adapter will usually require a USB port to be available on the computer platform described in section 0.
- **Nintendo Wii Remote Sensor Bar (Full Version Only)**

Computer Software Requirements

The following software should be installed on the computer platform detailed in section 0 in order for the Virtual Jousting Simulator to run. If any of this software is not installed, it can be obtained from the websites provided for each item.

- **Latest version of Java JRE:** http://www.java.com/en/download/index.jsp
- **Latest version of Flash Player:** http://get.adobe.com/flashplayer/
- **The Virtual Jousting Simulator software package**
- **Bluetooth Software and Drivers for the Bluetooth Receiver to be used (Full Version Only)**

Nintendo Wii Remote Setup (Full Version Only)

This section details the procedures for setting up the Nintendo Wii Remotes for the full version of the Virtual Jousting Simulator. If setting up for the demonstration version of the Virtual Jousting Simulator, advance to the next section. Ensure that all prerequisites have been met before starting this procedure.

Bluetooth Installation

If using an external USB Bluetooth adapter (Figure 1.2.1), insert it into an available USB port on the computer (Figure 1.2.2) that will run the Virtual Jousting Simulator. Otherwise, omit this step.

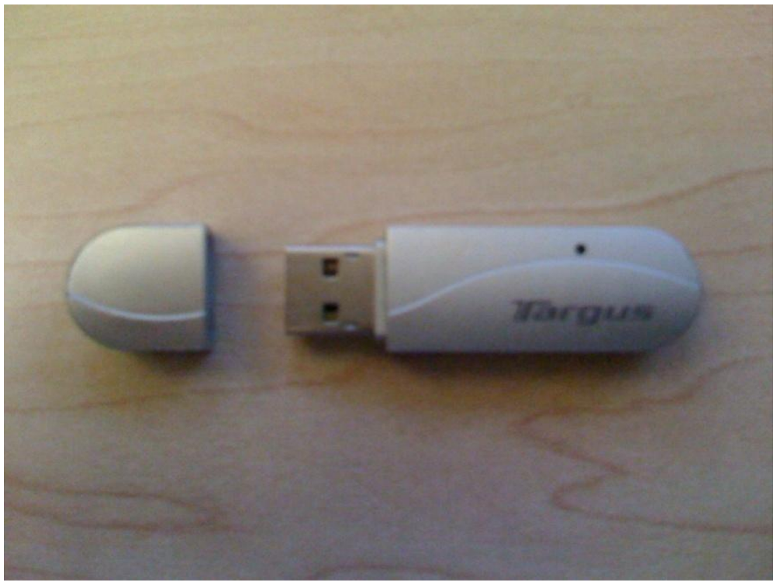


Figure 1.2.1 – A Targus USB Bluetooth Adapter.

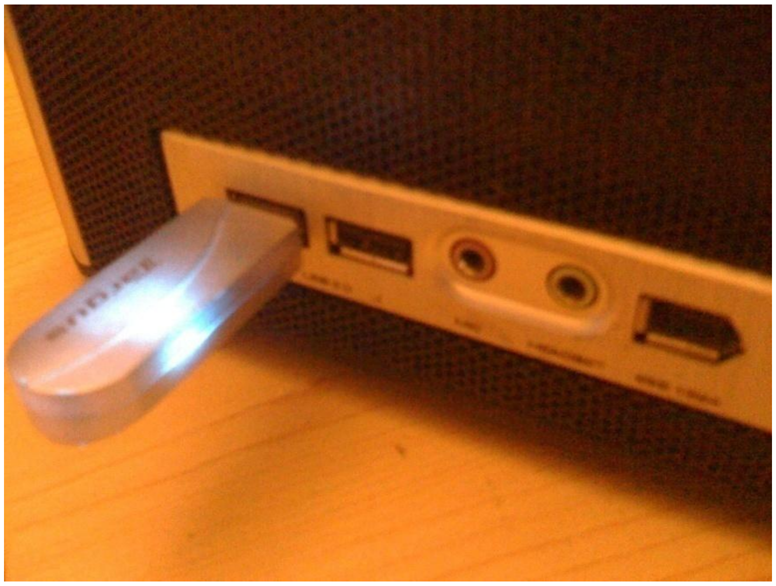


Figure 1.2.2 – Targus USB Bluetooth Adapter inserted into a USB port.

If Bluetooth drivers have not been installed for the Bluetooth device, follow the Windows driver software installation utilities instructions to install the required software to use the Bluetooth device. Otherwise, omit this step.

If the Bluetooth device is working properly, a small Bluetooth icon should be located on the Windows taskbar (Figure 1.2.3). Single left-click on the Bluetooth icon and single left-click on "Show Bluetooth Devices" option on the Bluetooth pop-up menu (Figure 1.2.4).

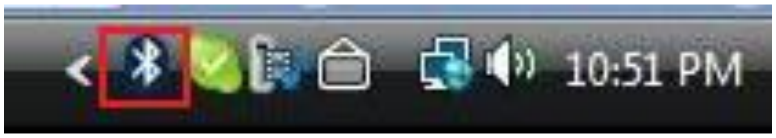


Figure 1.2.3 – Bluetooth icon (outlined in red) on Windows Vista taskbar.

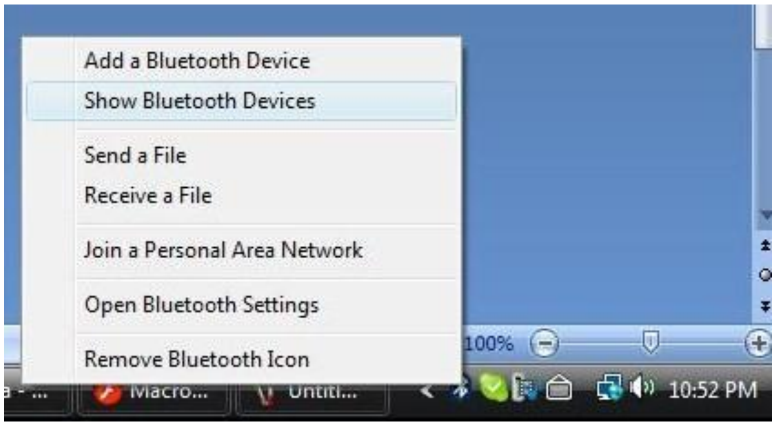


Figure 1.2.4 – "Show Bluetooth Devices" option on Bluetooth pop-up menu.

Verify that the "Bluetooth Devices" window appears on the desktop as in Figure 1.2.5. If any devices containing the word, "Nintendo", in its label is present in the "Devices" tab of the "Bluetooth Devices" window, advance to section 1.2.3 and perform the necessary steps to reset the Nintendo Wii Remote installation process before continuing to step 0.

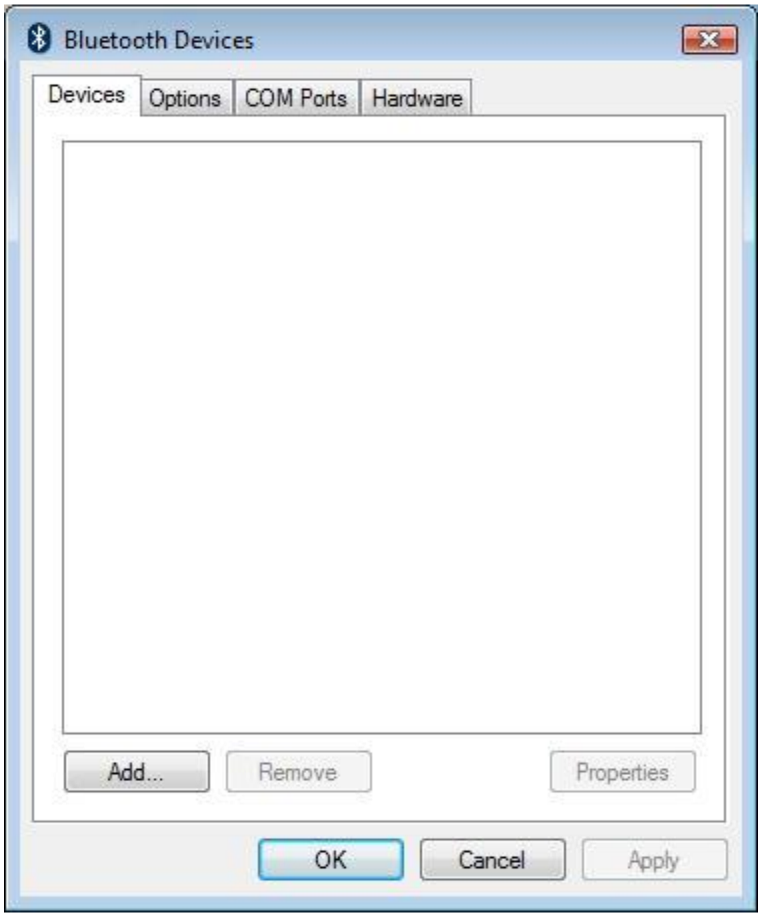
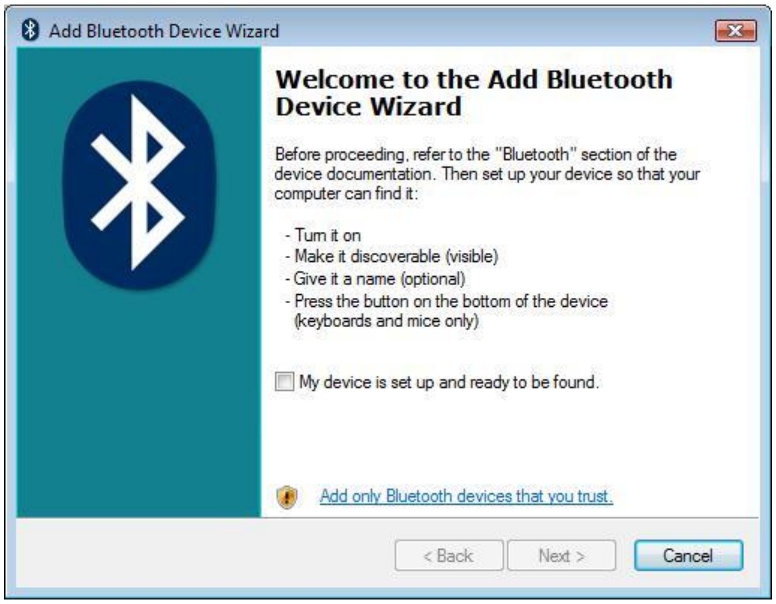


Figure 1.2.5 – “Bluetooth Devices” window.

Click the “Add…” button in the “Bluetooth Devices” window (Figure 1.2.5).

Verify that the “Add Bluetooth Device Wizard” (Figure 1.2.6) appears on the desktop.


Add Bluetooth Device Wizard

Welcome to the Add Bluetooth Device Wizard

Before proceeding, refer to the "Bluetooth" section of the device documentation. Then set up your device so that your computer can find it:

- Turn it on
- Make it discoverable (visible)
- Give it a name (optional)
- Press the button on the bottom of the device (keyboards and mice only)

☐ My device is set up and ready to be found.

Add only Bluetooth devices that you trust.

Figure 1.2.6 – "Add Bluetooth Device Wizard" window.

In the "Add Bluetooth Device Wizard" (Figure 1.2.6), single left-click the check-box labeled, "My device is set up and ready to be found".

IMPORTANT:

Steps 0 through 0 should be performed as quickly as possible in order to get the Nintendo Wii Remote to connect to the computer running the Virtual Jousting Simulator. If at any point during these steps the blue LED lights on the Nintendo Wii Remote stop flashing, stop the installation process and start the Nintendo Wii Remote reset procedures in section 0.

Locate a Nintendo Wii Remote (Figure 1.2.7) and locate the "1" and "2" buttons on the Nintendo Wii Remote.

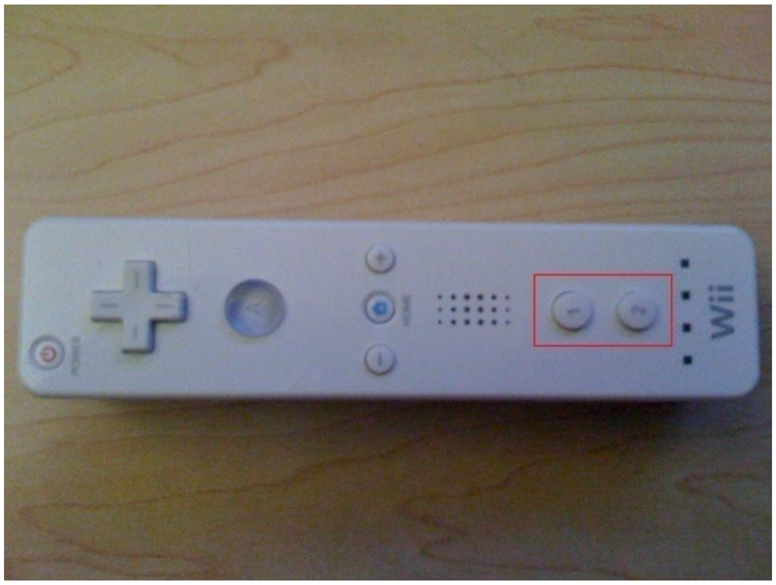


Figure 1.2.7 – A Nintendo Wii Remote with the “1” and “2” button outlined in red.

Simultaneously depress and release the “1” and “2” buttons on the Nintendo Wii Remote as shown in Figure 1.2.8.

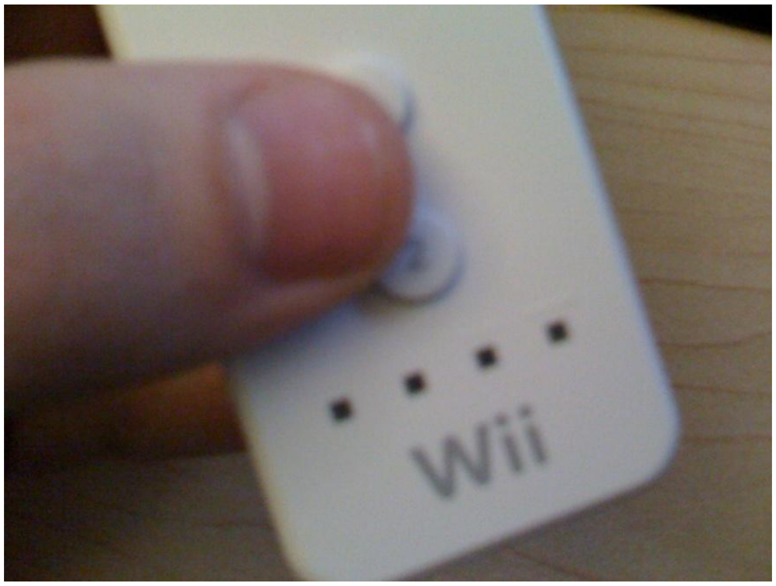


Figure 1.2.8 – Depressing and releasing the “1” and “2” buttons.

Verify that the four blue LED lights on the Nintendo Wii Remote flash on and off continuously (Figure 1.2.9). (*Note: If less than four LED lights flash continuously on the Nintendo Wii Remote, this is an indicator that it may not have sufficient power from its batteries and may not be able to connect to the computer. It is recommended that the batteries in the Nintendo Wii Remote either be replaced with new ones or be recharged if this occurs*).

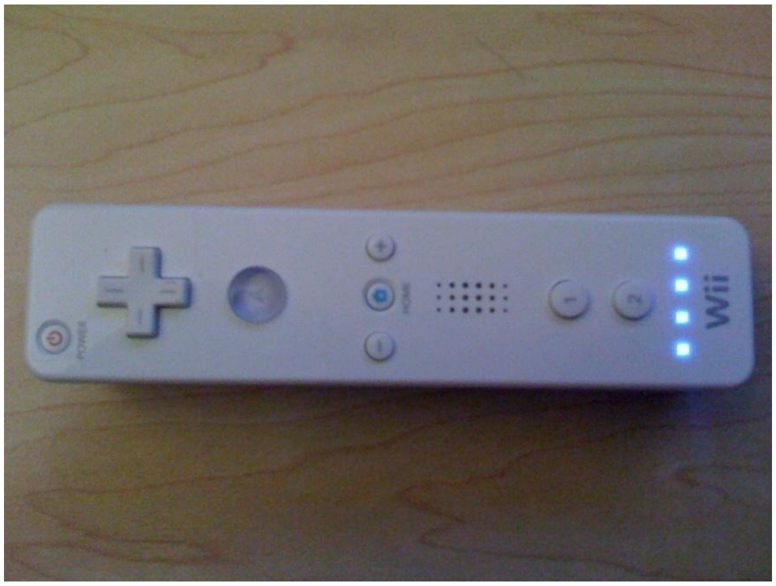
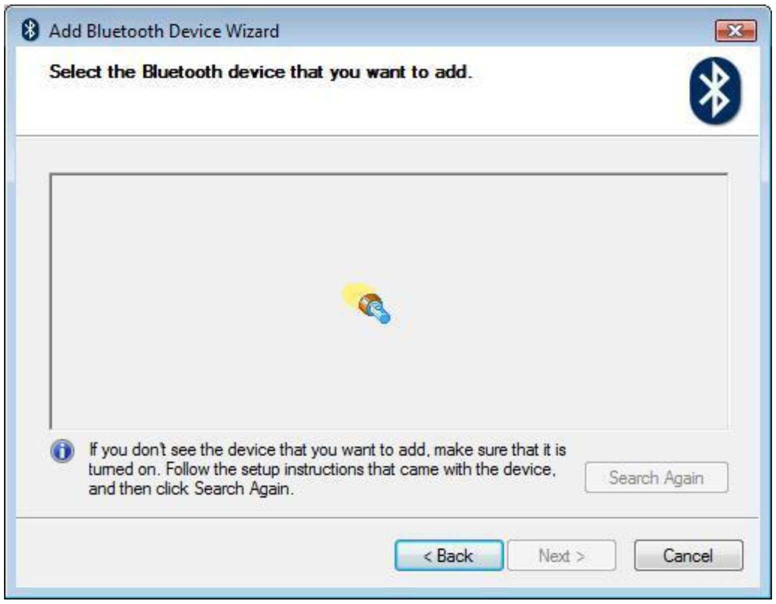


Figure 1.2.9 – Nintendo Wii Remote with four of its LED lights on.

In the "Add Bluetooth Device Wizard" (Figure 1.2.6), single left-click the "Next" button.

Verify that the "Add Bluetooth Device Wizard" window changes and begins searching for Bluetooth devices as shown in Figure 1.2.10.

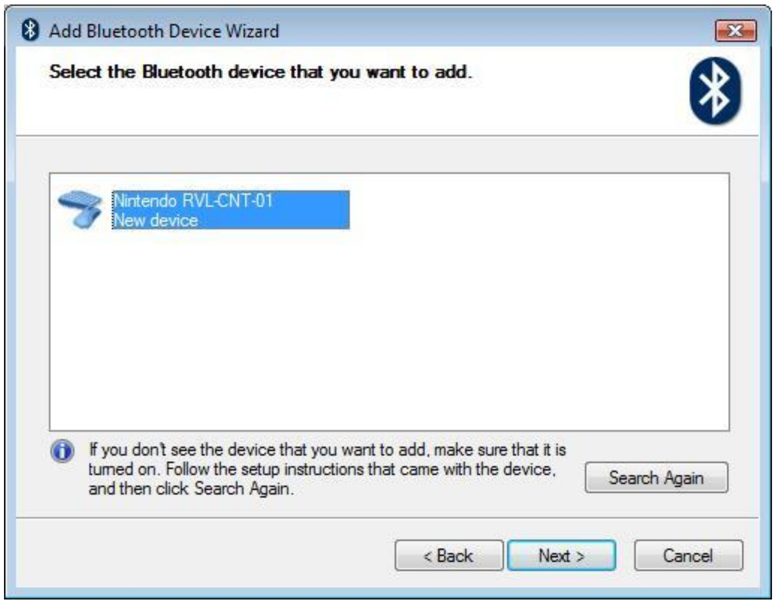

Add Bluetooth Device Wizard

Select the Bluetooth device that you want to add.

If you don't see the device that you want to add, make sure that it is turned on. Follow the setup instructions that came with the device, and then click Search Again.

Figure 1.2.10 – "Add Bluetooth Device Wizard" searching for Bluetooth devices.

Verify that when the "Add Bluetooth Device Wizard" finishes searching, a device labeled "Nintendo RVL-CNT-01" appears as shown in Figure 1.2.11


Add Bluetooth Device Wizard

Select the Bluetooth device that you want to add.

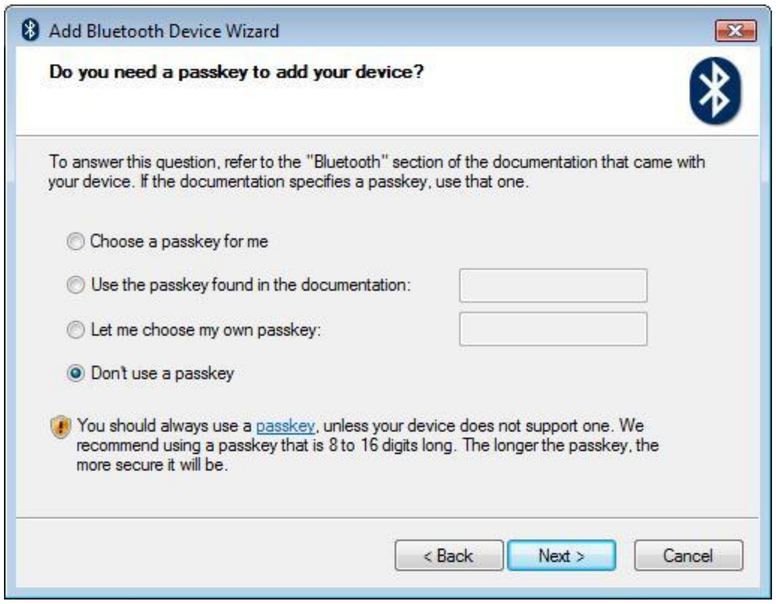
Nintendo RVL-CNT-01
New device

If you don't see the device that you want to add, make sure that it is turned on. Follow the setup instructions that came with the device, and then click Search Again.

Figure 1.2.11 – "Add Bluetooth Device Wizard" with Nintendo RVL-CNT-01 device selected.

Select the device labeled “Nintendo RVL-CNT-01” by single left-clicking its associated icon and single left-click the “Next” button.

Verify that the “Do you need a passkey to add your device?” screen appears in the “Add Bluetooth Device Wizard” window (Figure 1.2.12).



Do you need a passkey to add your device?

To answer this question, refer to the "Bluetooth" section of the documentation that came with your device. If the documentation specifies a passkey, use that one.

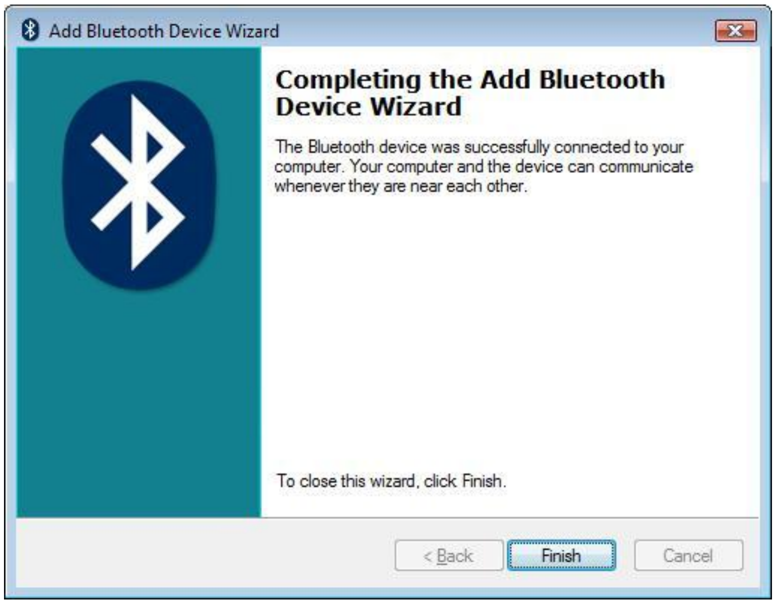
- ○ Choose a passkey for me
- ○ Use the passkey found in the documentation:
- ○ Let me choose my own passkey:
- ● Don't use a passkey

You should always use a passkey, unless your device does not support one. We recommend using a passkey that is 8 to 16 digits long. The longer the passkey, the more secure it will be.

Figure 1.2.12 – The “Do you need a passkey to add your device?” screen.

In the “Do you need a passkey to add your device?” in the “Add Bluetooth Device Wizard” window, single left-click the “Don’t use a passkey” radio button to select it and single left-click the “Next” button.

Verify that the “Add Bluetooth Device Wizard” completes as seen in Figure 1.2.13.



Completing the Add Bluetooth Device Wizard

The Bluetooth device was successfully connected to your computer. Your computer and the device can communicate whenever they are near each other.

To close this wizard, click Finish.

Figure 1.2.13 – "Add Bluetooth Device Wizard" has completed.

In the "Add Bluetooth Device Wizard", single left-click the "Finish" button.

Verify that the four blue LED lights are still flashing on the Nintendo Wii Remote. If the four blue LED lights are still flashing, repeat steps 0 through 0 to add the second Nintendo Wii Remote. If the second Nintendo Wii Remote has already been connected, omit the next steps and continue to the next section. If the four blue LED lights are not flashing, continue to the next step.

Verify that the newly added "Nintendo RVL-CNT-01" device is present in the "Bluetooth Devices" window (Figure 1.2.14).

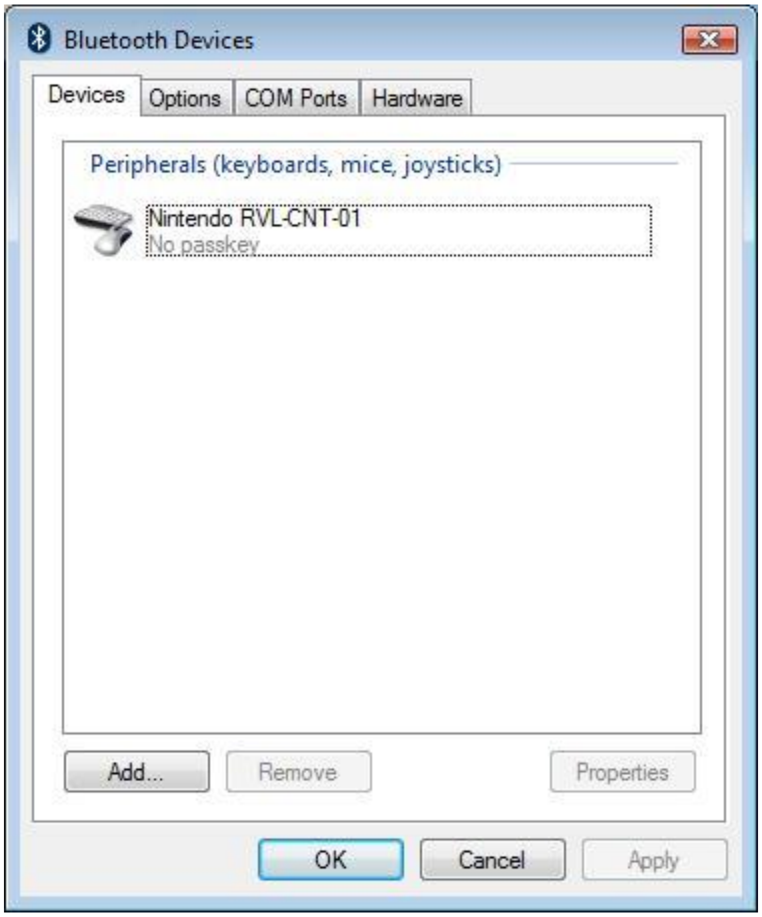
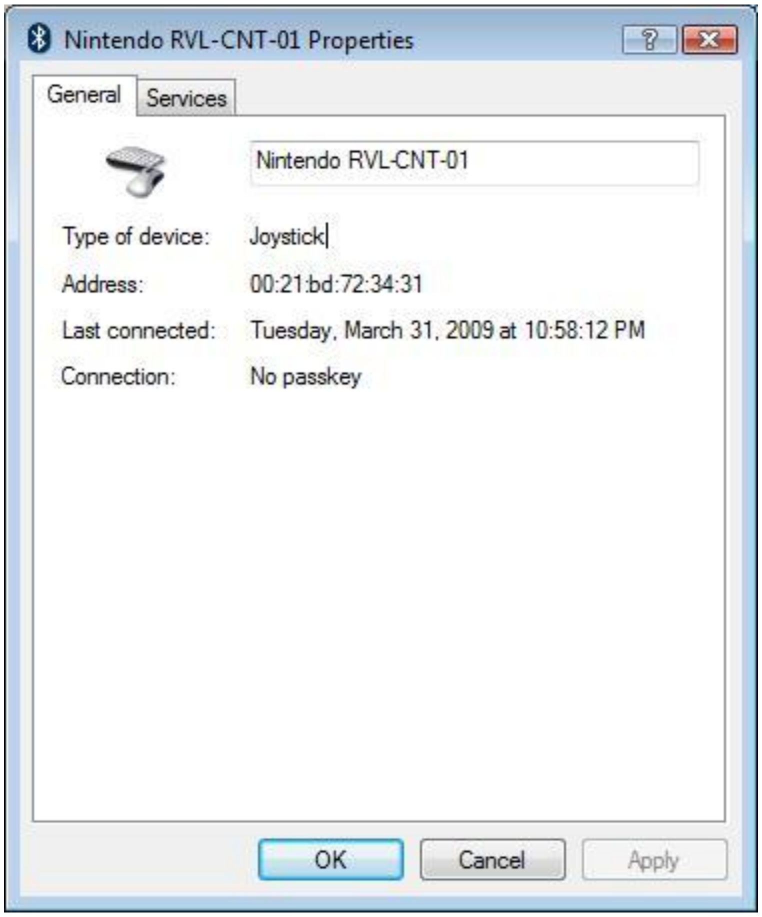


Figure 1.2.14 – Bluetooth Devices window with a single Nintendo RVL-CNT-01 device.

Single left-click on the "Nintendo RVL-CNT-01" device icon to select it and single left-click the properties button.

Verify that the "Nintendo RVL-CNT-01 Properties" window (Figure 1.2.15) appears on the desktop.


Nintendo RVL-CNT-01 Properties

General | Services

Nintendo RVL-CNT-01

Type of device: Joystick

Address: 00:21:bd:72:34:31

Last connected: Tuesday, March 31, 2009 at 10:58:12 PM

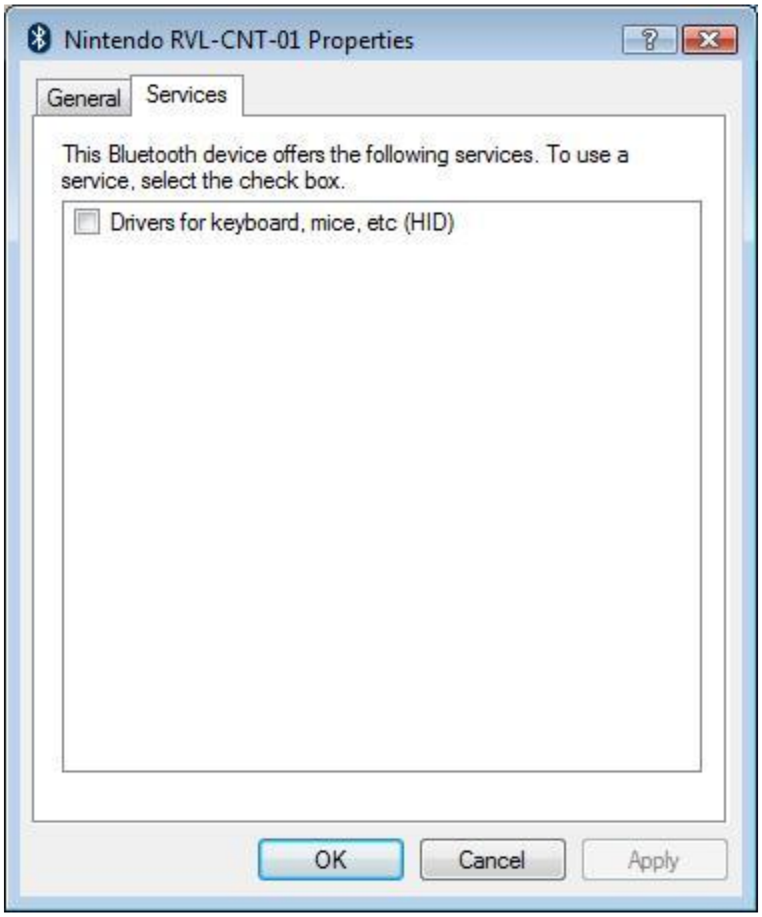
Connection: No passkey

Figure 1.2.16 – The “Nintendo RVL-CNT-01 Properties” window.

Repeat steps 0 and 0.

Single left-click the “Services” tab in the “Nintendo RVL-CNT-01 Properties” window.

Verify that the “Drivers for keyboard, mice, etc (HID)” service appears in the “Services” tab in the “Nintendo RVL-CNT-01 Properties” window (Figure 1.2.17).



This Bluetooth device offers the following services. To use a service, select the check box.

☐ Drivers for keyboard, mice, etc (HID)

Figure 1.2.17 – The "Drivers for keyboard, mice, etc (HID)" service.

Single left-click the "Drivers for keyboard, mice, etc (HID)" checkbox to select it if it is not already selected. If the checkbox is already selected, deselect it by single left-clicking on the checkbox and then select it again by single left-clicking the checkbox.

Click the "Apply" button in the "Nintendo RVL-CNT-01 Properties" window.

Verify that the four blue LED lights on the Nintendo Wii Remote flash on and off continuously for at least 30 seconds. They should continue flashing as long as the Nintendo Wii Remote is connected to the computer. If they stop flashing, repeat steps 0 through 0 or continue to section 0 to reset the Nintendo Wii Remote installation process.

Resetting the Nintendo Wii Remote Installation Process

In the “Devices” tab in the “Bluetooth Devices” window (Figure 1.2.14), select a device with a label containing the word, “Nintendo”, and press the “Remove” button.

Verify that the selected device in the “Devices” tab in the “Bluetooth Devices” window (Figure 1.2.14) is removed (it may take a few seconds for the device to be removed).

Repeat steps 0 and 0 until all devices with labels containing the word, “Nintendo”, are removed from the “Devices” tab in the “Bluetooth Devices” window.

Continue to step 0 to begin connecting Nintendo Wii Remotes.

Visual Display and Nintendo Wii Remote Sensor Bar Setup

This section details the procedures to setup the visual displays for the full and demonstration versions of the Virtual Jousting Simulator. This section also details the procedures required to operate and position the Nintendo Wii Remote sensor bar. It is assumed that the full version of the Jousting Simulator will use an LCD projector as a primary display and a standard LCD or CRT computer monitor as a secondary display. It is assumed that the demonstration version of the Jousting Simulator will use only a standard LCD or CRT computer monitor as its primary display. If installing and setting up the full version of the Virtual Jousting Simulator, follow the steps outlined in section 0. Otherwise, follow the steps outlined in section 0. When finished with the steps in either section 0 or 0, advance to section 0.

Connect the LCD projector to the computer that will run the Virtual Jousting Simulator.

Ensure that the LCD projector is connected to a power supply and turn it on.

Setup and modify the positioning of the projector screen as needed in order to achieve a desired display from the LCD projector.

Single right-click on the desktop and single-left click the "Properties" option if using Windows XP or select the "Personalize" option if using Windows Vista.

If using Windows Vista, verify that the "Personalization" window appears on the desktop. If using Windows XP, verify that the "Display Properties" window appears on the desktop.

If using Windows Vista, single left-click on the "Display Settings" option in the "Personalization" window. If using Windows XP, single left-click on the "Settings" tab.

If using Windows Vista, verify that the "Display Setting" window appears on the desktop. If using Windows XP, verify that the "Settings" screen is visible in the "Display Properties" window.

In the "Display Setting" or "Settings" screen, select the name monitor corresponding to the LCD projector from the drop-down menu containing the name of the currently selected monitor.

Single left-click on the check-boxes labeled, "This is my main monitor" and "Extend the desktop onto this monitor", to select them if they are not already selected.

Single left-click and hold the slider underneath the label, "Resolution", and drag it to the position that reads, "1024 by 768 pixels". (Note: If there is no "1024 by 768 pixels" option, then the projector may not be able to support the display requirements for the Virtual Jousting Simulator).

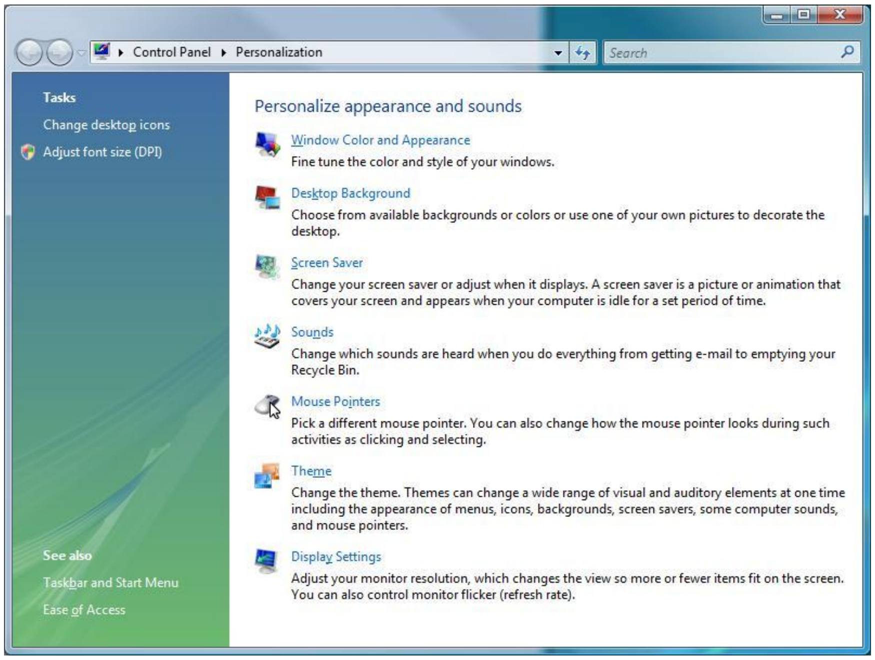
Single left-click the "Apply" button.

Acknowledge any confirmation dialog boxes by single left-clicking the "OK" or "Yes" buttons in those dialog boxes.

Close any open windows.

Single right-click on the desktop and single-left click the "Properties" option if using Windows XP or select the "Personalize" option if using Windows Vista.

If using Windows Vista, verify that the "Personalization" window appears on the desktop. If using Windows XP, verify that the "Display Properties" window appears on the desktop.



Personalize appearance and sounds

Window Color and Appearance
Fine tune the color and style of your windows.

Desktop Background
Choose from available backgrounds or colors or use one of your own pictures to decorate the desktop.

Screen Saver
Change your screen saver or adjust when it displays. A screen saver is a picture or animation that covers your screen and appears when your computer is idle for a set period of time.

Sounds
Change which sounds are heard when you do everything from getting e-mail to emptying your Recycle Bin.

Mouse Pointers
Pick a different mouse pointer. You can also change how the mouse pointer looks during such activities as clicking and selecting.

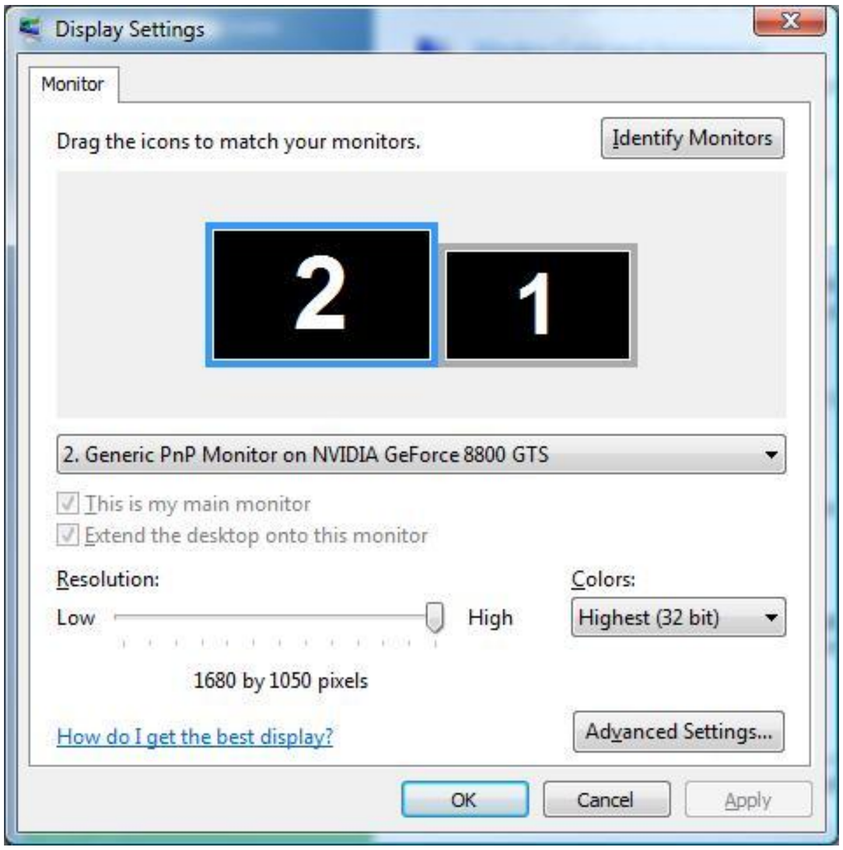
Theme
Change the theme. Themes can change a wide range of visual and auditory elements at one time including the appearance of menus, icons, backgrounds, screen savers, some computer sounds, and mouse pointers.

Display Settings
Adjust your monitor resolution, which changes the view so more or fewer items fit on the screen. You can also control monitor flicker (refresh rate).

Figure 1.3.1 – Windows Vista "Personalization" screen

If using Windows Vista, single left-click on the "Display Settings" option in the "Personalization" window. If using Windows XP, single left-click on the "Settings" tab.

If using Windows Vista, verify that the "Display Setting" window appears on the desktop. If using Windows XP, verify that the "Settings" screen is visible in the "Display Properties" window.


Display Settings

Monitor

Drag the icons to match your monitors.

2. Generic PnP Monitor on NVIDIA GeForce 8800 GTS

✓ This is my main monitor

✓ Extend the desktop onto this monitor

Resolution: Low — High

1680 by 1050 pixels

Colors: Highest (32 bit)

Figure 1.3.2 – Windows Vista “Display Settings” screen.

Single left-click and hold the slider underneath the label, “Resolution”, and drag it to the position that reads, “1024 by 768 pixels”. (Note: If there is no “1024 by 768 pixels” option, then the projector may not be able to support the display requirements for the Virtual Jousting Simulator).

Single left-click the “Apply” button.

Acknowledge any confirmation dialog boxes by single left-clicking the “OK” or “Yes” buttons in those dialog boxes.

Close any open windows.

Nintendo Wii Remote Sensor Bar Setup

Locate the Nintendo Wii Remote sensor bar.

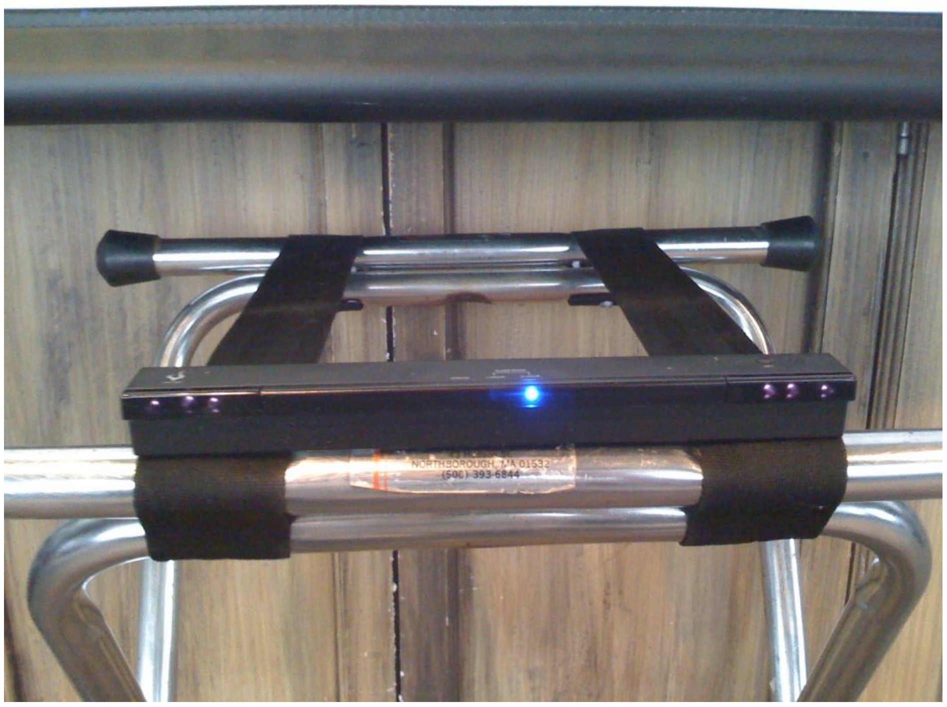


Figure 1.3.3 – A third-party Nintendo Wii Remote sensor bar.

On the Nintendo Wii Remote sensor bar, depress and release the "SET" button once to turn the sensor bar on for one hour and twice to turn the sensor bar on for two hours.

Verify that the blue LED light appears underneath the desired one or two hour option, labeled "1-HOUR" and "2-HOUR" respectively.

If using the full version of the Virtual Jousting Simulator, place the Nintendo Wii Remote sensor bar at the bottom of the projector screen with the blue LED light facing away from the screen as shown in figure 1.3.1.

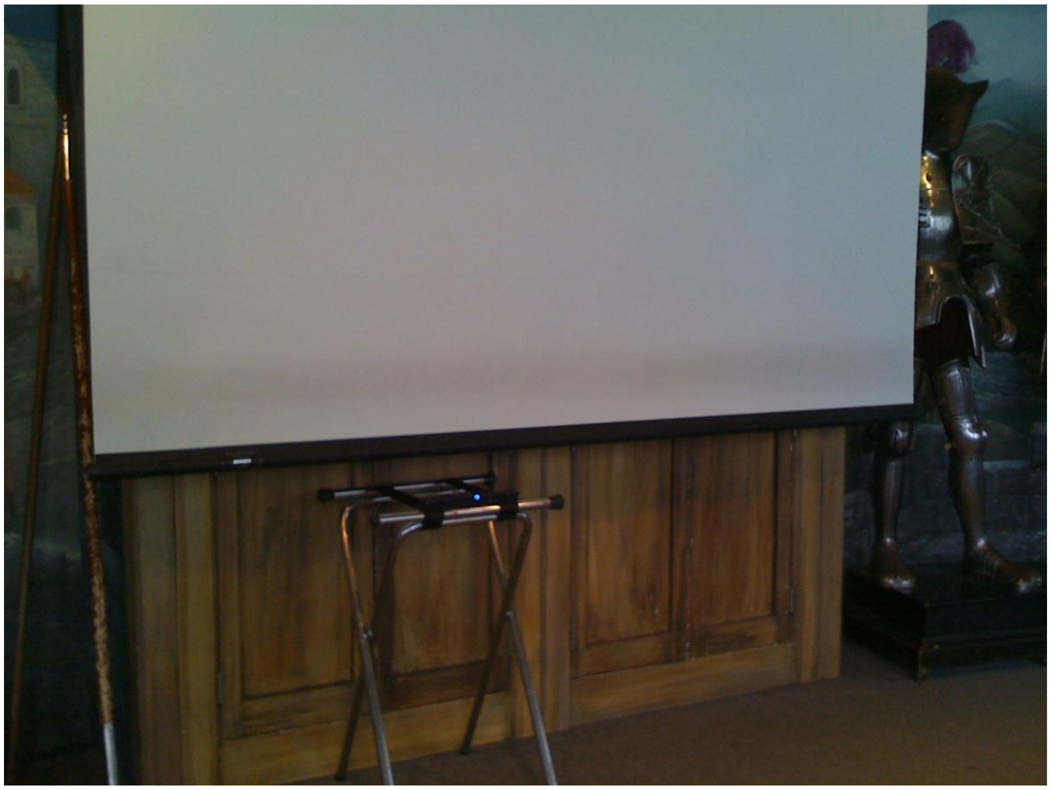


Figure 1.3.4 – A third-party sensor bar (has a single blue LED light lit) placed underneath a projector screen.

Software Installation and Setup

This section details the procedures to install and setup the Virtual Jousting Simulator software. Completing this section will allow the Virtual Jousted Simulator to be played.

Installing the Virtual Jousting Simulator Software Package

Insert the CD-ROM containing the Virtual Jousting Simulator Software Package into the CD-ROM drive.

On the desktop, single left-click the "Start" or Windows icon on the taskbar.

Single left-click on the icon labeled "My Computer" or "Computer".

Verify that a window appears containing a list of drives and devices.

Locate the device containing the Virtual Jousting Simulator Software Package and double left-click that device's icon.

Verify that a Windows Explorer window appears containing the "JoustSim" folder.

Single right-click on the "JoustSim" folder and single left-click on the "Copy" option in the resulting pop-up menu.

Single right-click on an empty area on the desktop and single left-click on the "Paste" option in resulting the pop-up menu.

Verify that the "JoustSim" folder is fully copied to the desktop (this may take a few seconds or a few minutes).

Setting up the HVA Jousting Simulator Administrative Console

Locate the "JoustSim" folder on the desktop and double left-click on its icon.

Verify that a Windows Explorer window appears containing the contents of the "JoustSim" folder.

Double left-click on the HVAJoustAdmin.jar file.

Verify that the "HVA Jousting Simulator Administrative Console" window appears.

If running the full version of the Virtual Jousting Simulator, verify that one Nintendo Wii Remote has a single blue LED light lit and the other Nintendo Wii Remote has two blue LED lights lit.

If running the full version of the Virtual Jousting Simulator, verify that the Nintendo Wii Remote with a single blue LED light lit is able to control mouse movement on the computer running the Virtual Jousting Simulator by pointing it at the Nintendo Wii Remote sensor bar. If this does not occur, restart the HVA Jousting Simulator Administrative Console by closing it and starting over from step 0.

If running the full version of the Virtual Jousting Simulator, single left-click the "Start Server" button in the "Wii Data Server" tab in the "HVA Jousting Simulator Administrative Console" window.

Single left-click the "XML Data Server" tab in the "HVA Jousting Simulator Administrative Console" window.

Verify that the "XML Data Server" screen appears in the "HVA Jousting Simulator Administrative Console" window.

Click the "Start Server" button.

Setting up the Virtual Jousting Simulator Application

Locate and single left-click on the Windows Explorer window containing the contents of the "JoustSim" folder.

Double left-click on the JoustSim.exe file.

Verify that the Virtual Jousting Simulator application appears in a "Macromedia Flash Player <X>" window, where X is the current version of the Macromedia Flash Player installed on the computer running the Virtual Jousting Simulator software.

Single left-click on the Macromedia Flash Player icon in the top-right corner of the Virtual Jousting Simulator application window and select the "Maximize" option in the resulting pop-up window.

Verify that the Virtual Jousting Simulator application window covers the entire screen.

Simultaneously press and release the "Ctrl" and "F" keys on the keyboard.

Verify that the Windows taskbar and menu bar on the Virtual Jousting Simulator application window are no longer visible.

Shutting Down

This section details the steps required to shut down software and hardware components of the Virtual Jousting Simulator. To shut down the entire system, all steps in this section must be completed in order and in their entirety.

Shutting Down the Software

Closing the Virtual Jousting Simulator Application

To close the Virtual Jousting Simulator Application, simultaneously press and release the "Ctrl" and "F" keys to minimize the application and single left-click the "X" button in the top-right corner of the application window.

Closing the HVA Jousting Simulator Administrative Console

To close the HVA Jousting Simulator Administrative Console, single left-click the "X" button in the top-right corner of the application window.

Shutting Down the Hardware

Disconnecting the Nintendo Wii Remotes

To disconnect all Nintendo Wii Remotes, follow the steps outlined in section 0 with the exception of step 0.

Turning Off the third-party Nintendo Wii Remote Sensor Bar

To turn off the third-party Nintendo Wii Remote sensor bar, depress the "SET" button on the sensor bar until the blue LED light on the sensor bar turns off.

System Administration, Maintenance, and Customization

Hardware Maintenance

Recharging the Nintendo Wii Remotes

A single Nintendo Wii Remote is powered by two AA batteries. If it is believed that a Nintendo Wii Remote is low on power, these batteries can either be replaced, or recharged if they are of the rechargeable type.

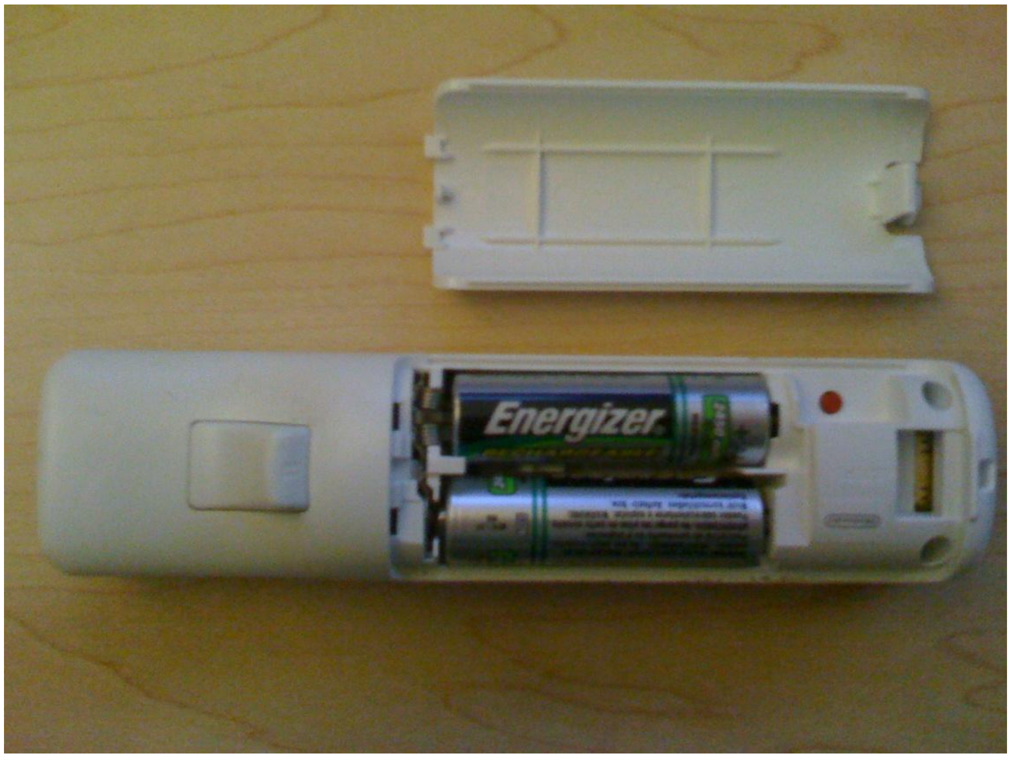


Figure 3.1.1 – A Nintendo Wii Remote with its AA batteries exposed.

Recharging the Nintendo Wii Remote Sensor Bar

The third-party Nintendo Wii Remote sensor bar has an internal power source that can only be recharged via its USB interface. To recharge the third-party Nintendo Wii Remote sensor bar, locate a USB-to-Mini-USB cable (figure 3.1.2), plug the smaller, Mini-USB end of the cable into the sensor bar (figure 3.1.3) and plug the larger USB end of the cable into a computer that is powered on.

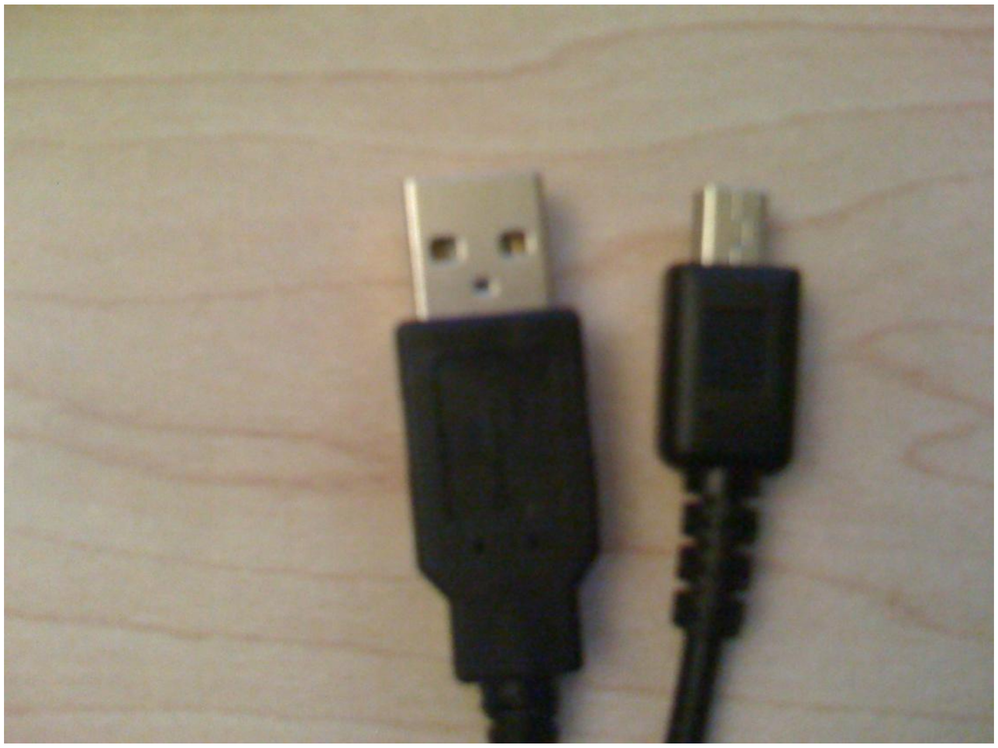


Figure 3.1.2 – A USB-to-Mini-USB cable (Left-end: USB, Right-end: Mini-USB).

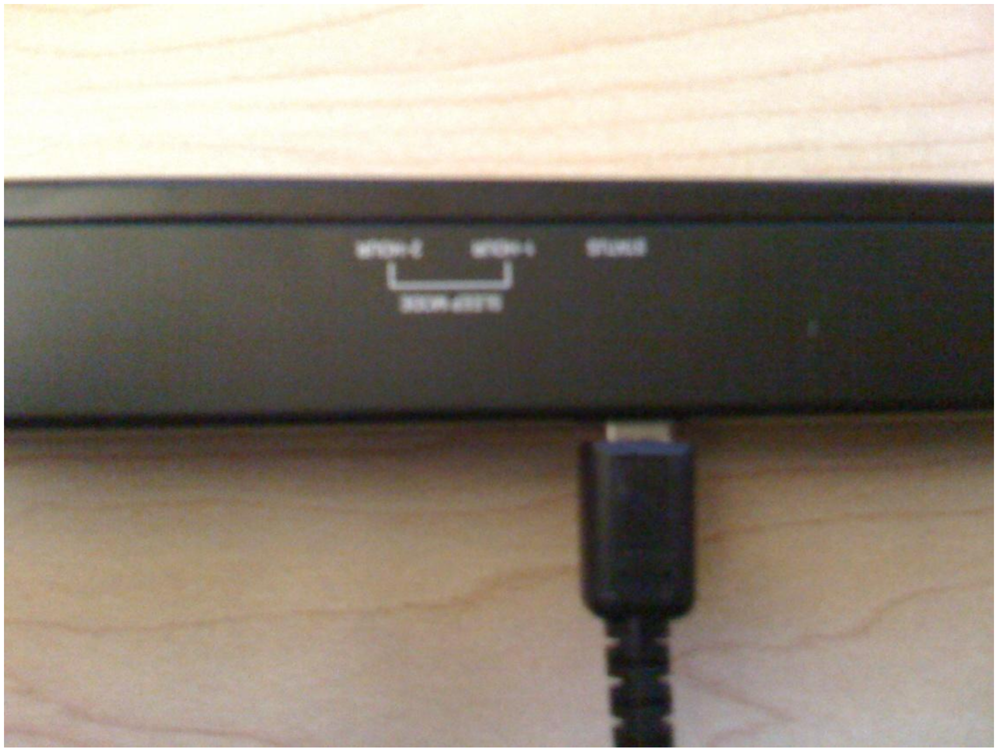


Figure 3.1.3 – Mini-USB end of cable plugged into the third-party Nintendo Wii Remote sensor bar.

OPERATING THE NINTENDO WII REMOTE SERVER

OVERVIEW

The Nintendo Wii Remote data server is a component of the HVA Jousting Simulator Administrative Console application, which allows for communication between the Nintendo Wii Remotes and the Virtual Jousting Simulator. When the HVA Jousting Simulator Administrative Console application is initialized, it will detect any Nintendo Wii Remotes that are connected to the host computer. The server will only allow communication between these Nintendo Wii Remotes and any connected applications until the administrative console is restarted.

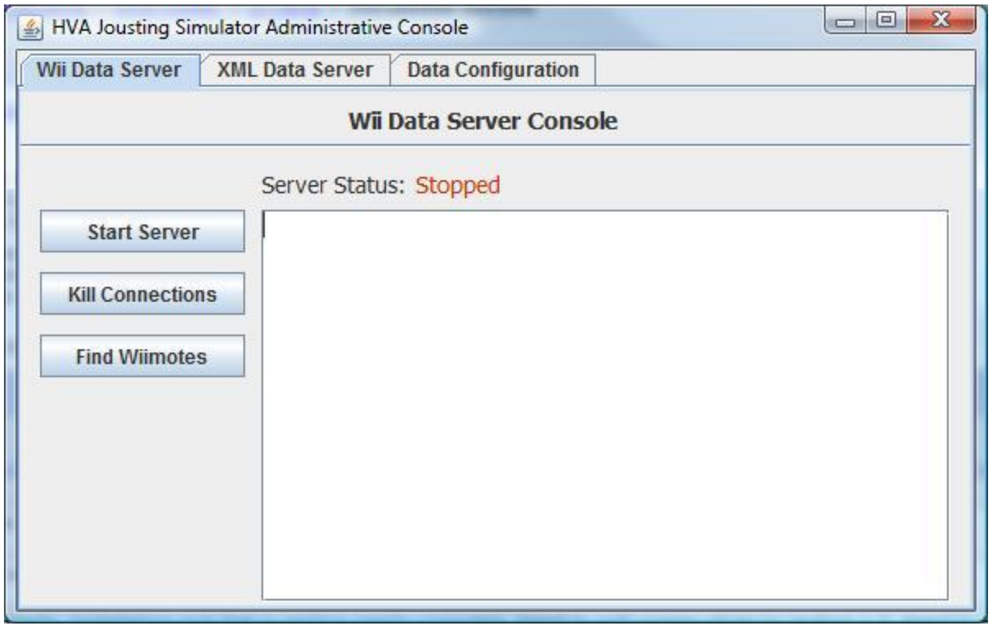


Figure 3.2.1 – The Nintendo Wii Data Server Console

STARTING THE SERVER

Starting the Nintendo Wii Remote server will cause it to listen for new connection requests from other applications (i.e. the Virtual Jousting Simulator).

To start the server, single left-click the "Start Server" button in the Wii Data Server Console. Note that the "Start Server" button will change to a "Stop Server" button after it is pressed.

STOPPING THE SERVER

Stopping the Nintendo Wii Remote server will cause it to stop listening for new connection requests from other applications. This will not sever existing connections to the server and will allow for applications already connected to the server to continue communicating with any Nintendo Wii Remotes that are connected.

To stop the server, single left-click the “Stop Server” button in the Wii Data Server Console. Note that the “Stop Server” button will change back to a “Start Server” button after it is pressed.

Severing Existing Connections

Severing existing connections to the Nintendo Wii Remote Server will stop communication between any connected applications and the Nintendo Wii Remotes.

To sever existing connections, single left-click the “Kill Connections” button in the Wii Data Server Console.

Detecting Nintendo Wii Remotes

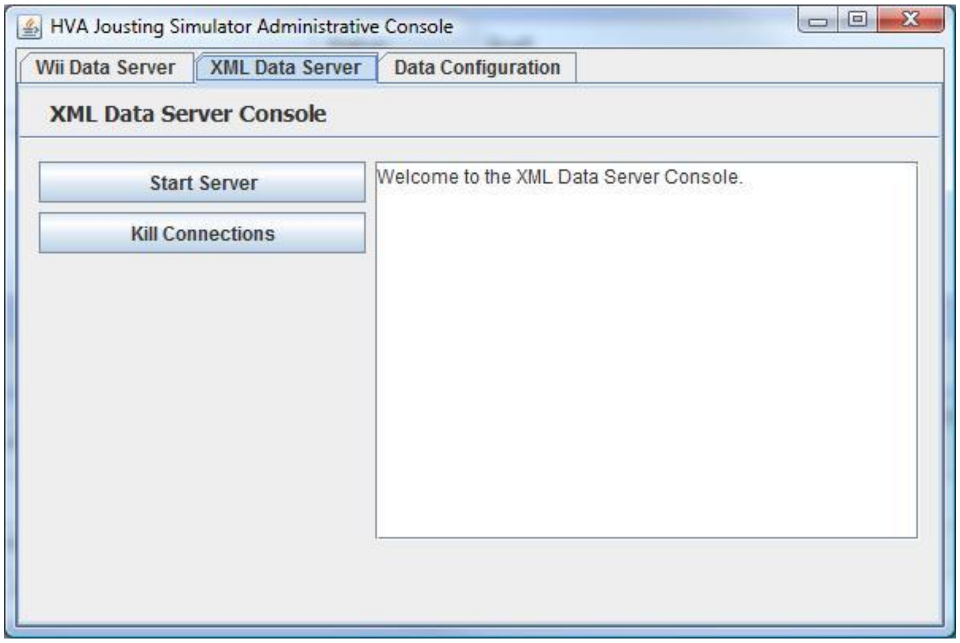
The Wii Data Server Console has the ability to find new Nintendo Wii Remotes that are connected to the host computer after the HVA Jousting Simulator Administrative Console application has been started and allow connections made from applications to communicate with these Nintendo Wii Remotes.

To find new Nintendo Wii Remotes, single left-click the “Find Wiimotes” button in the Wii Data Server Console.

Operating the XML Data Server

Overview

The XML Data server is a component of the HVA Jousting Simulator Administrative Console application, which allows for the Virtual Jousting Simulator to write information to essential files, which include data in regards to player information, scoring, and several other important features. It is essential for this server to be running before starting the Virtual Jousting Simulator. If this server is not running, then the Virtual Jousting Simulator will not function correctly.



Welcome to the XML Data Server Console.

Figure 3.3.1 – XML Data Server Console

Starting the Server

Starting the XML data server will cause it to listen for new connection requests from other applications (i.e. the Virtual Jousting Simulator).

To start the server, single left-click the "Start Server" button in the XML Data Server Console. Note that the "Start Server" button will change to a "Stop Server" button after it is pressed.

Stopping the Server

Stopping the XML data server will cause it to stop listening for new connection requests from other applications. This will <u>not</u> sever existing connections to the server.

To stop the server, single left-click the "Stop Server" button in the XML Data Server Console. Note that the "Stop Server" button will change back to a "Start Server" button after it is pressed.

Severing Existing Connections

Severing existing connections to the Nintendo Wii Remote Server will stop communication between any connected applications and the Nintendo Wii Remotes.

To sever existing connections, single left-click the "Kill Connections" button in the XML Data Server Console.

Content Customization and Maintenance

Overview

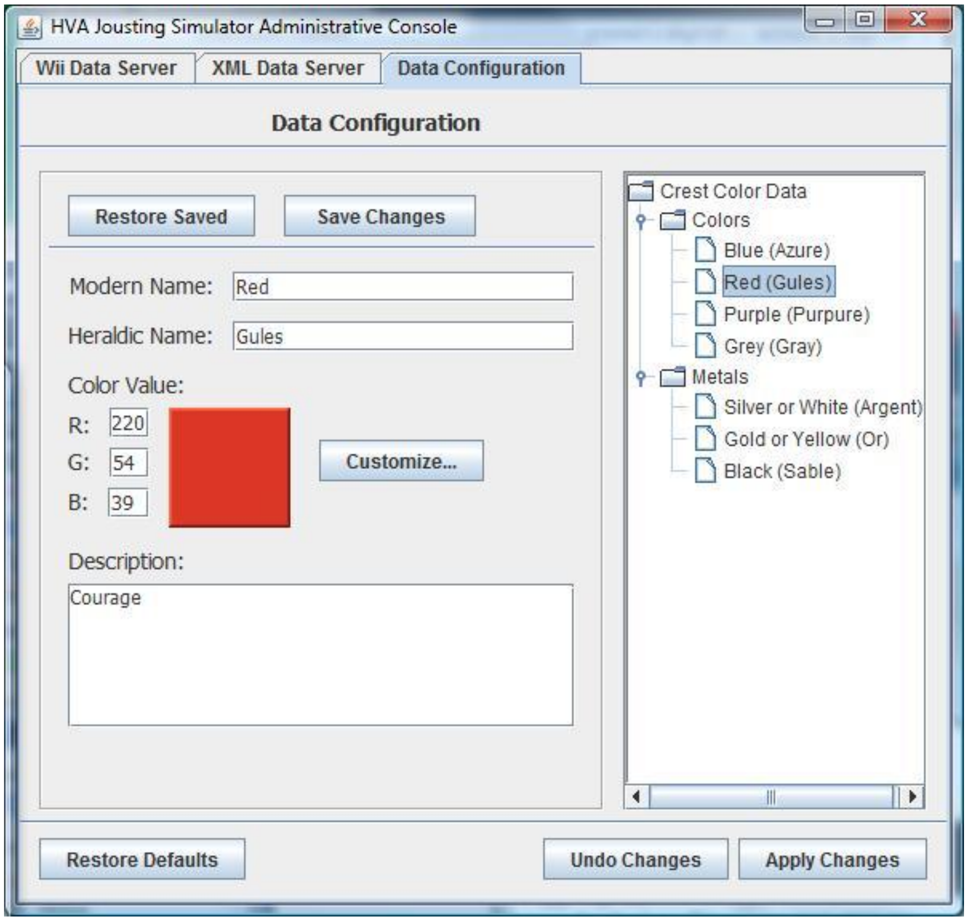
The Data Configuration panel is a component of the HVA Jousting Simulator Administrative Console. This panel allows for the modification of the heraldic crest colors for the heraldic crest customizer component of the Virtual Jousting Simulator.

Customizing Heraldic Crest Content

The individual colors and metals in the heraldic crest customizer can be edited by double left-clicking on either the "Colors" or "Metals" folder icon in the left pane of the Data Configuration panel and single left-clicking on a color or metal item (labeled by [Color] ([Heraldic Name])).

Clicking on any of these items will bring put a color configuration panel, where the modern name, heraldic name, color values (R = red value, G = green value, B = blue value), and description can be modified. When the desired changes have been made, single left-click the "Save Changes" button, or single left-click the "Restore Saved" button to restore the original values of the color.

After the desired number of colors and/or metals have been modified, click the "Apply Changes" button to write these changes to disk and make them permanent.



Data Configuration

Modern Name: Red

Heraldic Name: Gules

Color Value:

R: 220

G: 54

B: 39

Description:

Courage

Crest Color Data
- Colors
  - Blue (Azure)
  - Red (Gules)
  - Purple (Purpure)
  - Grey (Gray)
- Metals
  - Silver or White (Argent)
  - Gold or Yellow (Or)
  - Black (Sable)

Figure 3.4.1 – Data Configuration panel with the color, "Red (Gules)" selected.

Troubleshooting

Known Hardware Issues

The Nintendo Wii Remote with a single blue LED lit does not control the mouse on the PC running the Virtual Jousting Simulator after the HVA Jousting Simulator Administrative Console has been started.

Solution: Try restarting the HVA Jousting Simulator Administrative Console. If the problem is not resolved after more than three restarts, it may be possible that the Nintendo Wii Remote meant to control the mouse is low on power, and must either be recharged or have its batteries replaced.

The Nintendo Wii Remote with a single blue LED lit is demonstrating unusual or buggy mouse behavior.

Solution: Ensure that the Nintendo Wii Remote controlling the mouse is at least 10 feet away from the third-party Nintendo Wii Remote sensor bar. The Nintendo Wii Remote infrared sensors are also very sensitive to infrared light sources other than those from the sensor bar (such as sunlight, candle-light, etc). Ensure that the shiny, black infrared sensor is not exposed to any infrared light sources other than from the sensor bar (close window blinds, douse candles, etc).

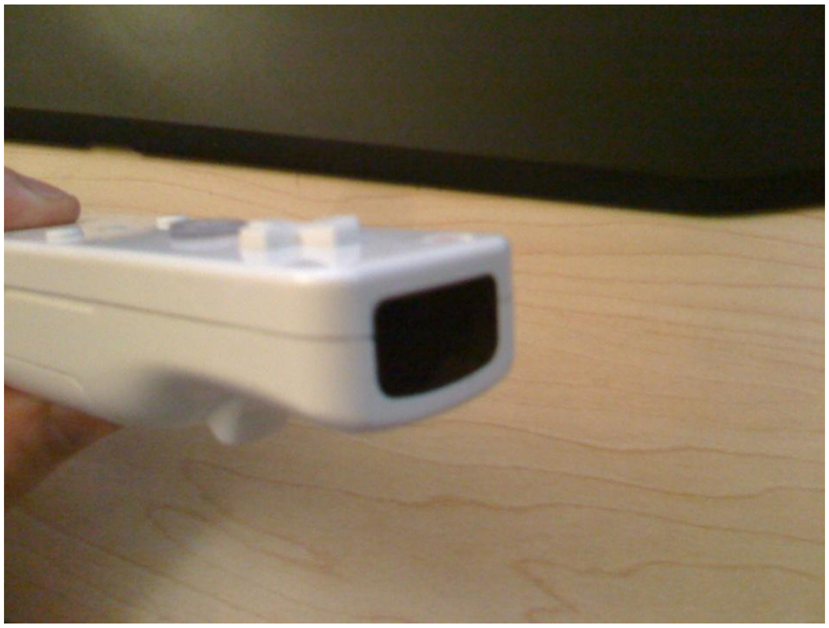


Figure 4.1.1 – The infrared sensor (black rectangle) on the Nintendo Wii Remote.

One or more of the Nintendo Wii Remotes will not remain connected to the PC running the Virtual Jousting Simulator.

Solution: This is most likely due to either an incorrect connection or low power for the Nintendo Wii Remote(s) in question. To ensure that the Nintendo Wii Remote(s) are properly connected, follow the steps in section 0 to remove the existing Nintendo Wii Remote connection(s) and then follow the steps in section 0 to reconnect the Nintendo Wii Remote(s). If the problem persists, it is most likely that the Nintendo Wii Remote(s) is/are low on power and must either be recharged or have its batteries replaced.

The third-party Nintendo Wii Remote sensor bar made several loud beeping noises and then turned off unexpectedly.

Solution: This is normal behavior for the third-party Nintendo Wii Remote sensor bar. The beeping noise is an indication that its timer has expired and will turn off. Follow the steps in section 0 to turn the third-party Nintendo Wii Remote sensor bar back on.

The third-party Nintendo Wii Remote sensor bar turns off unexpectedly and/or the blue LED light is flashing.

Solution: This is an indication that the third-party Nintendo Wii Remote sensor bar is low on power and must be recharged. Follow the steps in section 0 to recharge the third-party Nintendo Wii Remote sensor bar.

Gameplay Issues

Moving the lance up and down is very "jump" or buggy.

Solution: Ensure that the Nintendo Wii Remote controlling the lance is oriented such that the buttons are facing upward toward the ceiling (figure 4.2.1) and not sideways towards the walls (figure 4.2.2). The accelerometer microchip inside the Nintendo Wii Remote is known to have some issues when it is oriented in a sideways position and will most likely cause unexpected behavior in the Virtual Jousting Simulator.

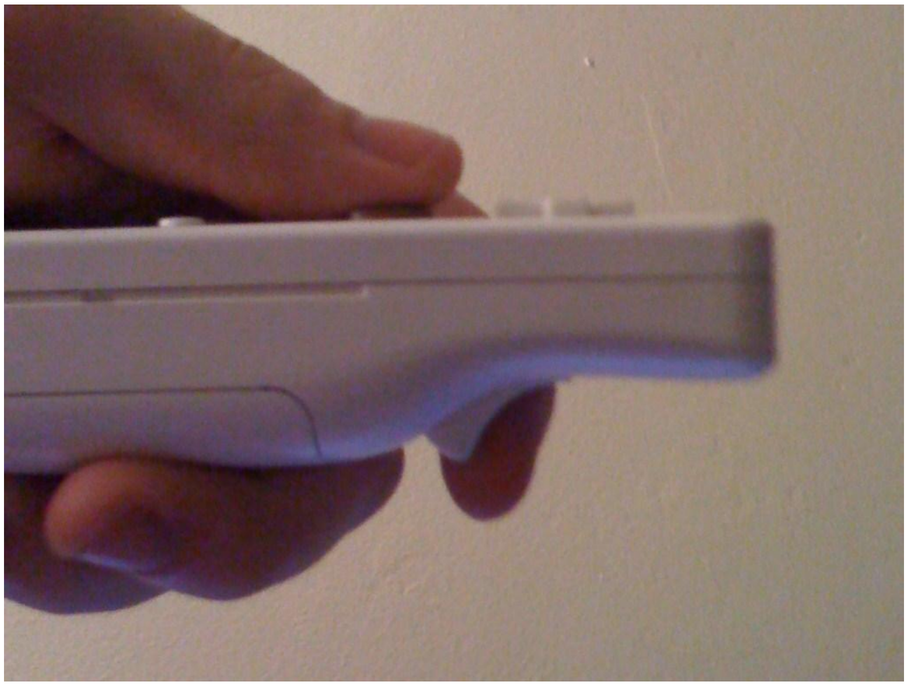


Figure 4.2.1 – Correct orientation of Nintendo Wii Remote for controlling the lance.

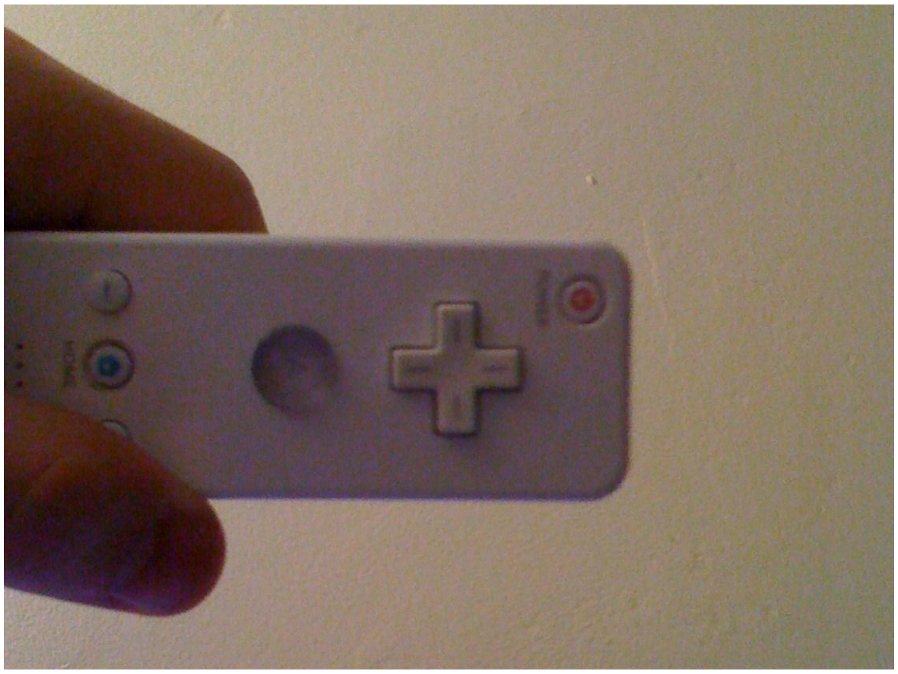


Figure 4.2.2 – Incorrect orientation of Nintendo Wii Remote for controlling the lance.

The horse controller does not seem to be registering left and right movements for the horse.

Solution: Ensure that the Nintendo Wii Remote controlling the horse is being held correctly. Refer to figure 4.2.3 and 4.2.4 for the correct position to hold the horse controller. Refer to figures 4.2.5 and 4.2.6 for the correct right and left movements.

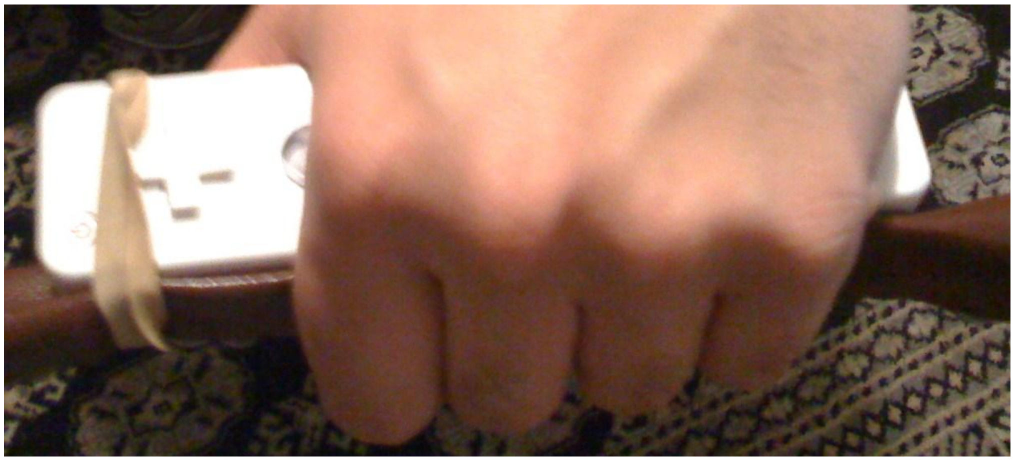


Figure 4.2.3 – Correct orientation of horse controller.

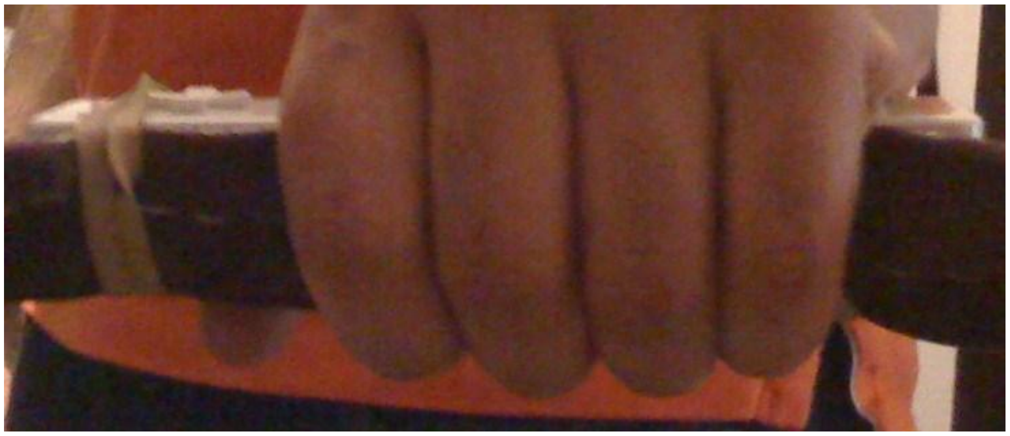


Figure 4.3.4 – Correct orientation of horse controller.

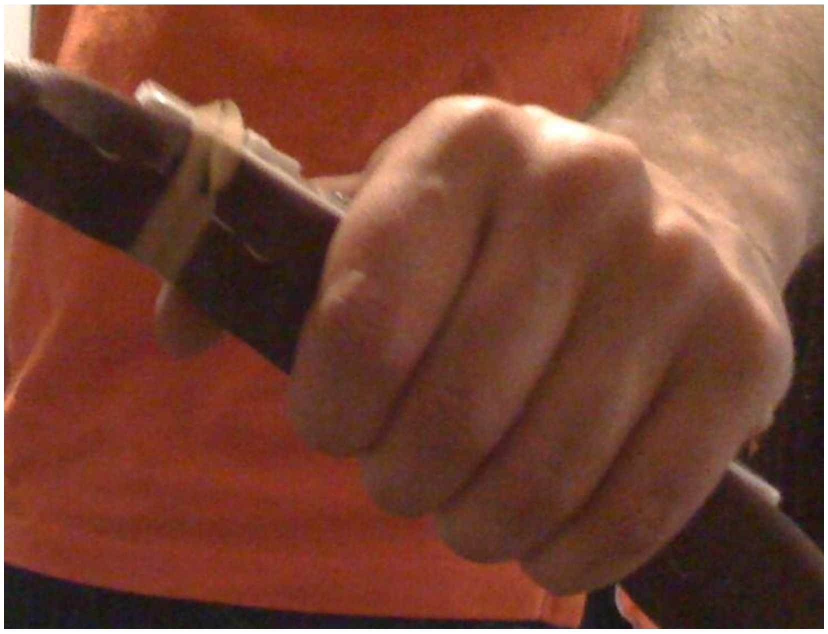


Figure 4.2.5 – Tilting the horse controller left.

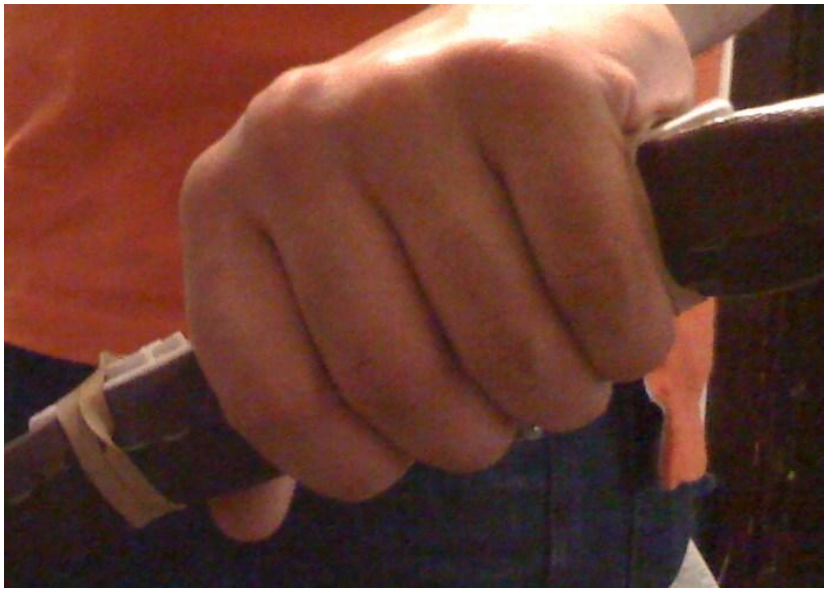


Figure 4.2.6 – Tilting the horse controller right.

THE LANCE OCCASIONALLY DOES NOT SEEM TO MOVE LEFT AND/OR RIGHT WHEN DIRECTED TO DO SO WITH THE LANCE CONTROLLER.

Solution: Ensure that the Nintendo Wii Remote that is controlling the lance is pointed at the third-party Nintendo Wii Remote sensor bar when attempting to move the lance left and right. Moving the lance controller too far to the left or right may cause the infrared sensor in the Nintendo Wii Remote to go out of range of the sensor bar and not be able to make the lance move left and right.

THE MOUSE, LANCE, OR HORSE MOVEMENTS SEEM TO BE "LAGGING" OR LESS RESPONSIVE THAN THEY SHOULD BE.

Solution: When the Virtual Jousting Simulator is run for extended periods of time, a "lagging" issue may occur. To fix this problem, close the Virtual Jousting Simulator application and the HVA Jousting Simulator Administrative Console and repeat the steps in sections 0 and 0 to restart these applications.

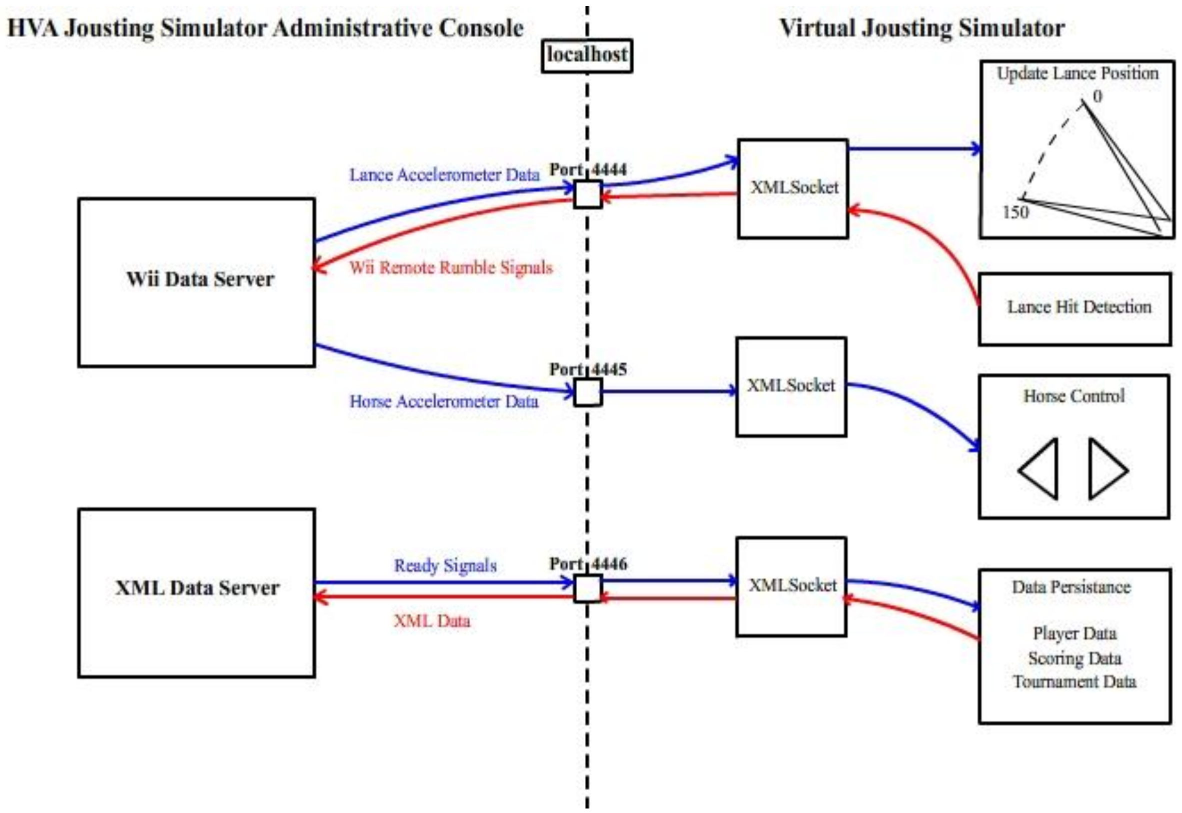
Developer's Guide

The Virtual Joust

TABLE OF CONTENTS

CLIENT-SERVER DATA COMMUNICATION

A basic client and server scenario was developed for the Virtual Jousting Simulator due to the limitation of Flash in regards to communicating with Bluetooth devices and writing to XML files. The client in this case is the Virtual Jousting Simulator Flash application and the server is the HVA Virtual Jousting Administrative console. All communication between the client and server if performed using the Transmission Control Protocol and the Internet Protocol, which is commonly referred to as TCP/IP. The overall data-flow for this client and server setup is shown in the diagram below:



The HVA Jousting Simulator Administrative Console is responsible for running and maintaining two servers, the Wii Data Server and the XML Data Server. The Wii Data Server is responsible for listening for Nintendo Wii Remote events using the WiiuseJ API, such as button presses or accelerometer movements, and sending some of these events to the Virtual Jousting Simulator client. The Virtual Jousting Simulator client uses this information to determine the position of the virtual lance and the controls associated with horse movements. The client also sends signals back to the Wii Data Server, which the server uses to send signals to the Nintendo Wii Remotes to make them rumble. The XML Data server is responsible for listening for XML data to be sent to it from the client, which is then subsequently written to a specified file. The XML Data Server will send signals back to the client to notify it that it is either ready to write to a file specified from a previous transmission or has finished writing received data to a file.

Both servers in the HVA Jousting Simulator Administrative Console listen for TCP/IP connection requests using Java ServerSockets. The client establishes these connections through the use of Flash XMLSockets. The Wii Data Server specifically listens for connections from two different ServerSockets in two concurrent processes running in Java Threads.

One of these ServerSockets, listening on port 4444, is responsible for accepting connections requesting to send and receive data to and from the Nintendo Wii Remote that controls the virtual lance. When one of these connections is made, it is transferred to its own Thread to handle communication concurrently. Whenever the Nintendo Wii Remote controlling the lance moves, the server will send a message to the client in the following format:

<pl>[0 – 100]</pl>

This message will specify a value between 0 and 100 in the place-holder marked, "[0-100]", which corresponds to pitch value of the Nintendo Wii Remote, where 0 is level with the ground and 100 when it is in the upright position. The client uses this value to determine how to display the orientation of the virtual lance.

The client can also send "rumble" signals back to the Wii Data Server, which will cause the Nintendo Wii Remote to vibrate for 300 milliseconds. The message sent to the Wii Data Server to make the Nintendo Wii Remote rumble is the following:

<1 \>

The second ServerSocket on the Wii Data Server listens for connections on port 4445 requesting to receive data from the Nintendo Wii Remote that is being used as the horse controller. When a connection is made, the connection is transferred to its own Thread to handle communication concurrently, similar to the behavior of the server controlling lance connections. Whenever the Nintendo Wii Remote controlling the horse moves, the server will send a message to the client in the following format:

<h><p>[pitch]</p><r>[roll]</r></h>

This message will specify a value between -100 and 100 for pitch and roll in the place-holders marked "[pitch]" and "[roll]" respectively. The client uses this data to determine whether the user is trying to move the horse left, right, or not at all.

As stated above, the XML Data Server is responsible for writing XML data to specified XML files with data sent from the client, since Flash does not have the ability to write to files on its own. The basic flow of events for writing data to a file from the client is as follows:

1. Client sends a request to the server to write to a file by specifying the filename in the following format:
   <filename>[full path to file and filename</filename>
2. The server prepares to write to the specified file and sends the following confirmation message to the client to say that it is ready to write:
   <1 />
3. When the ready signal is received, the client sends the XML data it wants to be written to the previously specified file in its entirety to the server.
4. The server receives the XML data, writes it to the specified file, and sends an identical confirmation signal from step 2 to the client when it is finished.

GAME DATA FILE

This section provides information about the various files that are needed to run the game. It specifies the location of the file and what it does. It will also include information about the contents of the file. All of these files may be located within the data folder. If they are not there, the behavior of the program is not defined. Some of the files the game can run without, others it must have for proper functioning.

COLOR_DATA.XML

This file's primary use is to store the information about the colors used on the shields of the knights. The colors contained within the file will be combined to form the different color combinations given to the player. Each color is combined with a metal to form a color choice. Another choice with the reverse, the metal being the primary color and the color the secondary, will also be generated. Adding extra entries into this file will create more choices for the player to pick or allow you to edit the existing colors

Table 1 Tags in color_data.xml

Tag Name	Purpose	Child of
<color_data>	Start of the color_data file	Base node
<colors>	Beginning of list of colors	<color_data>
<color>	Start of a color data entry	<colors>
<name>	The name of the color	<color>
<heraldic_name>	Heraldic name of the color	<color>
<hex>	The hex value for the color	<color>
<description>	What the color means	<color>
<metals>	Beginning of list of metal colors	<color_data>
<metal>	Start of a metal data entry	<metals>

CURRENTPLAYER.XML

This file contains information about the current person that is playing the game. It can also be used to transfer data between scenes when needed. This file has the unique use that it is used to identify if the player is new. If the name of the player is "NEW PLAYER" the game will recognize this as a signal that the player is new. For new players, the file will be partially created after they finish the crest customizer. For returning players, the file will be populated with their data immediately after it is found.

As part of the scene data transfer function of this file, the results from the player's jousting will be placed into this file. This data will not be present until after the player has finished jousting all four rounds.

Table 2 Tags in currentPlayer.xml

Tag Name	Purpose	Child of
<currentPlayer>	The start of the currentPlayer file	Base node
<name>	The name of the player	<currentPlayer>
<herald>	Start of heraldry information	<currentPlayer>
<primary>	Start of primary color information	<herald>
<secondary>	Start of secondary color information	<herald>
<name>	Name of shield color	<primary>,<secondary>
<heraldic_name>	Heraldic name of the color	<primary>,<secondary>
<hex>	The hex value for the color	<primary>,<secondary>
<description>	Repeat of the hex value	<primary>,<secondary>
<emblem>	Number code for player's emblem	<herald>
<design>	Number code for player's design	<herald>
<PlayerScoringRecord>	Start of data about past player scores	<player>
<High_Score>	The current high score for this player	<PlayerScoringRecord>
<scoring>	Begin jousting results	<currentPlayer>
<RoundOne>	Start entry for round one results	<scoring>
<RoundTwo>	Start entry for round two results	<scoring>
<RoundThree>	Start entry for round three results	<scoring>
<RoungFour>	Start entry for round four results	<scoring>
<result>	What happened for a round	<RoundOne>, <RoundTwo>, <RoundThree>, <RoundFour>
<HitLocation>	The area that was hit this round	<RoundOne>, <RoundTwo>, <RoundThree>, <RoundFour>

PLAYER_PROFILE.XML

This file is used as a proxy database. In it, old player information is stored so that it can be retrieved at a later time. The player's name is used as the key for the entries. There is no protection against entering the same name multiple times but this will not cause any crashes. The instance of that name will be called instead.

Table 3 Tags in player_profile.xml

Tag Name	Purpose	Child of
<player_profile>	The start of the player profile data	Base node

<player>	Start of a player entry	<player_profile>
<name>	The name of the player	<player_profile>
<herald>	Start of heraldry information	<player_profile>
<primary>	Start of primary color information	<herald>
<secondary>	Start of secondary color information	<herald>
<name>	Name of shield color	<primary>,<secondary>
<heraldic_name>	Heraldic name of the color	<primary>,<secondary>
<hex>	The hex value for the color	<primary>,<secondary>
<description>	Repeat of the hex value	<primary>,<secondary>
<emblem>	Number code for player's emblem	<herald>
<design>	Number code for player's design	<herald>
<PlayerScoringRecord>	Start of data about past player scores	<player>
<High_Score>	The current high score for this player	<PlayerScoringRecord>

TAB_DATA.XML

This file contains the information that is shown on the tabs of the crest customizer. This allows people to edit the wording if necessary later.

Table 4 Tags in tab_data.xml

Tag Name	Purpose	Child of
<TabData>	The start of the tab data	Base node
<DesignTab>	Text for the design tab	<TabData>
<DesignTabText>	The text displayed when the design tab is selected	<TabData>
<ColorTab>	Text for the color tab	<TabData>
<ColorTabText>	The text displayed when the color tab is selected	<TabData>
<EmblemTab>	Text for the emblem tab	<TabData>
<EmblemTabText>	The text displayed when the emblem tab is selected	<primary>,<secondary>

TOURNAMENTNAME.XML

A small file that holds the number code for what the name of the next tournament will be.

Table 5 Tags in tournamentName.xml

Tag Name	Purpose	Child of
<tournamentName>	The start of the tournament name data	Base node
<tournamentSymbol>	The symbol code that is used to determine the name of the next tournament	<tournamentName>

External ActionScript Files

Importing external ActionScript files

Before these files may be used in a scene, they must be imported into that scene. This is done by using the import command. If the class is not in the same directory as the fla document, then you will need to specify the path. For the classes in the as folder, you would need to do import as.Foo. The extra as must be part of the classes name or flash will not recognize it. External ActionScript files must be loaded once per scene. These imported ActionScript files do not carry over scene to scene.

CrestColor

This class contains the data for a single color set. Each set contains a primary color and secondary color along with its description.

CrestColor Constructor

This will create a new CrestColor class. It requires as parameters the primary and secondary colors in hex and then the description strings for each.

setPrimary

This method allows you to change the primary color of this CrestColor. It requires you to provide the new color as a hex value and the new description for the color.

setSecondary

The same as setPrimary except deals with the secondary color for the shield.

Getters

This class has a list of getters for the colors contained within it. The names of the getters explain what will be returned.

- getPrimaryColor
- getSecondaryColor

- getPrimaryColorDescription
- getSecondaryColorDescription

CurrentPlayer

This class is used to retrieve information about the current player. It will load the currentPlayer.xml document and allow you to access the information contained within it. The currentPlayer.xml contains all the necessary information about the player that is currently playing the game.

CurrentPlayer constructor

CurrentPlayer's constructor has one parameter, a string that is either the path to the currentPlayer.xml file or null. If it is called with null then it will use the default path. There is no reason to send anything other than null to the CurrentPlayer class unless you are calling the administrator console from somewhere root folder of this project.

getFile

This method will return to you the XML gateway into the file.

isNewPlayer

Determines if the current person playing is a new player. If the file has not yet loaded, a trace will appear if debugging and the function will return false. True will be returned if the player is indeed new.

Getters

There are a large number of getter methods in this class to retrieve information about the current player. All getters that are in the player information section below are accessible at any time that you know you have an old player. The getters in the final scoring section are not available until the joust is finished. The getters getGridResult and getGridHitArea will require the round number you want returned as a parameter. In addition, all getters of this class will return a string except for getTotalScore.

Player Information Getters

- getPrimaryColorName
- getHeraldicName
- getPrimaryColorHex
- getPrimaryColorDescription
- getSecondaryColorName
- getHeraldicNameSecondary
- getSecondaryColorHex
- getSecondaryColorDescription
- getEmblem

- getDesign
- getPlayerSymbol
- getName
- getHighScore

Final Scoring

- getR1Result
- getR2Result
- getR3Result
- getR4Result
- getGridResult
- getGridHitArea
- getTotalScore

HitCalculator

This class is responsible for calculating the result of the joust. It requires that its helper methods be called first to set the numbers used to calculate the score. This allows the scoring system to be extended as any new aspects of the score can be added by creating a new setter for this class and adding that new number to the finalScore.

Calculate

After all the necessary helper methods have been called, this will method will return to you the joust result. If you want to add additions scoring items, create a new private variable and setter to store the new score item. Then add your new scoring piece to the line that reads

```
finalScore = points + this.horseScore + this.lanceScore;
```

The constants UNHORSE_THRESHOLD and LANCESHATTER_THRESHOLD will likely needed to be changed if new scoring aspects are added. These represent the score that is necessary to have one of these events occur. Changing the value of it will result in a larger score needed to get that result.

getTotalScore

This method will return to the caller a number with the final score of the round. Will not function correctly until calculate has been called.

Setters

HitCalculator has a setter for each of the parts of the scoring system. These setters need to be called to set the values of the numbers for the round of jousting before the calculate method is called. Listed below are the setters methods in this class and what argument they require.

Name of Setter	Parameters
setHitPoint	HitPoint class
setHorseScore	Number
setLanceScore	Number

HitPoint

This class is used to emulate enums. In it, the locations that can be hit and their point values are declared. Adding new hit locations can be done by adding the following to the end of the list

static public var nameOfHitPoint:HitPoint = new HitPoint("Name of area", score for area)

Calling one of the items from the class requires that you import it into the scene. After that, you get the string of the hit area by using HitPoint.nameOfHitPoint and can get the score of that area by using the getPoints method.

forName

Returns the enum object for a given string. Requires as a parameter the name of the object you want.

getLocation

Get the location of the object as a string.

getPoints

Get the number of points your hit location is worth

equals

Determines if the given HitPoint is the same as itself. Will return true if it is, false otherwise

JoustResult

Similar to HitPoint, the JoustResult class is used as an emulated an enum to give different options for joust results. To add a new JoustResult, add the following to the end of the list of current joust results

static public var result_Name:JoustResult = new JoustResult('result name', 'Text to display on hit')

getResult

Return the result string for this object

getCause

Return the cause string for this object

LoadOldPlayer

This class is used to load old player information from the playerProfile.xml. When created, the path to this file needs to be specified so the class can find it.

getFile

A getter for the XML data file that is used in the object.

getArrayOfPlayers

This method is used to generate an array with the names of all past players. The player names must come from a valid playerProfile.xml file.

findOldPlayer

Searches the opened playerProfile.xml file for an old player. This requires that you pass a string containing the name of the player to search for in the file. If this name is found, the method will return true. Otherwise the method will return false.

getBaseNode

This method will return the node of the player whose name matches the input given in the parameter. This method is useful for removing data from the playerProfile.xml . If it is unable to find the name of the player whose named is inputted, the method will return null.

HorseControlRating

This class keeps track of the horse control performance during a run. Horse control is done by having arrows pop up randomly on the screen and the player needing to react to them. If react quickly enough, they will score points.

update

This method is used to update the horse control scores. The parameter, hType, refers to how they did on a given horse control stage. 1 is for poor control, 2 for fair and 3 is for good.

getGoodControl

Returns the number of instances of when the horse was in good control.

getFairControl

Returns the number of instances of when the horse was in fair control.

getPoorControl

Returns the number of instances of when the horse was in poor control.

incrementGoodControl

Increases the number of good horse control instances by 1.

INCREMENTFAIRCONTROL
Increases the number of fair horse control instances by 1.

INCREMENPOORCONTROL
Increases the number of poor horse control instances by 1.

CALCULATESCORE
Determines the average of the horse control instances. This is used for the score for the horse control.

TABDATA
This class contains the text that is displayed in the crest customizer screen on the tabs and the descriptions of what to on each tab.

TABDATA CONSTRUCTOR
To create an instance of this class, six arguments are needed. The first three are the strings for the tabs themselves. These strings will be displayed on the tabs. The next three are the strings for the description of each of the tabs. This is only displayed when that tab is selected. The order of the arguments is emblem, color, design with the text to be on the tab first and the description strings as the latter three arguments.

GETEMBLEMTAB
Return the text as a string that should be placed in the emblem tab.

GETCOLORTAB
Return the text as a string that should be placed in the color tab.

GETDESIGNTAB
Return the text as a string that should be placed in the design tab

GETEMBLEMTEXT
Return the description of the emblem tab as a string.

GETCOLORTEXT
Return the description of the color tab as a string.

GETDESIGNTEXT
Return the description of the design tab as a string.

XMLCOLORDATA
This class retrieves the necessary data and creates the grid of possible color choices the player has.

XMLColorData Constructor

To create a new XMLColorData class, the path to a color_data.xml file is needed. This file contains the needed information to create the grid of colors that the player can select.

getFile

Return the file that the class is currently using.

getColorData

This private method is used to retrieve the data from the file that was passed into the class in the constructor. It requires three parameters. The first is the type of node to look for. The next is the type of entry that you are looking for. The last parameter is the attribute that you want to find from the file. This method will then return to you an array that contains each instance that matches the parameters.

getArrayOfCrestColor

This will create an array of crest color combinations that the players are able to select.

XMLTabData

This is a retriever class. It provides a method that will return a TabData class that is filled with the information necessary to fill in the tabs.

XMLTabData Constructor

This will create a new XMLTabData class. It requires as a parameter the path to the location of the tab_data.xml file that will be searched through.

getFile

Returns the file that this class is currently using.

getTabData

This method will return a TabData class that contains the data necessary to fill in the tabs in the crest customizer.

Wii Control and Data server

WiiMote control files

The root folder contains two important files. These files are wiiuse.dll and WiiUseJ.dll. Both of these are used by the administrative console to connect and control the WiiMotes. They must be in the same directory as the HVAJoustAdmin jar file. Without these libraries, the administrative console will be unable to connect to the WiiMotes.

HVAJoustAdmin.jar

This is the executable version of the administrator console. For information about the source files, look at the javadoc.

Flash FLA

Start Scene

The purpose of this scene is twofold. It will play a screensaver movie until a user comes to play the game and then will determine if they are an old or new player. For new players, they will be directed in the Crest Customizer to create their heraldry and give themselves a name. For returning players, their information is already saved and they can have it retrieved. The player will be told what score they need to beat in order to win either through voiceovered text or text displayed on the screen.

Frame 1

This frame contains the opening movie clip, slideshow. This movie clip, which is found in the starting slide show layer, will run continuously when the game is not being played. For a player to start, they need to click on the green button in the lower right side of the screen that is in the button layer.

Frames 2-44

These are transitions frames. Frames 2-29 are used for the fade out of the starting screen and frames 30-44 are used for the scroll lowering phase. The tween for the starting screen is found in layers Button and Starting Slide Show while the tween for the scroll is found in Scroll Tween.

Frame 45

This frame is used to ask the player if they have played the game before. It has a voiceover to assist with those that might not be able to read yet.

Input Layer

The ActionScript on this layer is used to import the LoadOldPlayer class and to start playing the voiceover for this section. The voiceover, 1st line, is started and played until the player hits one of the buttons. The sound file for the voiceover can be found in the Opening Voiceovers folder in the library.

This layer also includes the two buttons that the player can click. If they click on the no button, a new currentPlayer.xml file with a flag specifying that the player is new is created. If they click on yes, we will move to frame 47 to get information used to retrieve their information.

Variables Layer

This contains variables that are useful to have across the opening frames. The variables are

Name of variable	Purpose
Path	Used if the files are not in the default position
playerName	The name of the player that is currently playing. Used for returning players
newPlayerFlag	Flag that determines if the player is new or a returning player

Text

This layer contains the text that is shown on the frame.

Scroll

This layer contains the image of the scroll that is seen in the frame

Background

This layer contains the black background that is seen behind the scroll.

FRAME 47

This frame is used if the player is a returning player. It will ask for their name and search the database for the player's information. It also contains a voiceover, 2nd line, of the text. If the user can't find their name or if they if they hit yes by mistake, they are given the option to register as a new knight.

Input

This layer contains the input boxes and buttons to move forward. The upper two are used for the player to enter their name and have it searched for. When they click on the green arrow next to the name box, the name that was typed in is sent to the LoadOldPlayer method, findOldPlayer. If it returns true, we have a returning player and can load their information into the currentPlayer.xml file. If the method returns false, a message is displayed on the screen stating that the name could not be found.

The lower button is used if they came to this screen by mistake or if they were unable to find their name. This will have the same effect as clicking no on the previous frame.

FRAME 48

This is an unused frame. As there is no in game tutorial, there was no need to ask returning players if they would want to go through it again. If a tutorial is added at a later time, this frame would be the place to ask if returning players would like to go through it again. We recommend that all new players be required to use the tutorial so that they can learn to play the game.

FRAME 49

This frame gives the introduction to the tournament and explains a bit about the scoring. If the user does not want to listen to any of this, they are able to skip it using the green arrow in the lower right corner.

Input

This layer contains the skip button. At any time during the movie, the player is able to click on this button to stop the text and move forward in the game. This will bring the player to frame 50.

Text

This layer contains the scrolling text for this frame. It is contained within the movie clip scroll text if the wording needs to be edited. The voiceover the text is part of the movie clip itself.

FRAME 50

For returning players, this is used to say what score they will need to beat in order to win the tournament again. Currently, the player's last high score is not saved so it will always tell them that they need to beat 14 points.

Text

Depending on the status of the player, different messages will be displayed. For new players, this screen is skipped entirely and they are sent to the end of the scene. For returning players, it will check to see what their saved high score is. If it is 14, it will display a message telling them that they need to beat this score to win. If it is higher than 14, it will tell them that in order to win they must beat or equal their previous score.

FRAMES 51-199

These are used to give the player time to read the information that is displayed on screen.

FRAME 200

The end of the start scene, we now are going to move to the Storyboard scene.

STORYBOARD

This scene contains most of the wrapping story that leads into the joust. It will show the player a few images of what the joust looked like with a voiceover that gives a bit of story. The ending image of the judge's gallery has two different voiceovers. One is in the event that the player is new, the other in the event that the player is old.

SKIP BUTTON LAYER

This layer contains a button that will allow the player to skip the storyboard and go immediately to the next scene in the game. This button will also turn off all sounds that are currently rolling so they are not heard in the next scenes.

JUDGESGALLERY LAYER

This layer contains the image of the judge's galley and its tween. The fade in will start at frame 455 and end at 491 with the image completed shown. The fade out of the image will begin on frame 665 and end on frame 740. The scene will be advanced to the Crest Customizer.

HERALD LAYER

This layer contains the image of the herald that is holding heraldry. It will start to fade in at frame 284 and finish fading in at frame 324. Otherwise this layer is blank.

CITYSCAPE LAYER

This contains the image of the city that the player is going to be jousting in. It will finish fading in at frame 60 and will being to fade out at frame 284.

SOUNDS

This layer controls the sounds and selection of the tournament name. On frame 1, it will open the currentPlayer.xml file which will be used later in determining if they are a new player or not. In addition, the tournamentName.xml file will be opened. This file contains the number code for the name of the tournament. The number code represents the frame number inside of the crest customizer that that image is on.

On frame two, the sound clip for the tournament name will be played. This is accomplished by a switch statement and then attaching the correct sound to the welcome variable

On frame 455, the currentPlayer class is used to determine if the player is new or not. If the player is new, the "Heraldy" voiceover will be played while if they are a veteran it will play the "Veteran" voiceover.

All sound clips that are used in this scene are located within the Storyboard folder in the library. Each of the different names for the tournament is located within the Tournament Names folder.

Crest Customizer

The Crest Customizer is used by new players to create their heraldry and to name themselves. It will get the data for the colors it should use for the shields from the color_data.xml file that is in the data folder. Returning players will have this scene bypassed as they do not need to make heraldry.

Frame 1

Variables Layer

This layer contains variables that are used throughout the entire scene. It will also import the LoadOldPlayer class.

Actions Layer

This layer loads the three files that the crest customizer will need in order to operate. To prevent it from moving forward without having one of these files loaded, it stops at this frame and waits for all the files to load. Each time one of the files finishes loading, the loadApp() function is called. This checks to see if all the files have been loaded.

After all the files have been loaded, it will check to see if the player is new. If they are not new, it will move to frame 56, forcing a move to the next scene. Otherwise it enters into the crest customizer itself.

Frame 2

This is where the player builds their new crest. The colors which are available for the crests are taken from the color_data.xml file. The ordinaries that can be used are located within the Crest Customizer/Charges folder. The emblems that are used are located in the Crest Customizer/Ordinaries folder.

Button Layer

This contains the button that is used to advance to the next part of the crest customization sequence. This can be clicked at any time when the user has finished selecting the heraldry that they want.

Divider Layer

The divider that separates the shield that is being created from the pages movie clip that contains the choices for the shield.

Pages Layer

This layer contains the pages movie clip. This movie clip is where the user is able to select their choices for the various aspects of the heraldry.

Pages Movie Clip

This movie clip has a total of three frames. Each frame represents one if the three different items that can be added to the heraldry. Frame 1 is the design tab, frame 2 is the color tab, and frame 3 is the emblem tab. The text for each tab is set through the text layer. This will use the TabData class to get what each of the description texts for that frame should be.

Each time a different frame is accessed, it will repopulate itself. This is done in the actions layer. The variable columns is current set to 4 to allow for 4 columns of choices. It is recommended that this not change this number. The population is done through nested four loops to create the grid. Each item is created as a new movie clip with an onRelease function that will modify the user's shield.

Tabs Layer

This contains the tabs that are above the Pages movie clip that allow the user to change which screen is shown on the Pages movie clip. Each tab is a movie clip with an onRelease function that changes the frame of the Pages movie clip.

Text area layer

The text area on the lower left part of the screen is contained in this layer. Information about the selected item can be displayed in here by desc_text variable in the crest_desc movie clip.

Shield

The shield is contained within the shield folder and in the Crest layer. The shield folder takes care of all the texturing and imaging for the shield. The charge designs are placed on the shield and a mask is used to ensure that their colors will be properly applied. In the Crest layer, the movie clip with the linkage name crest exists. This is where the emblem is located. Inside of the movie clip, there is another movie clip named charge. Inside of this movie clip, each frame contains a different emblem.

To change the emblem, use the follow command:

```
_root.crest.charge.gotoAndStop(emblemIndex)
```

Background layer

Provides a background color for the left side of the screen.

Shield base layer

Provides a background color for the shield.

FRAME 30

This is where the user is able to input their name for their knight. It also allows them to view their completed heraldry.

PlayerNameInput Layer

This layer has two parts, the button that tells the scene to move forward and the box that allows the player to input their name. When the arrow is pressed, it will send to the server the new data to the server to have it written to the currentPlayer.xml. After this has been completed, we fade out of this and move to the jousting scene.

JOUST

This is the main portion of the game. Here the player is able to test their skills with the horse and lance. The joust runs for a total of six seconds, with frames afterward to make the transitions easier.

GENERAL LAYERS

Landscape

This provides the basic background colors for the scene.

Crowd stands layer

This holds the movie clip that represents the people in the stands. It is by default named stands. To start or stop the stands loop, you can use stands.play() and stands.stop().

Building layer

This is a building in the background to give the player a better feel that they are moving forward. It will slowly enlarge through tweens as the joust rounds plays out. If you look closely at the building, you can see images of the enemy's and player's heraldry on it. This is done in a similar way to the crest customizer.

CONTROLLER LAYER

This layer is responsible for the controlling the incoming wiimote messages and resetting the horse control positions at the end of a match. On frame two, it will hide your mouse pointer so you will only see the lance. …

GENERAL EVENTS LAYER

This layer contains general events that occur during the joust and contains the global variables for the scene.

Frame 2

This frame is used to load the data necessary to create a random enemy knight, color in the player's horse, and put the player's heraldry onto the signpost.

Frame 22

This is the frame responsible for starting each round of the joust. This can be done in one of two ways. The first is through the use of the wiimotes and the other is through the keyboard.

If using the wiimotes, an XML socket is created and an onXML function attached to it. This method, named handleIncoming, will receive the xml message that it received. If the message contains a node named "g", this will cause the joust to begin and remove the XML socket listener.

The second option is through the keyboard. This is accomplished by using keyboard. Here, a new object was created and a function added to it for onKeyDown. This function would then get the code for the key that was pressed and check if it was a space. If it was, it will remove the function listener from itself and die. This new object is then added to a key listener by calling Key.addListener(object).

Frame 122

This is the start of the short freeze so the player can see where they managed to hit. The all movement on the screen will stop. Through the usage of ColorTransform and Transform, the area that is hit is shaded. First a new Transform is created for the area that will be shaded. For example to color the enemy knight's head you would call new Transform(enemy.head). Next This new Transform object has the method colorTransform invoked on it using the created ColorTransform. The following is a complete example:

```
var colorTrans:ColorTransform = new ColorTransform();
colorTrans.redOffset = 200;
enemy.head._alpha = 33;
var trans:Transform = new Transform(enemy.head);
trans.colorTransform = colorTrans;
```

Frame 123

This frame plays the correct sound based on where the opponent hit and shakes the wiimote if necessary.

For the sounds, it attaches all the possible sound files to variables. It will then check the jousting cause to see which one should be played.

For the shaking of the wiimote, u send over the lance socket <1/>. The more that are used the longer the wiimote will shake.

In addition, the lance shatter animation begins here. This is done by replacing the lance move clip with the breaking lance movie clip.

Frame 145

This will undo the previous color transformation and restart the moving objects in the scene.

Frame 197

This frame determines if the joust is finished or if we need to loop again. If it has not played the correct number of rounds, it will store the information about the round and go back to frame two. Otherwise it will append the jousting result data to the currentPlayer.xml file. The data is stored in two arrays. The round results are stored in the roundResults array and the area that was hit is stored in the roundHitLocs array.

HORSE CONTROLLER

Horse control is one aspect of the game used to give it a more realistic feel. The ActionScript needed to create this mechanism is held in the Horse Control Events layer.

Frame 2

This is the setup area for the horse controller. It will reset the horse head to its starting position.

Frame 23

This frame contains the functions that are needed for the horse controller. The bobbin() function is used to prevent the horse from going too in one direction. The moveHorseHead() function is used to physically move the horse's head. The flashLeftArrow and flashRightArrow will cause the respective arrow to flash. Combined with setInterval, this arrow will flash on screen at the correct time.

Horse Control event

There are two instances of horse control, the first on frame 40 and 60 with the other on 75 and 95. During the first of each of those, the direction of the horse is randomly chosen and the current frame number is saved in startFrame. The input for the controller can come from either the wiimotes or the keyboard.

For the wiimote, a XML socket is used. Using the horse_socket variable, the function checkHCWii is called. In this message will be the value from the wiimote on the reins. If the player turns the wiimote the correct direction, the program will recognize this and record the frame that this occurred on in endFrame. The listener is then removed from the socket.

If it is coming from the keyboard, it is treated the same way as starting the joust is except it will be used to control the horse controller.

The part of the horse controller happens on frames 60 and 95. Here the arrows are cleared and any remaining listeners removed. The difference between endFrame and startFrame determines how well the player performed horse control. The HorseControlRating class instance named hRating has its update method called to store the result of this round of horse control.

Lance control

The other aspect of jousting is to correctly lower your lance. This is implemented similar to the horse controller in the GeneralEvents layer, Frame 23. First, the starting frame is saved into the variable initlowerframe. Next the function lance_lowered is set to be checked every 30 milliseconds to see if the lance is lowered. When called, it will check to see if the lance as been lowered to the threshold. If it has it will record the frame that it was lowered in as lowered_frame and find the difference between lowered_frame and initlowerframe. Depending on the delay between the two, this will determine your score for lance control. The lance score is set to the HitCalculator class directly.

Hitting the knight

The calculations for hitting the knight are done through the joust/lance items/lance hit movie clip. In this movie clip, it has a function that will execute upon entering the frame (onEnterFrame function). Using the hitTest call against the various parts of the enemy knight, the program determines where the knight was hit. The function will then update the root HitPoint class with the area that was hit.

Scoring

Scoring is controlled by the HitCalculator class. Currently, hitCalc is its instance in the joust scene. After setting the point that was hit, the lance control score, and the horse control score by using the setters setHitPoint, setLanceScore, and setsetHorseScore you can call the calculate method. This will return a JoustResult to you. By using the getCause method of the JoustResult, it is possible to determine how the player did.

Results

This scene is used to give the player a small movie based on how they did and then to display their results in an authentic scorecard augmented with subtitles to explain how they did.

Result movies

The player will see a movie based upon how they did during the joust. The currentPlayer.xml file will be loaded in frame one in the actions layer. In frame two, a check will be performed to determine if they managed to win or not. This occurs in the CheckForWin layer. If they did manage to win, this layer will output the new tournament name into the tournamentName.xml file and play the winner movie. If they lost, a scroll will be displayed telling them that they lost.

The lose movie will play from the Lose layer. It begins on frame three and continues until frame 365.

The win movie is contained in the Win layer. It begins on frame 375 and ends on frame 620.

In either case, after the movie has finished the player will be brought to the result screen. Here they will be able to view how they did during the joust on an authentic scorecard. This happens in frame 650.

RESULT SCREEN

This screen creates an authentic scorecard for the player to see how they did during their jousting. All of its actions occur in frame 650.

Global Vars Layer

This layer contains variables that are needed across the entire scene. Currently, it imports the CurrentPlayer class and creates the variable player. It will not proceed until the currentPlayer.xml file is loaded and ready for use. The import and wait occur on frame 1.

SetHeraldry Layer

Using currentPlayer and the player variable, it retrieves information about the player and how they did during the joust. The ActionScript first will create their heraldry and place their name over it. Next, it will read in the results from each of the four rounds of jousting. This text is placed in the text fields named rx, where x is the round number.

Output Layer

This layer contains the Dynamic Text boxes that will hold the results of each of the four rounds and the name of the player. The lower four boxes are used to hold the results of the rounds and are named rX, where x refers to the round number of the results that should be put in them. The upper text box is the name of the knight is and named pName.

Run Ticks Layer

The primary usage of this layer is to control the scorecard image itself. Adding the ticks to the base image is accomplished through ActionScript. In the while loop which will run for the number of runs that were done, it will first use the currentPlayer class to get the area that was hit and the result of for that area. Using this information, it will determine where the new 'tick' should be placed. For an example, we will assume that the shield was hit. The tick will go in different places depending on if they unhorsed the opponent or shattered their lance. It will check for these conditions and then duplicate the tick movie clips that are outside of the stage and place it in the correct position.

Grid Layer

This layer contains the static text for labeling the rounds and the line the separates the upper portion of the screen from the lower section.

Parchment Layer

The layer contains the image of the parchment that is used as a background for the text.

Background layer

This layer contains a black background for the areas that are not covered by the parchment layer.

HVA Jousting Simulator Administrative Console

Javadoc Source Code Documentation

Patrick Newell

4/23/2009

One of the more unique events during medieval tournaments was the joust. In this event, mounted knights would charge at their enemies using various weapons, with the intention of dismounting them. In an effort to replicate this event, a virtual jousting game will be produced. Four main areas of research will support this project: historical context of the joust, mechanics of the joust, design concepts, and technical concepts. Background research on the joust will be required to set up the historical context and game mechanics, and design and technical research will be required in order to construct the actual game.

PACKAGE OVERVIEW

HVA Jousting Simulator Administrative Console

Packages	
hva.core	
hva.crest.data	
hva.crest.gui	
hva.gui	
hva.tree	
hva.wii.gui	
hva.wii.net	
hva.wii.remote	
hva.xml.gui	
hva.xml.util	

Package Hierarchy

Hierarchy For All Packages

Package Hierarchies:

hva.core, hva.crest.data, hva.crest.gui, hva.gui, hva.tree, hva.wii.gui, hva.wii.net, hva.wii.remote, hva.xml.gui, hva.xml.util

---

Class Hierarchy

- java.lang.Object
  - java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - java.awt.Container
      - javax.swing.JComponent (implements java.io.Serializable)
        - javax.swing.JPanel (implements javax.accessibility.Accessible)
          - hva.crest.gui.CrestColorCustomizerPanel
          - hva.crest.gui.DataConfigPanel (implements javax.swing.event.TreeSelectionListener)
          - hva.wii.gui.WiiServerPanel
          - hva.xml.gui.XMLDataServerPanel
      - java.awt.Window (implements javax.accessibility.Accessible)
        - java.awt.Frame (implements java.awt.MenuContainer)
          - javax.swing.JFrame (implements javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants)
            - hva.gui.MainFrame

  hva.crest.data.CrestColor
  hva.crest.data.CrestColorTreeGenerator (implements hva.tree.ITreeGenerator)
  hva.core.Main
  java.lang.Thread (implements java.lang.Runnable)
    - hva.wii.remote.WiiHorseListenerThread (implements hva.wii.remote.IWiiListenerThread)
    - hva.wii.remote.WiiLanceListenerThread (implements hva.wii.remote.IWiiListenerThread)
    - hva.wii.remote.WiiRumbleListener
    - hva.xml.util.XMLListener
    - hva.xml.util.XMLListenerThread

  hva.wii.remote.WiiHorseListener (implements wiiusej.wiiusejevents.utils.WiimoteListener)
  hva.wii.net.WiiHorseServer (implements hva.wii.net.IServer)
  hva.wii.remote.WiiLanceListener (implements wiiusej.wiiusejevents.utils.WiimoteListener)
  hva.wii.net.WiiLanceServer (implements hva.wii.net.IServer)
  hva.wii.remote.WiiMouse
  hva.wii.remote.WiiMouseClickListener (implements wiiusej.wiiusejevents.utils.WiimoteListener)

- hva.wii.remote.WiiMouseListener (implements wiiusej.wiiusejevents.utils.WiimoteListener)
- hva.wii.net.XMLDataServer (implements hva.wii.net.IServer)
- hva.xml.util.XMLDocument
    - hva.crest.data.CrestColorXMLDocument
    - hva.xml.util.XMLDataDocument
- hva.wii.net.XMLSocketConnection (implements hva.wii.net.IConnection)

Interface Hierarchy

- hva.wii.net.IConnection
- hva.wii.net.IServer
- hva.tree.ITreeGenerator
- hva.wii.remote.IWiiListenerThread

Enum Hierarchy

- java.lang.Object
    - java.lang.Enum<E> (implements java.lang.Comparable<T>, java.io.Serializable)
        - hva.crest.data.CrestColor.Material

Class Details

HVA.CREST.DATA

Class CrestColor

```
java.lang.Object
  └ hva.crest.data.CrestColor
```

```
public class CrestColor
extends java.lang.Object
```

This is the CrestColor class.

Author:

Patrick Newell

Nested Class Summary

`static class`	`CrestColor.Material` Enumeration of material types for crest color

Constructor Summary

`CrestColor(java.lang.String name, java.lang.String heraldicName, java.lang.String hexRGB, java.lang.String description, CrestColor.Material material)` Constructor for the CrestColor class

Method Summary

`java.lang.String`	`getDescription()`

	Get the description of the color.
`java.lang.String`	`getHeraldicName()` Get the heraldic name of the color.
`java.lang.String`	`getHexRGB()` Get the hexadecimal value of the color.
`CrestColor.Material`	`getMaterial()` Get the material of the color.
`java.lang.String`	`getName()` Get the modern name of the color.
`void`	`setDescription(java.lang.String description)` Set the description of the color.
`void`	`setHeraldicName(java.lang.String heraldicName)` Set the heraldic name of the color.
`void`	`setHexRGB(java.lang.String hexRGB)` Set the hexadecimal value of the color.
`void`	`setMaterial(CrestColor.Material material)` Set the material of the color.
`void`	`setName(java.lang.String name)` Set the modern name of the color.
`java.lang.String`	`toString()`

Methods inherited from class java.lang.Object

```
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait
```

Constructor Detail

CrestColor

```
public CrestColor(java.lang.String name,
                  java.lang.String heraldicName,
                  java.lang.String hexRGB,
                  java.lang.String description,
                  CrestColor.Material material)
```

Constructor for the CrestColor class

Parameters:

`name` - String value of modern name of color

`heraldicName` - String value of heraldic name of color

`hexRGB` - String value of hexadecimal representation of color

`description` - String value of description of color

Method Detail

getName

```
public java.lang.String getName()
```

Get the modern name of the color.

Returns:

String value of modern name

---

setName

```
public void setName(java.lang.String name)
```

Set the modern name of the color.

Parameters:

`name` - String value of modern name

---

getHeraldicName

`public java.lang.String` **`getHeraldicName`**`()`

Get the heraldic name of the color.

Returns:

String value of heraldic name

---

setHeraldicName

`public void` **`setHeraldicName`**`(java.lang.String heraldicName)`

Set the heraldic name of the color.

Parameters:

`heraldicName` - String value of heraldic name

---

getHexRGB

`public java.lang.String` **`getHexRGB`**`()`

Get the hexadecimal value of the color.

Returns:

String value of hex value

---

setHexRGB

`public void` **`setHexRGB`**`(java.lang.String hexRGB)`

Set the hexadecimal value of the color.

Parameters:

`hexRGB` - String value of hex value

---

getDescription

`public java.lang.String` **`getDescription`**`()`

Get the description of the color.

Returns:

String value of description of color

SETDESCRIPTION

`public void` **`setDescription`**`(java.lang.String description)`

Set the description of the color.

Parameters:

`description` - String value of description of color

GETMATERIAL

`public` `CrestColor.Material` **`getMaterial`**`()`

Get the material of the color.

Returns:

Material of color

SETMATERIAL

`public void` **`setMaterial`**`(CrestColor.Material material)`

Set the material of the color.

Parameters:

`material` - Material value of color

TOSTRING

`public java.lang.String` **`toString`**`()`

Overrides:

`toString` in class `java.lang.Object`

HVA.CREST.DATA

ENUM CRESTCOLOR.MATERIAL

```
java.lang.Object
  └ java.lang.Enum<CrestColor.Material>
      └ hva.crest.data.CrestColor.Material
```

All Implemented Interfaces:

java.io.Serializable, java.lang.Comparable<CrestColor.Material>

Enclosing class:

CrestColor

```
public static enum CrestColor.Material
extends java.lang.Enum<CrestColor.Material>
```

Enumeration of material types for crest color

Enum Constant Summary

`COLOR`
`METAL`

Method Summary

`static CrestColor.Material`	`valueOf(java.lang.String name)` Returns the enum constant of this type with the specified name.
`static CrestColor.Material[]`	`values()` Returns an array containing the constants of this enum type, in the order they are declared.

Methods inherited from class java.lang.Enum
`compareTo, equals, getDeclaringClass, hashCode, name, ordinal, toString, valueOf`

Methods inherited from class java.lang.Object
`getClass, notify, notifyAll, wait, wait, wait`

Enum Constant Detail

METAL

`public static final CrestColor.Material` **`METAL`**

COLOR

`public static final CrestColor.Material` **`COLOR`**

Method Detail

VALUES

`public static CrestColor.Material[]` **`values`**`()`

Returns an array containing the constants of this enum type, in the order they are declared. This method may be used to iterate over the constants as follows:

```
for (CrestColor.Material c : CrestColor.Material.values())
    System.out.println(c);
```

Returns:

an array containing the constants of this enum type, in the order they are declared

VALUEOF

`public static CrestColor.Material` **`valueOf`**`(java.lang.String name)`

Returns the enum constant of this type with the specified name. The string must match *exactly* an identifier used to declare an enum constant in this type. (Extraneous whitespace characters are not permitted.)

Parameters:

`name` - the name of the enum constant to be returned.

Returns:

the enum constant with the specified name

Throws:

`java.lang.IllegalArgumentException` - if this enum type has no constant with the specified name

`java.lang.NullPointerException` - if the argument is null

HVA.CREST.GUI

CLASS CRESTCOLORCUSTOMIZERPANEL

```
java.lang.Object
  └ java.awt.Component
      └ java.awt.Container
          └ javax.swing.JComponent
              └ javax.swing.JPanel
                  └ hva.crest.gui.CrestColorCustomizerPanel
```

All Implemented Interfaces:

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible

---

```
public class CrestColorCustomizerPanel
extends javax.swing.JPanel
```

This is the CrestColorCustomizerPanel class. It is responsible for providing a front-end to modifying individual colors and metals.

Author:

Patrick Newell

See Also:

Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class javax.swing.JComponent
`javax.swing.JComponent.AccessibleJComponent`

Nested classes/interfaces inherited from class java.awt.Component
`java.awt.Component.BaselineResizeBehavior`

Field Summary

Fields inherited from class javax.swing.JComponent
`TOOL_TIP_TEXT_KEY, UNDEFINED_CONDITION, WHEN_ANCESTOR_OF_FOCUSED_COMPONENT, WHEN_FOCUSED, WHEN_IN_FOCUSED_WINDOW`

Fields inherited from class java.awt.Component
`BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT`

Fields inherited from interface java.awt.image.ImageObserver
`ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH`

Constructor Summary

`CrestColorCustomizerPanel(CrestColor ccData)` Constructor for the CrestColorCustomizerPanel

Method Summary

`void`	`loadData()` Restores last data saved to CrestColor object.
`void`	`saveData()`

	This writes any modified data in the GUI back to the CrestColor object.
`void`	`updateData(CrestColor ccData)` This updates the data in the CrestColorCustomizer panel with the information from the given CrestColor object.

Methods inherited from class javax.swing.JPanel
`getAccessibleContext, getUI, getUIClassID, setUI, updateUI`

Methods inherited from class javax.swing.JComponent
`addAncestorListener, addNotify, addVetoableChangeListener, computeVisibleRect, contains, createToolTip, disable, enable, firePropertyChange, firePropertyChange, firePropertyChange, getActionForKeyStroke, getActionMap, getAlignmentX, getAlignmentY, getAncestorListeners, getAutoscrolls, getBaseline, getBaselineResizeBehavior, getBorder, getBounds, getClientProperty, getComponentPopupMenu, getConditionForKeyStroke, getDebugGraphicsOptions, getDefaultLocale, getFontMetrics, getGraphics, getHeight, getInheritsPopupMenu, getInputMap, getInputMap, getInputVerifier, getInsets, getInsets, getListeners, getLocation, getMaximumSize, getMinimumSize, getNextFocusableComponent, getPopupLocation, getPreferredSize, getRegisteredKeyStrokes, getRootPane, getSize, getToolTipLocation, getToolTipText, getToolTipText, getTopLevelAncestor, getTransferHandler, getVerifyInputWhenFocusTarget, getVetoableChangeListeners, getVisibleRect, getWidth, getX, getY, grabFocus, isDoubleBuffered, isLightweightComponent, isManagingFocus, isOpaque, isOptimizedDrawingEnabled, isPaintingForPrint, isPaintingTile, isRequestFocusEnabled, isValidateRoot, paint, paintImmediately, paintImmediately, print, printAll, putClientProperty, registerKeyboardAction, registerKeyboardAction, removeAncestorListener, removeNotify, removeVetoableChangeListener, repaint, repaint, requestDefaultFocus, requestFocus, requestFocus, requestFocusInWindow, resetKeyboardActions, reshape, revalidate, scrollRectToVisible, setActionMap, setAlignmentX, setAlignmentY, setAutoscrolls, setBackground, setBorder, setComponentPopupMenu, setDebugGraphicsOptions, setDefaultLocale, setDoubleBuffered, setEnabled, setFocusTraversalKeys, setFont, setForeground, setInheritsPopupMenu, setInputMap,`

setInputVerifier, setMaximumSize, setMinimumSize, setNextFocusableComponent, setOpaque, setPreferredSize, setRequestFocusEnabled, setToolTipText, setTransferHandler, setVerifyInputWhenFocusTarget, setVisible, unregisterKeyboardAction, update

Methods inherited from class java.awt.Container
add, add, add, add, add, addContainerListener, addPropertyChangeListener, addPropertyChangeListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getComponentZOrder, getContainerListeners, getFocusTraversalKeys, getFocusTraversalPolicy, getLayout, getMousePosition, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusCycleRoot, isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paintComponents, preferredSize, printComponents, remove, remove, removeAll, removeContainerListener, setComponentZOrder, setFocusCycleRoot, setFocusTraversalPolicy, setFocusTraversalPolicyProvider, setLayout, transferFocusBackward, transferFocusDownCycle, validate

Methods inherited from class java.awt.Component
action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, contains, createImage, createImage, createVolatileImage, createVolatileImage, dispatchEvent, enable, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusCycleRootAncestor, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getForeground, getGraphicsConfiguration, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputContext, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocale, getLocation, getLocationOnScreen, getMouseListeners, getMouseMotionListeners, getMousePosition, getMouseWheelListeners, getName,

`getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners, getSize, getToolkit, getTreeLock, gotFocus, handleEvent, hasFocus, hide, imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isMaximumSizeSet, isMinimumSizeSet, isPreferredSizeSet, isShowing, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, postEvent, prepareImage, prepareImage, remove, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, resize, resize, setBounds, setBounds, setComponentOrientation, setCursor, setDropTarget, setFocusable, setFocusTraversalKeysEnabled, setIgnoreRepaint, setLocale, setLocation, setLocation, setName, setSize, setSize, show, show, size, toString, transferFocus, transferFocusUpCycle`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

CrestColorCustomizerPanel

`public` **`CrestColorCustomizerPanel`**`(CrestColor ccData)`

Constructor for the CrestColorCustomizerPanel

Parameters:

`ccData` - CrestColor object containing color or metal data to modify

Method Detail

updateData

`public void` **`updateData`**`(CrestColor ccData)`

This updates the data in the CrestColorCustomizer panel with the information from the given CrestColor object.

Parameters:

`ccData` - CrestColor object to

SAVEDATA

`public void` **`saveData`**`()`

This writes any modified data in the GUI back to the CrestColor object.

LOADDATA

`public void` **`loadData`**`()`

Restores last data saved to CrestColor object.

HVA.CREST.DATA

CLASS CRESTCOLORTREEGENERATOR

```
java.lang.Object
  └ hva.crest.data.CrestColorTreeGenerator
```

All Implemented Interfaces:

ITreeGenerator

```
public class CrestColorTreeGenerator
extends java.lang.Object
implements ITreeGenerator
```

This is the CrestColorTreeGenerator class.

Author:

Patrick Newell

Field Summary

static java.lang.String	ROOT_NAME CONSTANT String value of root node

Constructor Summary

CrestColorTreeGenerator(java.lang.String docPath) Constructor for CrestColorTreeGenerator

Method Summary

CrestColorXMLDocument	getCrestColorXMLDocument() Get the CrestColorXMLDocument

`javax.swing.tree.DefaultMutableTreeNode`	`getRootNode()` Returns the root node.

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Field Detail

ROOT_NAME

`public static final java.lang.String` **`ROOT_NAME`**

CONSTANT String value of root node

See Also:

Constant Field Values

Constructor Detail

CRESTCOLORTREEGENERATOR

`public` **`CrestColorTreeGenerator`**`(java.lang.String docPath)`

Constructor for CrestColorTreeGenerator

Parameters:

`docPath` - String value of path and filename of XML file containing data

Method Detail

GETCRESTCOLORXMLDOCUMENT

`public CrestColorXMLDocument` **`getCrestColorXMLDocument`**`()`

Get the CrestColorXMLDocument

Returns:

CrestColorXMLDocument object

GETROOTNODE

`public javax.swing.tree.DefaultMutableTreeNode` **`getRootNode`**`()`

Returns the root node.

Specified by:

`getRootNode` in interface `ITreeGenerator`

Returns:

DefaultMutableTreeNode root node of tree

HVA.CREST.DATA

CLASS CRESTCOLORXMLDOCUMENT

```
java.lang.Object
  └ hva.xml.util.XMLDocument
      └ hva.crest.data.CrestColorXMLDocument
```

```
public class CrestColorXMLDocument
extends XMLDocument
```

This is the CrestColorXMLDocument class.

Author:

Patrick Newell

Constructor Summary
`CrestColorXMLDocument(java.lang.String xmlFile)` Constructor for the ColorDataXMLDocument class.

Method Summary	
`java.util.ArrayList<CrestColor>`	`getColors()` Get the list of colors.
`java.util.ArrayList<CrestColor>`	`getMetals()` Get the list of metals.
`void`	`writeChanges()` Writes current data in this class to the XML file that it read its data from originally.

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

CrestColorXMLDocument

`public` **`CrestColorXMLDocument`**`(java.lang.String xmlFile)`

Constructor for the ColorDataXMLDocument class.

Parameters:

`xmlFile` - String value of XML filename

Method Detail

writeChanges

`public void` **`writeChanges`**`()`

Writes current data in this class to the XML file that it read its data from originally. It will make a backup of the original file first, validate the newly written file, and then delete the old XML file.

---

getColors

`public java.util.ArrayList<CrestColor>` **`getColors`**`()`

Get the list of colors.

Returns:

ArrayList of CrestColor data objects of colors

---

getMetals

`public java.util.ArrayList<CrestColor>` **`getMetals`**`()`

Get the list of metals.

Returns:

ArrayList of CrestColor data objects of metals

HVA.CREST.GUI

CLASS DATACONFIGPANEL

```
java.lang.Object
  └ java.awt.Component
      └ java.awt.Container
          └ javax.swing.JComponent
              └ javax.swing.JPanel
                  └ hva.crest.gui.DataConfigPanel
```

All Implemented Interfaces:

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.util.EventListener, javax.accessibility.Accessible, javax.swing.event.TreeSelectionListener

---

```
public class DataConfigPanel
extends javax.swing.JPanel
implements javax.swing.event.TreeSelectionListener
```

This is the DataConfigPanel class. It is responsible for providing a front-end to modify XML data for the Virtual Jousting Simulator.

Author:

Patrick Newell

See Also:

Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class javax.swing.JComponent

`javax.swing.JComponent.AccessibleJComponent`

Nested classes/interfaces inherited from class java.awt.Component

`java.awt.Component.BaselineResizeBehavior`

Field Summary

Fields inherited from class javax.swing.JComponent
`TOOL_TIP_TEXT_KEY, UNDEFINED_CONDITION, WHEN_ANCESTOR_OF_FOCUSED_COMPONENT, WHEN_FOCUSED, WHEN_IN_FOCUSED_WINDOW`

Fields inherited from class java.awt.Component
`BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT`

Fields inherited from interface java.awt.image.ImageObserver
`ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH`

Constructor Summary

`DataConfigPanel()` Constructor for this class.

Method Summary

`void`	`valueChanged(javax.swing.event.TreeSelectionEvent arg0)`

	Required override for implementing TreeSelectionListener.

Methods inherited from class javax.swing.JPanel
getAccessibleContext, getUI, getUIClassID, setUI, updateUI

Methods inherited from class javax.swing.JComponent
addAncestorListener, addNotify, addVetoableChangeListener, computeVisibleRect, contains, createToolTip, disable, enable, firePropertyChange, firePropertyChange, firePropertyChange, getActionForKeyStroke, getActionMap, getAlignmentX, getAlignmentY, getAncestorListeners, getAutoscrolls, getBaseline, getBaselineResizeBehavior, getBorder, getBounds, getClientProperty, getComponentPopupMenu, getConditionForKeyStroke, getDebugGraphicsOptions, getDefaultLocale, getFontMetrics, getGraphics, getHeight, getInheritsPopupMenu, getInputMap, getInputMap, getInputVerifier, getInsets, getInsets, getListeners, getLocation, getMaximumSize, getMinimumSize, getNextFocusableComponent, getPopupLocation, getPreferredSize, getRegisteredKeyStrokes, getRootPane, getSize, getToolTipLocation, getToolTipText, getToolTipText, getTopLevelAncestor, getTransferHandler, getVerifyInputWhenFocusTarget, getVetoableChangeListeners, getVisibleRect, getWidth, getX, getY, grabFocus, isDoubleBuffered, isLightweightComponent, isManagingFocus, isOpaque, isOptimizedDrawingEnabled, isPaintingForPrint, isPaintingTile, isRequestFocusEnabled, isValidateRoot, paint, paintImmediately, paintImmediately, print, printAll, putClientProperty, registerKeyboardAction, registerKeyboardAction, removeAncestorListener, removeNotify, removeVetoableChangeListener, repaint, repaint, requestDefaultFocus, requestFocus, requestFocus, requestFocusInWindow, resetKeyboardActions, reshape, revalidate, scrollRectToVisible, setActionMap, setAlignmentX, setAlignmentY, setAutoscrolls, setBackground, setBorder, setComponentPopupMenu, setDebugGraphicsOptions, setDefaultLocale, setDoubleBuffered, setEnabled, setFocusTraversalKeys, setFont, setForeground, setInheritsPopupMenu, setInputMap, setInputVerifier, setMaximumSize, setMinimumSize, setNextFocusableComponent, setOpaque, setPreferredSize, setRequestFocusEnabled, setToolTipText, setTransferHandler, setVerifyInputWhenFocusTarget, setVisible, unregisterKeyboardAction, update

Methods inherited from class java.awt.Container
`add, add, add, add, add, addContainerListener, addPropertyChangeListener, addPropertyChangeListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getComponentZOrder, getContainerListeners, getFocusTraversalKeys, getFocusTraversalPolicy, getLayout, getMousePosition, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusCycleRoot, isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paintComponents, preferredSize, printComponents, remove, remove, removeAll, removeContainerListener, setComponentZOrder, setFocusCycleRoot, setFocusTraversalPolicy, setFocusTraversalPolicyProvider, setLayout, transferFocusBackward, transferFocusDownCycle, validate`

Methods inherited from class java.awt.Component
`action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, contains, createImage, createImage, createVolatileImage, createVolatileImage, dispatchEvent, enable, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusCycleRootAncestor, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getForeground, getGraphicsConfiguration, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputContext, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocale, getLocation, getLocationOnScreen, getMouseListeners, getMouseMotionListeners, getMousePosition, getMouseWheelListeners, getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners, getSize, getToolkit, getTreeLock, gotFocus, handleEvent, hasFocus, hide, imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isMaximumSizeSet, isMinimumSizeSet, isPreferredSizeSet, isShowing,`

`isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, postEvent, prepareImage, prepareImage, remove, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, resize, resize, setBounds, setBounds, setComponentOrientation, setCursor, setDropTarget, setFocusable, setFocusTraversalKeysEnabled, setIgnoreRepaint, setLocale, setLocation, setLocation, setName, setSize, setSize, show, show, size, toString, transferFocus, transferFocusUpCycle`

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

DataConfigPanel

`public` **`DataConfigPanel`**`()`

Constructor for this class.

Method Detail

valueChanged

`public void` **`valueChanged`**`(javax.swing.event.TreeSelectionEvent arg0)`

Required override for implementing TreeSelectionListener.

Specified by:

`valueChanged` in interface `javax.swing.event.TreeSelectionListener`

hva.wii.net

Interface IConnection

All Known Implementing Classes:

XMLSocketConnection

```
public interface IConnection
```

This is the IConnection interface.

Author:

Patrick Newell

Method Summary

`java.net.Socket`	`getClientSocket()` Returns the ClientSocket of the connection.
`java.io.BufferedReader`	`getInStream()` Returns the BufferedReader of the input stream for this connection.
`java.net.ServerSocket`	`getServerSocket()` Returns the ServerSocket of the connection.
`void`	`send(java.lang.String msg)` Sends a given message across the connection.

Method Detail

getServerSocket

```
java.net.ServerSocket getServerSocket()
```

Returns the ServerSocket of the connection.

Returns:

ServerSocket of connection

GETCLIENTSOCKET

`java.net.Socket` **`getClientSocket`**`()`

Returns the ClientSocket of the connection.

Returns:

Socket of the connection.

SEND

`void` **`send`**`(java.lang.String msg)`

Sends a given message across the connection.

Parameters:

`msg` - String value of message

GETINSTREAM

`java.io.BufferedReader` **`getInStream`**`()`

Returns the BufferedReader of the input stream for this connection.

Returns:

BufferedReader of input stream

INTERFACE ISERVER

All Known Implementing Classes:

WiiHorseServer, WiiLanceServer, XMLDataServer

```
public interface IServer
```

This is the IServer interface.

Author:

Patrick Newell

Method Summary	
`java.util.ArrayList<IConnection>`	`getConnections()` Returns a list of connections to the IServer
`int`	`getPort()` Returns the port of the IServer.
`boolean`	`isStarted()` Returns status of the IServer.
`void`	`restart()` Restarts the IServer.
`void`	`start()` Starts the IServer.
`void`	`stop()` Stops the IServer.

Method Detail

START

`void` **`start`**`()`

Starts the IServer.

STOP

`void` **`stop`**`()`

Stops the IServer.

RESTART

`void` **`restart`**`()`

Restarts the IServer.

GETPORT

`int` **`getPort`**`()`

Returns the port of the IServer.

Returns:

int value of port

ISSTARTED

`boolean` **`isStarted`**`()`

Returns status of the IServer.

Returns:

true if IServer is started, false otherwise

GETCONNECTIONS

`java.util.ArrayList<IConnection>` **`getConnections`**`()`

Returns a list of connections to the IServer

Returns:

ArrayList of IConnections connected to the IServer

HVA.TREE

INTERFACE ITREEGENERATOR

All Known Implementing Classes:

CrestColorTreeGenerator

```
public interface ITreeGenerator
```

This is the ITreeGenerator interface.

Author:

Patrick Newell

Method Summary

`javax.swing.tree.DefaultMutableTreeNode`	`getRootNode()` Returns the root node of the tree.

Method Detail

GETROOTNODE

```
javax.swing.tree.DefaultMutableTreeNode getRootNode()
```

Returns the root node of the tree.

Returns:

DefaultMutableTreeNode root node

HVA.WII.REMOTE

INTERFACE IWIILISTENERTHREAD

All Known Implementing Classes:

WiiHorseListenerThread, WiiLanceListenerThread

`public interface IWiiListenerThread`

This is the IWiiListenerThread interface. It is a template for the Thread that handles the listening behavior for a Wiimote.

Author:

Patrick Newell

Method Summary

void	`removeListeners()` Removes listeners.
void	`run()` Runs the Thread.
void	`startListening()` Starts listening.
void	`stopListening()` Stops listening.

Method Detail

RUN

`void run()`

Runs the Thread.

STOPLISTENING

`void stopListening()`

Stops listening.

STARTLISTENING

`void startListening()`

Starts listening.

REMOVELISTENERS

`void removeListeners()`

Removes listeners.

hva.core

Class Main

```
java.lang.Object
  └ hva.core.Main
```

```
public class Main
extends java.lang.Object
```

This is the Main class for this application.

Author:

Patrick Newell

Constructor Summary

`Main()`

Method Summary

`static void`	`main(java.lang.String[] args)`

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

Main

```
public Main()
```

Method Detail

MAIN

```
public static void main(java.lang.String[] args)
```

Parameters:

`args` -

HVA.GUI

CLASS MAINFRAME

```
java.lang.Object
  └ java.awt.Component
      └ java.awt.Container
          └ java.awt.Window
              └ java.awt.Frame
                  └ javax.swing.JFrame
                      └ hva.gui.MainFrame
```

All Implemented Interfaces:

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants

---

```
public class MainFrame
extends javax.swing.JFrame
```

This is the MainFrame class

Author:

Patrick Newell

See Also:

Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class java.awt.Component
`java.awt.Component.BaselineResizeBehavior`

Field Summary

Fields inherited from class javax.swing.JFrame
`EXIT_ON_CLOSE`

Fields inherited from class java.awt.Frame
`CROSSHAIR_CURSOR, DEFAULT_CURSOR, E_RESIZE_CURSOR, HAND_CURSOR, ICONIFIED, MAXIMIZED_BOTH, MAXIMIZED_HORIZ, MAXIMIZED_VERT, MOVE_CURSOR, N_RESIZE_CURSOR, NE_RESIZE_CURSOR, NORMAL, NW_RESIZE_CURSOR, S_RESIZE_CURSOR, SE_RESIZE_CURSOR, SW_RESIZE_CURSOR, TEXT_CURSOR, W_RESIZE_CURSOR, WAIT_CURSOR`

Fields inherited from class java.awt.Component
`BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT`

Fields inherited from interface javax.swing.WindowConstants
`DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, HIDE_ON_CLOSE`

Fields inherited from interface java.awt.image.ImageObserver
`ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH`

Constructor Summary

`MainFrame()` Constructor for MainFrame class.

Method Summary

Methods inherited from class javax.swing.JFrame
`getAccessibleContext, getContentPane, getDefaultCloseOperation, getGlassPane, getGraphics, getJMenuBar, getLayeredPane, getRootPane, getTransferHandler, isDefaultLookAndFeelDecorated, remove, repaint, setContentPane, setDefaultCloseOperation, setDefaultLookAndFeelDecorated, setGlassPane, setIconImage, setJMenuBar, setLayeredPane, setLayout, setTransferHandler, update`

Methods inherited from class java.awt.Frame
`addNotify, getCursorType, getExtendedState, getFrames, getIconImage, getMaximizedBounds, getMenuBar, getState, getTitle, isResizable, isUndecorated, remove, removeNotify, setCursor, setExtendedState, setMaximizedBounds, setMenuBar, setResizable, setState, setTitle, setUndecorated`

Methods inherited from class java.awt.Window
`addPropertyChangeListener, addPropertyChangeListener, addWindowFocusListener, addWindowListener, addWindowStateListener, applyResourceBundle, applyResourceBundle, createBufferStrategy, createBufferStrategy, dispose, getBufferStrategy, getFocusableWindowState, getFocusCycleRootAncestor, getFocusOwner, getFocusTraversalKeys, getGraphicsConfiguration, getIconImages, getInputContext, getListeners, getLocale, getModalExclusionType, getMostRecentFocusOwner, getOwnedWindows, getOwner, getOwnerlessWindows, getToolkit, getWarningString, getWindowFocusListeners, getWindowListeners, getWindows, getWindowStateListeners, hide, isActive, isAlwaysOnTop, isAlwaysOnTopSupported, isFocusableWindow, isFocusCycleRoot, isFocused, isLocationByPlatform, isShowing, pack, postEvent, removeWindowFocusListener, removeWindowListener, removeWindowStateListener, reshape, setAlwaysOnTop, setBounds, setBounds, setCursor, setFocusableWindowState, setFocusCycleRoot, setIconImages, setLocationByPlatform, setLocationRelativeTo, setMinimumSize,`

setModalExclusionType, setSize, setSize, setVisible, show, toBack, toFront

Methods inherited from class java.awt.Container
add, add, add, add, add, addContainerListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getComponentZOrder, getContainerListeners, getFocusTraversalPolicy, getInsets, getLayout, getMaximumSize, getMinimumSize, getMousePosition, getPreferredSize, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paint, paintComponents, preferredSize, print, printComponents, remove, removeAll, removeContainerListener, setComponentZOrder, setFocusTraversalKeys, setFocusTraversalPolicy, setFocusTraversalPolicyProvider, setFont, transferFocusBackward, transferFocusDownCycle, validate

Methods inherited from class java.awt.Component
action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, contains, contains, createImage, createImage, createVolatileImage, createVolatileImage, disable, dispatchEvent, enable, enable, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBaseline, getBaselineResizeBehavior, getBounds, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getFontMetrics, getForeground, getHeight, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocation, getLocation, getLocationOnScreen, getMouseListeners, getMouseMotionListeners, getMousePosition, getMouseWheelListeners, getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners, getSize, getSize, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent,

`hasFocus, imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isDoubleBuffered, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isMaximumSizeSet, isMinimumSizeSet, isOpaque, isPreferredSizeSet, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, prepareImage, prepareImage, printAll, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, requestFocus, requestFocusInWindow, resize, resize, setBackground, setComponentOrientation, setDropTarget, setEnabled, setFocusable, setFocusTraversalKeysEnabled, setForeground, setIgnoreRepaint, setLocale, setLocation, setLocation, setMaximumSize, setName, setPreferredSize, show, size, toString, transferFocus, transferFocusUpCycle`

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Methods inherited from interface java.awt.MenuContainer

`getFont, postEvent`

Constructor Detail

MAINFRAME

`public` **`MainFrame`**`()`

Constructor for MainFrame class.

HVA.WII.REMOTE

CLASS WIIHORSELISTENER

```
java.lang.Object
  └ hva.wii.remote.WiiHorseListener
```

All Implemented Interfaces:

java.util.EventListener, wiiusej.wiiusejevents.utils.WiimoteListener

---

```
public class WiiHorseListener
extends java.lang.Object
implements wiiusej.wiiusejevents.utils.WiimoteListener
```

This is the WiiHorselistener which implements the WiimoteListener interface.

Author:

Patrick Newell

Constructor Summary

`WiiHorseListener(XMLSocketConnection xmlSocket)`   Constructor for the WiiHorseListener.

Method Summary

void	`onButtonsEvent(wiiusej.wiiusejevents.physicalevents.WiimoteButtonsEvent arg0)`
void	`onClassicControllerInsertedEvent(wiiusej.wiiusejevents.wiiuseapievents.ClassicControllerInsertedEvent arg0)`
void	`onClassicControllerRemovedEvent(wiiusej.wiiusejevents.wiiuseapievents.ClassicControllerRemovedEvent arg0)`

void	onDisconnectionEvent(wiiusej.wiiusejevents.wiiuseapievents.DisconnectionEvent arg0)
void	onExpansionEvent(wiiusej.wiiusejevents.physicalevents.ExpansionEvent arg0)
void	onGuitarHeroInsertedEvent(wiiusej.wiiusejevents.wiiuseapievents.GuitarHeroInsertedEvent arg0)
void	onGuitarHeroRemovedEvent(wiiusej.wiiusejevents.wiiuseapievents.GuitarHeroRemovedEvent arg0)
void	onIrEvent(wiiusej.wiiusejevents.physicalevents.IREvent arg0)
void	onMotionSensingEvent(wiiusej.wiiusejevents.physicalevents.MotionSensingEvent arg0)
void	onNunchukInsertedEvent(wiiusej.wiiusejevents.wiiuseapievents.NunchukInsertedEvent arg0)
void	onNunchukRemovedEvent(wiiusej.wiiusejevents.wiiuseapievents.NunchukRemovedEvent arg0)
void	onStatusEvent(wiiusej.wiiusejevents.wiiuseapievents.StatusEvent arg0)

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

WiiHorseListener

`public **WiiHorseListener**(XMLSocketConnection xmlSocket)`

Constructor for the WiiHorseListener.

Parameters:

`xmlSocket` - XMLSocketConnection object to communicate with Flash

Method Detail

onButtonsEvent

```
public void
onButtonsEvent(wiiusej.wiiusejevents.physicalevents.WiimoteButtonsEvent arg0)
```

Specified by:

`onButtonsEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

---

onClassicControllerInsertedEvent

```
public void
onClassicControllerInsertedEvent(wiiusej.wiiusejevents.wiiuseapievents.ClassicContr
ollerInsertedEvent arg0)
```

Specified by:

`onClassicControllerInsertedEvent` in

interface `wiiusej.wiiusejevents.utils.WiimoteListener`

---

onClassicControllerRemovedEvent

```
public void
onClassicControllerRemovedEvent(wiiusej.wiiusejevents.wiiuseapievents.ClassicContro
llerRemovedEvent arg0)
```

Specified by:

`onClassicControllerRemovedEvent` in

interface `wiiusej.wiiusejevents.utils.WiimoteListener`

---

onDisconnectionEvent

`public void` **`onDisconnectionEvent`**`(wiiusej.wiiusejevents.wiiuseapievents.DisconnectionEvent arg0)`

Specified by:

`onDisconnectionEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

---

onExpansionEvent

`public void` **`onExpansionEvent`**`(wiiusej.wiiusejevents.physicalevents.ExpansionEvent arg0)`

Specified by:

`onExpansionEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

---

onGuitarHeroInsertedEvent

`public void` **`onGuitarHeroInsertedEvent`**`(wiiusej.wiiusejevents.wiiuseapievents.GuitarHeroInsertedEvent arg0)`

Specified by:

`onGuitarHeroInsertedEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

---

onGuitarHeroRemovedEvent

`public void` **`onGuitarHeroRemovedEvent`**`(wiiusej.wiiusejevents.wiiuseapievents.GuitarHeroRemovedEvent arg0)`

Specified by:

`onGuitarHeroRemovedEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

---

onIrEvent

`public void` **`onIrEvent`**`(wiiusej.wiiusejevents.physicalevents.IREvent arg0)`

Specified by:

`onIrEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

---

onMotionSensingEvent

`public void`
`onMotionSensingEvent``(wiiusej.wiiusejevents.physicalevents.MotionSensingEvent arg0)`

Specified by:

`onMotionSensingEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

---

onNunchukInsertedEvent

`public void`
`onNunchukInsertedEvent``(wiiusej.wiiusejevents.wiiuseapievents.NunchukInsertedEvent a`
`rg0)`

Specified by:

`onNunchukInsertedEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

---

onNunchukRemovedEvent

`public void`
`onNunchukRemovedEvent``(wiiusej.wiiusejevents.wiiuseapievents.NunchukRemovedEvent arg`
`0)`

Specified by:

`onNunchukRemovedEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

---

onStatusEvent

`public void` **`onStatusEvent`**`(wiiusej.wiiusejevents.wiiuseapievents.StatusEvent arg0)`

Specified by:

`onStatusEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

hva.wii.remote

Class WiiHorseListenerThread

```
java.lang.Object
  └java.lang.Thread
      └hva.wii.remote.WiiHorseListenerThread
```

All Implemented Interfaces:

IWiiListenerThread, java.lang.Runnable

```
public class WiiHorseListenerThread
extends java.lang.Thread
implements IWiiListenerThread
```

This is the WiiHorseListenerThread. It is responsible for the listening behavior of the horse controller.

Author:

Patrick Newell

Nested Class Summary

Nested classes/interfaces inherited from class java.lang.Thread
`java.lang.Thread.State, java.lang.Thread.UncaughtExceptionHandler`

Field Summary

Fields inherited from class java.lang.Thread
`MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY`

Constructor Summary
`WiiHorseListenerThread(java.net.ServerSocket ss, wiiusej.Wiimote[] wiimotes)`   Constructor for the WiiHorseListener

Method Summary	
`void`	`removeListeners()` Removes listeners.
`void`	`run()` Runs the Thread.
`void`	`startListening()` Starts listening.
`void`	`stopListening()` Stops listening.

Methods inherited from class java.lang.Thread
`activeCount, checkAccess, countStackFrames, currentThread, destroy, dumpStack, enumerate, getAllStackTraces, getContextClassLoader, getDefaultUncaughtExceptionHandler, getId, getName, getPriority, getStackTrace, getState, getThreadGroup, getUncaughtExceptionHandler, holdsLock, interrupt,`

`interrupted, isAlive, isDaemon, isInterrupted, join, join, join, resume, setContextClassLoader, setDaemon, setDefaultUncaughtExceptionHandler, setName, setPriority, setUncaughtExceptionHandler, sleep, sleep, start, stop, stop, suspend, toString, yield`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

WiiHorseListenerThread

```
public WiiHorseListenerThread(java.net.ServerSocket ss,
                              wiiusej.Wiimote[] wiimotes)
```

Constructor for the WiiHorseListener

Parameters:

`ss` - ServerSocket for server connection

`wiimotes` - array of connected Wiimotes

Method Detail

removeListeners

```
public void removeListeners()
```

Description copied from interface: `IWiiListenerThread`

Removes listeners.

Specified by:

removeListeners in interface IWiiListenerThread

RUN

`public void run()`

Description copied from interface: IWiiListenerThread

Runs the Thread.

Specified by:

`run` in interface `IWiiListenerThread`

Specified by:

`run` in interface `java.lang.Runnable`

Overrides:

`run` in class `java.lang.Thread`

STARTLISTENING

`public void startListening()`

Description copied from interface: IWiiListenerThread

Starts listening.

Specified by:

`startListening` in interface `IWiiListenerThread`

STOPLISTENING

`public void stopListening()`

Description copied from interface: IWiiListenerThread

Stops listening.

Specified by:

`stopListening` in interface `IWiiListenerThread`

HVA.WII.NET

CLASS WIIHORSESERVER

```
java.lang.Object
  └hva.wii.net.WiiHorseServer
```

All Implemented Interfaces:

IServer

```
public class WiiHorseServer
extends java.lang.Object
implements IServer
```

This is the WiiHorseServer class, which implements the IServer interface. It is responsible for handling communications between the Wiimote controlling the horse and Flash.

Author:

Patrick Newell

Constructor Summary
`WiiHorseServer(wiiusej.Wiimote[] wiimotes)`

Method Summary	
`java.util.ArrayList<IConnection>`	`getConnections()` Returns a list of connections to the IServer
`int`	`getPort()`

	Returns the port of the IServer.
`boolean`	`isStarted()` Returns status of the IServer.
`void`	`killConnections()`
`void`	`restart()` Restarts the IServer.
`void`	`start()` Starts the IServer.
`void`	`stop()` Stops the IServer.

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

WIIHORSESERVER

`public WiiHorseServer(wiiusej.Wiimote[] wiimotes)`

Method Detail

GETCONNECTIONS

`public java.util.ArrayList<IConnection> getConnections()`

Description copied from interface: `IServer`

Returns a list of connections to the IServer

Specified by:

`getConnections` in interface `IServer`

Returns:

ArrayList of IConnections connected to the IServer

GETPORT

`public int getPort()`

Description copied from interface: `IServer`

Returns the port of the IServer.

Specified by:

`getPort` in interface `IServer`

Returns:

int value of port

ISSTARTED

`public boolean isStarted()`

Description copied from interface: `IServer`

Returns status of the IServer.

Specified by:

`isStarted` in interface `IServer`

Returns:

true if IServer is started, false otherwise

RESTART
`public void **restart**()`

Description copied from interface: `IServer`

Restarts the IServer.

Specified by:

`restart` in interface `IServer`

START
`public void **start**()`

Description copied from interface: `IServer`

Starts the IServer.

Specified by:

`start` in interface `IServer`

KILLCONNECTIONS
`public void **killConnections**()`

STOP
`public void **stop**()`

Description copied from interface: `IServer`

Stops the IServer.

Specified by:

`stop` in interface `IServer`

HVA.WII.REMOTE

CLASS WIILANCELISTENER

```
java.lang.Object
  └hva.wii.remote.WiiLanceListener
```

All Implemented Interfaces:

java.util.EventListener, wiiusej.wiiusejevents.utils.WiimoteListener

---

```
public class WiiLanceListener
extends java.lang.Object
implements wiiusej.wiiusejevents.utils.WiimoteListener
```

This is the HVAWiiListener class.

Author:

Patrick Newell

Constructor Summary
WiiLanceListener(XMLSocketConnection xmlSocket)

Method Summary	
void	onButtonsEvent(wiiusej.wiiusejevents.physicalevents.WiimoteButtonsEvent arg0)
void	onClassicControllerInsertedEvent(wiiusej.wiiusejevents.wiiuseapievents.ClassicControllerInsertedEvent arg0)

void	onClassicControllerRemovedEvent(wiiusej.wiiusejevents.wiiuseapievents.ClassicControllerRemovedEvent arg0)
void	onDisconnectionEvent(wiiusej.wiiusejevents.wiiuseapievents.DisconnectionEvent arg0)
void	onExpansionEvent(wiiusej.wiiusejevents.physicalevents.ExpansionEvent arg0)
void	onGuitarHeroInsertedEvent(wiiusej.wiiusejevents.wiiuseapievents.GuitarHeroInsertedEvent arg0)
void	onGuitarHeroRemovedEvent(wiiusej.wiiusejevents.wiiuseapievents.GuitarHeroRemovedEvent arg0)
void	onIrEvent(wiiusej.wiiusejevents.physicalevents.IREvent arg0)
void	onMotionSensingEvent(wiiusej.wiiusejevents.physicalevents.MotionSensingEvent arg0)
void	onNunchukInsertedEvent(wiiusej.wiiusejevents.wiiuseapievents.NunchukInsertedEvent arg0)
void	onNunchukRemovedEvent(wiiusej.wiiusejevents.wiiuseapievents.NunchukRemovedEvent arg0)
void	onStatusEvent(wiiusej.wiiusejevents.wiiuseapievents.StatusEvent arg0)

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

WIILANCELISTENER

`public` **`WiiLanceListener`**`(XMLSocketConnection xmlSocket)`

Method Detail

ONBUTTONSEVENT

`public void`
`onButtonsEvent``(wiiusej.wiiusejevents.physicalevents.WiimoteButtonsEvent arg0)`

Specified by:

`onButtonsEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

ONCLASSICCONTROLLERINSERTEDEVENT

`public void`
`onClassicControllerInsertedEvent``(wiiusej.wiiusejevents.wiiuseapievents.ClassicContr`
`ollerInsertedEvent arg0)`

Specified by:

`onClassicControllerInsertedEvent` in

interface `wiiusej.wiiusejevents.utils.WiimoteListener`

ONCLASSICCONTROLLERREMOVEDEVENT

`public void`
`onClassicControllerRemovedEvent``(wiiusej.wiiusejevents.wiiuseapievents.ClassicContro`
`llerRemovedEvent arg0)`

Specified by:

`onClassicControllerRemovedEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

ONDISCONNECTIONEVENT

```
public void
onDisconnectionEvent(wiiusej.wiiusejevents.wiiuseapievents.DisconnectionEvent arg0)
```

Specified by:

`onDisconnectionEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

ONEXPANSIONEVENT

```
public void
onExpansionEvent(wiiusej.wiiusejevents.physicalevents.ExpansionEvent arg0)
```

Specified by:

`onExpansionEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

ONGUITARHEROINSERTEDEVENT

```
public void
onGuitarHeroInsertedEvent(wiiusej.wiiusejevents.wiiuseapievents.GuitarHeroInsertedE
vent arg0)
```

Specified by:

`onGuitarHeroInsertedEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

ONGUITARHEROREMOVEDEVENT

```
public void
onGuitarHeroRemovedEvent(wiiusej.wiiusejevents.wiiuseapievents.GuitarHeroRemovedEve
nt arg0)
```

Specified by:

`onGuitarHeroRemovedEvent` in

interface `wiiusej.wiiusejevents.utils.WiimoteListener`

ONIREVENT

```
public void onIrEvent(wiiusej.wiiusejevents.physicalevents.IREvent arg0)
```

Specified by:

`onIrEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

ONMOTIONSENSINGEVENT

```
public void
onMotionSensingEvent(wiiusej.wiiusejevents.physicalevents.MotionSensingEvent arg0)
```

Specified by:

`onMotionSensingEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

ONNUNCHUKINSERTEDEVENT

```
public void
onNunchukInsertedEvent(wiiusej.wiiusejevents.wiiuseapievents.NunchukInsertedEvent a
rg0)
```

Specified by:

`onNunchukInsertedEvent` in

interface `wiiusej.wiiusejevents.utils.WiimoteListener`

ONNUNCHUKREMOVEDEVENT

```
public void
onNunchukRemovedEvent(wiiusej.wiiusejevents.wiiuseapievents.NunchukRemovedEvent arg
0)
```

Specified by:

`onNunchukRemovedEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

onStatusEvent

`public void` **`onStatusEvent`**`(wiiusej.wiiusejevents.wiiuseapievents.StatusEvent arg0)`

Specified by:

`onStatusEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

Class WiiLanceListenerThread

```
java.lang.Object
  └java.lang.Thread
      └hva.wii.remote.WiiLanceListenerThread
```

All Implemented Interfaces:

IWiiListenerThread, java.lang.Runnable

```
public class WiiLanceListenerThread
extends java.lang.Thread
implements IWiiListenerThread
```

This is the HVAWiiListenerThread class.

Author:

Patrick Newell

Nested Class Summary

Nested classes/interfaces inherited from class java.lang.Thread
`java.lang.Thread.State, java.lang.Thread.UncaughtExceptionHandler`

Field Summary

Fields inherited from class java.lang.Thread
`MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY`

Constructor Summary

`WiiLanceListenerThread(java.net.ServerSocket ss, wiiusej.Wiimote[] wiimotes)` Constructor for the HVAWiiListenerThread.

Method Summary

`void`	`removeListeners()` Removes all listeners from the server.
`void`	`run()` Starts the main running loop of the server.
`void`	`startListening()` Sets the stop flag to false in the main loop.
`void`	`stopListening()` Sets the stop flag to true in the main loop.

Methods inherited from class java.lang.Thread
`activeCount, checkAccess, countStackFrames, currentThread, destroy, dumpStack, enumerate, getAllStackTraces, getContextClassLoader, getDefaultUncaughtExceptionHandler, getId, getName, getPriority, getStackTrace, getState, getThreadGroup, getUncaughtExceptionHandler, holdsLock, interrupt,`

`interrupted, isAlive, isDaemon, isInterrupted, join, join, join, resume, setContextClassLoader, setDaemon, setDefaultUncaughtExceptionHandler, setName, setPriority, setUncaughtExceptionHandler, sleep, sleep, start, stop, stop, suspend, toString, yield`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

WIILANCELISTENERTHREAD

```
public WiiLanceListenerThread(java.net.ServerSocket ss,
                              wiiusej.Wiimote[] wiimotes)
```

Constructor for the HVAWiiListenerThread.

Parameters:

`ss` - ServerSocket object for server

Method Detail

RUN

```
public void run()
```

Starts the main running loop of the server.

Specified by:

`run` in interface `IWiiListenerThread`

Specified by:

`run` in interface `java.lang.Runnable`

Overrides:

`run` in class `java.lang.Thread`

STOPLISTENING
`public void` **`stopListening`**`()`

Sets the stop flag to true in the main loop.

Specified by:

`stopListening` in interface `IWiiListenerThread`

STARTLISTENING
`public void` **`startListening`**`()`

Sets the stop flag to false in the main loop.

Specified by:

`startListening` in interface `IWiiListenerThread`

REMOVELISTENERS
`public void` **`removeListeners`**`()`

Removes all listeners from the server.

Specified by:

`removeListeners` in interface `IWiiListenerThread`

hva.wii.net

Class WiiLanceServer

```
java.lang.Object
  └hva.wii.net.WiiLanceServer
```

All Implemented Interfaces:

IServer

```
public class WiiLanceServer
extends java.lang.Object
implements IServer
```

This is the WiiFlashServer class.

Author:

Patrick Newell

Constructor Summary
`WiiLanceServer(int port, wiiusej.Wiimote[] wiimotes)` Constructor for WiiFlashServer.
`WiiLanceServer(wiiusej.Wiimote[] wiimotes)` Constructor for WiiFlashServer.

Method Summary	
`java.util.ArrayList<IConnection>`	`getConnections()` Returns the ArrayList of all IConnections that are connected to the WiiFlashServer.

`int`	`getPort()` Returns the port that the WiiFlashServer is listening on.
`boolean`	`isStarted()` Returns whether or not the WiiFlashServer is started.
`void`	`killConnections()` Severs all connections to the WiiFlashServer.
`void`	`restart()` This method restarts the WiiFlashServer.
`void`	`start()` Starts the WiiFlashServer.
`void`	`stop()` Stops the WiiFlashServer.

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

WIILANCESERVER

```
public WiiLanceServer(wiiusej.Wiimote[] wiimotes)
```

Constructor for WiiFlashServer. Will listen by default on port 4444.

WIILANCESERVER

```
public WiiLanceServer(int port,
                      wiiusej.Wiimote[] wiimotes)
```

Constructor for WiiFlashServer. Will listen on specified port.

Parameters:

`port` - int value of port to listen on

Method Detail

GETCONNECTIONS

```
public java.util.ArrayList<IConnection> getConnections()
```

Returns the ArrayList of all IConnections that are connected to the WiiFlashServer.

Specified by:

`getConnections` in interface `IServer`

Returns:

ArrayList of IConnections

ISSTARTED

```
public boolean isStarted()
```

Returns whether or not the WiiFlashServer is started.

Specified by:

`isStarted` in interface `IServer`

Returns:

true if started, false otherwise

START

`public void **start**()`

Starts the WiiFlashServer.

Specified by:

`start` in interface `IServer`

STOP

`public void **stop**()`

Stops the WiiFlashServer.

Specified by:

`stop` in interface `IServer`

RESTART

`public void **restart**()`

This method restarts the WiiFlashServer.

Specified by:

`restart` in interface `IServer`

KILLCONNECTIONS

`public void **killConnections**()`

Severs all connections to the WiiFlashServer.

GETPORT

`public int **getPort**()`

Returns the port that the WiiFlashServer is listening on.

Specified by:

`getPort` in interface `IServer`

Returns:

int value of port

HVA.WII.REMOTE

Class WiiMouse

```
java.lang.Object
  └hva.wii.remote.WiiMouse
```

```
public class WiiMouse
extends java.lang.Object
```

This is the WiiMouse class. It is responsible for handling mouse control with a Wiimote.

Author:

Patrick Newell

Constructor Summary

`WiiMouse(wiiusej.Wiimote[] wiimotes)` Constructor for the WiiMouse object.

Method Summary

`wiiusej.Wiimote[]`	`getWiimotes()` Gets the current Wiimote array.
`void`	`setWiimotes(wiiusej.Wiimote[] wiimotes)` Sets the current Wiimote array.

Methods inherited from class java.lang.Object

```
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
```

Constructor Detail

WIIMOUSE

`public WiiMouse(wiiusej.Wiimote[] wiimotes)`

Constructor for the WiiMouse object. Will start sending mouse movements once the class is instantiated.

Parameters:

`wiimotes` -

Method Detail

SETWIIMOTES

`public void setWiimotes(wiiusej.Wiimote[] wiimotes)`

Sets the current Wiimote array.

Parameters:

`wiimotes` - array of Wiimote objects

GETWIIMOTES

`public wiiusej.Wiimote[] getWiimotes()`

Gets the current Wiimote array.

Returns:

array of Wiimote objects

HVA.WII.REMOTE

Class WiiMouseClickListener

```
java.lang.Object
  └hva.wii.remote.WiiMouseClickListener
```

All Implemented Interfaces:

java.util.EventListener, wiiusej.wiiusejevents.utils.WiimoteListener

---

```
public class WiiMouseClickListener
extends java.lang.Object
implements wiiusej.wiiusejevents.utils.WiimoteListener
```

This is the WiiMouseClickListener. It listens for Wiimote events and converts them to mouse clicks.

Author:

Patrick Newell

Constructor Summary
`WiiMouseClickListener()` Constructor for the WiiMouseClickListener.

Method Summary	
void	`onButtonsEvent(wiiusej.wiiusejevents.physicalevents.WiimoteButtonsEvent arg0)`
void	`onClassicControllerInsertedEvent(wiiusej.wiiusejevents.wiiuseapievents.ClassicControllerInsertedEvent arg0)`

void	onClassicControllerRemovedEvent(wiiusej.wiiusejevents.wiiuseapievents.ClassicControllerRemovedEvent arg0)
void	onDisconnectionEvent(wiiusej.wiiusejevents.wiiuseapievents.DisconnectionEvent arg0)
void	onExpansionEvent(wiiusej.wiiusejevents.physicalevents.ExpansionEvent arg0)
void	onGuitarHeroInsertedEvent(wiiusej.wiiusejevents.wiiuseapievents.GuitarHeroInsertedEvent arg0)
void	onGuitarHeroRemovedEvent(wiiusej.wiiusejevents.wiiuseapievents.GuitarHeroRemovedEvent arg0)
void	onIrEvent(wiiusej.wiiusejevents.physicalevents.IREvent arg0)
void	onMotionSensingEvent(wiiusej.wiiusejevents.physicalevents.MotionSensingEvent arg0)
void	onNunchukInsertedEvent(wiiusej.wiiusejevents.wiiuseapievents.NunchukInsertedEvent arg0)
void	onNunchukRemovedEvent(wiiusej.wiiusejevents.wiiuseapievents.NunchukRemovedEvent arg0)

void	onStatusEvent(wiiusej.wiiusejevents.wiiuseapievents.StatusEvent arg0)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

WiiMouseClickListener

`public WiiMouseClickListener()`

Constructor for the WiiMouseClickListener.

Method Detail

onButtonsEvent

`public void`
onButtonsEvent`(wiiusej.wiiusejevents.physicalevents.WiimoteButtonsEvent arg0)`

Specified by:

`onButtonsEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

---

onClassicControllerInsertedEvent

`public void`
onClassicControllerInsertedEvent`(wiiusej.wiiusejevents.wiiuseapievents.ClassicControllerInsertedEvent arg0)`

Specified by:

`onClassicControllerInsertedEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

---

onClassicControllerRemovedEvent

```
public void
onClassicControllerRemovedEvent(wiiusej.wiiusejevents.wiiuseapievents.ClassicContro
llerRemovedEvent arg0)
```

Specified by:

`onClassicControllerRemovedEvent` in

interface `wiiusej.wiiusejevents.utils.WiimoteListener`

---

onDisconnectionEvent

```
public void
onDisconnectionEvent(wiiusej.wiiusejevents.wiiuseapievents.DisconnectionEvent arg0)
```

Specified by:

`onDisconnectionEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

---

onExpansionEvent

```
public void
onExpansionEvent(wiiusej.wiiusejevents.physicalevents.ExpansionEvent arg0)
```

Specified by:

`onExpansionEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

---

onGuitarHeroInsertedEvent

```
public void
onGuitarHeroInsertedEvent(wiiusej.wiiusejevents.wiiuseapievents.GuitarHeroInsertedE
vent arg0)
```

Specified by:

`onGuitarHeroInsertedEvent` in

interface `wiiusej.wiiusejevents.utils.WiimoteListener`

---

onGuitarHeroRemovedEvent

```
public void
onGuitarHeroRemovedEvent(wiiusej.wiiusejevents.wiiuseapievents.GuitarHeroRemovedEve
nt arg0)
```

Specified by:

`onGuitarHeroRemovedEvent` in

interface `wiiusej.wiiusejevents.utils.WiimoteListener`

---

onIrEvent

```
public void onIrEvent(wiiusej.wiiusejevents.physicalevents.IREvent arg0)
```

Specified by:

`onIrEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

---

onMotionSensingEvent

```
public void
onMotionSensingEvent(wiiusej.wiiusejevents.physicalevents.MotionSensingEvent arg0)
```

Specified by:

`onMotionSensingEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

---

onNunchukInsertedEvent

```
public void
onNunchukInsertedEvent(wiiusej.wiiusejevents.wiiuseapievents.NunchukInsertedEvent a
rg0)
```

Specified by:

`onNunchukInsertedEvent` in

interface `wiiusej.wiiusejevents.utils.WiimoteListener`

---

onNunchukRemovedEvent

```
public void
onNunchukRemovedEvent(wiiusej.wiiusejevents.wiiuseapievents.NunchukRemovedEvent arg
0)
```

Specified by:

`onNunchukRemovedEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

ONSTATUSEVENT

```
public void onStatusEvent(wiiusej.wiiusejevents.wiiuseapievents.StatusEvent arg0)
```

Specified by:

`onStatusEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

hva.wii.remote

Class WiiMouseListener

```
java.lang.Object
  └hva.wii.remote.WiiMouseListener
```

All Implemented Interfaces:

java.util.EventListener, wiiusej.wiiusejevents.utils.WiimoteListener

---

```
public class WiiMouseListener
extends java.lang.Object
implements wiiusej.wiiusejevents.utils.WiimoteListener
```

This is the WiiMouseListener class. It is responsible for handling Wiimote movement events and converting them to mouse movements.

Author:

Patrick Newell

Constructor Summary
`WiiMouseListener()`   Constructor for the WiiMouseListener class.

Method Summary	
void	`onButtonsEvent(wiiusej.wiiusejevents.physicalevents.WiimoteButtonsEvent arg0)`
void	`onClassicControllerInsertedEvent(wiiusej.wiiusejevents.wiiuseapievents.ClassicControllerInsertedEvent arg0)`

void	onClassicControllerRemovedEvent(wiiusej.wiiusejevents.wiiuseapievents.ClassicControllerRemovedEvent arg0)
void	onDisconnectionEvent(wiiusej.wiiusejevents.wiiuseapievents.DisconnectionEvent arg0)
void	onExpansionEvent(wiiusej.wiiusejevents.physicalevents.ExpansionEvent arg0)
void	onGuitarHeroInsertedEvent(wiiusej.wiiusejevents.wiiuseapievents.GuitarHeroInsertedEvent arg0)
void	onGuitarHeroRemovedEvent(wiiusej.wiiusejevents.wiiuseapievents.GuitarHeroRemovedEvent arg0)
void	onIrEvent(wiiusej.wiiusejevents.physicalevents.IREvent arg0)
void	onMotionSensingEvent(wiiusej.wiiusejevents.physicalevents.MotionSensingEvent arg0)
void	onNunchukInsertedEvent(wiiusej.wiiusejevents.wiiuseapievents.NunchukInsertedEvent arg0)
void	onNunchukRemovedEvent(wiiusej.wiiusejevents.wiiuseapievents.NunchukRemovedEvent arg0)

vo id	onStatusEvent(wiiusej.wiiusejevents.wiiuseapievents.StatusEvent arg0)

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

WIIMOUSELISTENER

`public` **`WiiMouseListener`**`()`

Constructor for the WiiMouseListener class.

Method Detail

ONBUTTONSEVENT

`public void`
`onButtonsEvent``(wiiusej.wiiusejevents.physicalevents.WiimoteButtonsEvent arg0)`

Specified by:

`onButtonsEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

ONCLASSICCONTROLLERINSERTEDEVENT

`public void`
`onClassicControllerInsertedEvent``(wiiusej.wiiusejevents.wiiuseapievents.ClassicContr ollerInsertedEvent arg0)`

Specified by:

`onClassicControllerInsertedEvent` in

interface `wiiusej.wiiusejevents.utils.WiimoteListener`

ONCLASSICCONTROLLERREMOVEDEVENT

public void
onClassicControllerRemovedEvent(wiiusej.wiiusejevents.wiiuseapievents.ClassicControllerRemovedEvent arg0)

Specified by:

onClassicControllerRemovedEvent in interface wiiusej.wiiusejevents.utils.WiimoteListener

ONDISCONNECTIONEVENT

public void
onDisconnectionEvent(wiiusej.wiiusejevents.wiiuseapievents.DisconnectionEvent arg0)

Specified by:

onDisconnectionEvent in interface wiiusej.wiiusejevents.utils.WiimoteListener

ONEXPANSIONEVENT

public void
onExpansionEvent(wiiusej.wiiusejevents.physicalevents.ExpansionEvent arg0)

Specified by:

onExpansionEvent in interface wiiusej.wiiusejevents.utils.WiimoteListener

ONGUITARHEROINSERTEDEVENT

public void
onGuitarHeroInsertedEvent(wiiusej.wiiusejevents.wiiuseapievents.GuitarHeroInsertedEvent arg0)

Specified by:

onGuitarHeroInsertedEvent in interface wiiusej.wiiusejevents.utils.WiimoteListener

onGuitarHeroRemovedEvent

`public void` **`onGuitarHeroRemovedEvent`**`(wiiusej.wiiusejevents.wiiuseapievents.GuitarHeroRemovedEvent arg0)`

Specified by:

`onGuitarHeroRemovedEvent` in

interface `wiiusej.wiiusejevents.utils.WiimoteListener`

---

onIrEvent

`public void` **`onIrEvent`**`(wiiusej.wiiusejevents.physicalevents.IREvent arg0)`

Specified by:

`onIrEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

---

onMotionSensingEvent

`public void` **`onMotionSensingEvent`**`(wiiusej.wiiusejevents.physicalevents.MotionSensingEvent arg0)`

Specified by:

`onMotionSensingEvent` in interface `wiiusej.wiiusejevents.utils.WiimoteListener`

---

onNunchukInsertedEvent

`public void` **`onNunchukInsertedEvent`**`(wiiusej.wiiusejevents.wiiuseapievents.NunchukInsertedEvent arg0)`

Specified by:

`onNunchukInsertedEvent` in

interface `wiiusej.wiiusejevents.utils.WiimoteListener`

---

onNunchukRemovedEvent

public void
onNunchukRemovedEvent(wiiusej.wiiusejevents.wiiuseapievents.NunchukRemovedEvent arg
0)

Specified by:

onNunchukRemovedEvent in interface wiiusej.wiiusejevents.utils.WiimoteListener

---

ONSTATUSEVENT

public void **onStatusEvent**(wiiusej.wiiusejevents.wiiuseapievents.StatusEvent arg0)

Specified by:

onStatusEvent in interface wiiusej.wiiusejevents.utils.WiimoteListener

HVA.WII.REMOTE

CLASS WIIRUMBLELISTENER

```
java.lang.Object
  └java.lang.Thread
      └hva.wii.remote.WiiRumbleListener
```

All Implemented Interfaces:

java.lang.Runnable

```
public class WiiRumbleListener
extends java.lang.Thread
```

This is the WiiRumbleListener class. This is responsible for listening for lance hit events in Flash and making the Wiimote rumble.

Author:

Patrick Newell

Nested Class Summary

Nested classes/interfaces inherited from class java.lang.Thread
`java.lang.Thread.State, java.lang.Thread.UncaughtExceptionHandler`

Field Summary

Fields inherited from class java.lang.Thread
`MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY`

Constructor Summary
`WiiRumbleListener(XMLSocketConnection xsc, wiiusej.Wiimote wl)` Constructor for the WiiRumbleListener.

Method Summary	
`void`	`rumble(long time)` Makes the Wiimote controlling the lance rumble for the given amount of time.
`void`	`run()` Runs the Thread.

Methods inherited from class java.lang.Thread
`activeCount, checkAccess, countStackFrames, currentThread, destroy, dumpStack, enumerate, getAllStackTraces, getContextClassLoader, getDefaultUncaughtExceptionHandler, getId, getName, getPriority, getStackTrace, getState, getThreadGroup, getUncaughtExceptionHandler, holdsLock, interrupt, interrupted, isAlive, isDaemon, isInterrupted, join, join, join, resume, setContextClassLoader, setDaemon, setDefaultUncaughtExceptionHandler, setName, setPriority, setUncaughtExceptionHandler, sleep, sleep, start, stop, stop, suspend, toString, yield`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

WIIRUMBLELISTENER

```
public WiiRumbleListener(XMLSocketConnection xsc,
                         wiiusej.Wiimote wl)
```

Constructor for the WiiRumbleListener.

Parameters:

`xsc` - XMLSocketConnection to communicate with Flash

`wl` - Wiimote controlling the lance

Method Detail

RUN

```
public void run()
```

Runs the Thread.

Specified by:

`run` in interface `java.lang.Runnable`

Overrides:

`run` in class `java.lang.Thread`

RUMBLE

`public void` **`rumble`**`(long time)`

Makes the Wiimote controlling the lance rumble for the given amount of time.

Parameters:

`time` - long value of amount of time to rumble in milliseconds

HVA.WII.GUI

CLASS WIISERVERPANEL

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └hva.wii.gui.WiiServerPanel
```

All Implemented Interfaces:

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible

---

```
public class WiiServerPanel
extends javax.swing.JPanel
```

This is the WiiServerPanel class.

Author:

Patrick Newell

See Also:

Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class javax.swing.JComponent
javax.swing.JComponent.AccessibleJComponent

Nested classes/interfaces inherited from class java.awt.Component
`java.awt.Component.BaselineResizeBehavior`

Field Summary

Fields inherited from class javax.swing.JComponent
`TOOL_TIP_TEXT_KEY, UNDEFINED_CONDITION, WHEN_ANCESTOR_OF_FOCUSED_COMPONENT, WHEN_FOCUSED, WHEN_IN_FOCUSED_WINDOW`

Fields inherited from class java.awt.Component
`BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT`

Fields inherited from interface java.awt.image.ImageObserver
`ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH`

Constructor Summary

`WiiServerPanel()`   Constructor for WiiServerPanel

Method Summary

`WiiMouse`	`getMouseHandler()` Gets the current mouse handler.
`void`	`setMouseHandler(WiiMouse mouseHandler)` Sets the current mouse handler.

Methods inherited from class javax.swing.JPanel
`getAccessibleContext, getUI, getUIClassID, setUI, updateUI`

Methods inherited from class javax.swing.JComponent
`addAncestorListener, addNotify, addVetoableChangeListener, computeVisibleRect, contains, createToolTip, disable, enable, firePropertyChange, firePropertyChange, firePropertyChange, getActionForKeyStroke, getActionMap, getAlignmentX, getAlignmentY, getAncestorListeners, getAutoscrolls, getBaseline, getBaselineResizeBehavior, getBorder, getBounds, getClientProperty, getComponentPopupMenu, getConditionForKeyStroke, getDebugGraphicsOptions, getDefaultLocale, getFontMetrics, getGraphics, getHeight, getInheritsPopupMenu, getInputMap, getInputMap, getInputVerifier, getInsets, getInsets, getListeners, getLocation, getMaximumSize, getMinimumSize, getNextFocusableComponent, getPopupLocation, getPreferredSize, getRegisteredKeyStrokes, getRootPane, getSize, getToolTipLocation, getToolTipText, getToolTipText, getTopLevelAncestor, getTransferHandler, getVerifyInputWhenFocusTarget, getVetoableChangeListeners, getVisibleRect, getWidth, getX, getY, grabFocus, isDoubleBuffered, isLightweightComponent, isManagingFocus, isOpaque, isOptimizedDrawingEnabled, isPaintingForPrint, isPaintingTile, isRequestFocusEnabled, isValidateRoot, paint, paintImmediately, paintImmediately, print, printAll, putClientProperty, registerKeyboardAction, registerKeyboardAction, removeAncestorListener, removeNotify, removeVetoableChangeListener, repaint, repaint, requestDefaultFocus, requestFocus, requestFocus, requestFocusInWindow, resetKeyboardActions, reshape,`

revalidate, scrollRectToVisible, setActionMap, setAlignmentX, setAlignmentY, setAutoscrolls, setBackground, setBorder, setComponentPopupMenu, setDebugGraphicsOptions, setDefaultLocale, setDoubleBuffered, setEnabled, setFocusTraversalKeys, setFont, setForeground, setInheritsPopupMenu, setInputMap, setInputVerifier, setMaximumSize, setMinimumSize, setNextFocusableComponent, setOpaque, setPreferredSize, setRequestFocusEnabled, setToolTipText, setTransferHandler, setVerifyInputWhenFocusTarget, setVisible, unregisterKeyboardAction, update

Methods inherited from class java.awt.Container
add, add, add, add, add, addContainerListener, addPropertyChangeListener, addPropertyChangeListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getComponentZOrder, getContainerListeners, getFocusTraversalKeys, getFocusTraversalPolicy, getLayout, getMousePosition, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusCycleRoot, isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paintComponents, preferredSize, printComponents, remove, remove, removeAll, removeContainerListener, setComponentZOrder, setFocusCycleRoot, setFocusTraversalPolicy, setFocusTraversalPolicyProvider, setLayout, transferFocusBackward, transferFocusDownCycle, validate

Methods inherited from class java.awt.Component
action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, contains, createImage, createImage, createVolatileImage, createVolatileImage, dispatchEvent, enable, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusCycleRootAncestor, getFocusListeners,

```
getFocusTraversalKeysEnabled, getFont, getForeground, getGraphicsConfiguration,
getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint,
getInputContext, getInputMethodListeners, getInputMethodRequests, getKeyListeners,
getLocale, getLocation, getLocationOnScreen, getMouseListeners,
getMouseMotionListeners, getMousePosition, getMouseWheelListeners, getName,
getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners,
getSize, getToolkit, getTreeLock, gotFocus, handleEvent, hasFocus, hide,
imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isEnabled,
isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet,
isLightweight, isMaximumSizeSet, isMinimumSizeSet, isPreferredSizeSet, isShowing,
isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus,
mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus,
paintAll, postEvent, prepareImage, prepareImage, remove, removeComponentListener,
removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener,
removeInputMethodListener, removeKeyListener, removeMouseListener,
removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener,
removePropertyChangeListener, repaint, repaint, repaint, resize, resize, setBounds,
setBounds, setComponentOrientation, setCursor, setDropTarget, setFocusable,
setFocusTraversalKeysEnabled, setIgnoreRepaint, setLocale, setLocation,
setLocation, setName, setSize, setSize, show, show, size, toString, transferFocus,
transferFocusUpCycle
```

Methods inherited from class java.lang.Object

```
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait
```

Constructor Detail

WIISERVERPANEL

`public **WiiServerPanel**()`

Constructor for WiiServerPanel

Method Detail

SETMOUSEHANDLER

```
public void setMouseHandler(WiiMouse mouseHandler)
```

Sets the current mouse handler.

Parameters:

`mouseHandler` - WiiMouse object to control mouse

GETMOUSEHANDLER

```
public WiiMouse getMouseHandler()
```

Gets the current mouse handler.

Returns:

WiiMouse object controlling the mouse

HVA.XML.UTIL

CLASS XMLDATADOCUMENT

```
java.lang.Object
  └hva.xml.util.XMLDocument
      └hva.xml.util.XMLDataDocument
```

```
public class XMLDataDocument
extends XMLDocument
```

This is the XMLDataDocument class. It is responsible for handling the data associated with and XML file.

Author:

Patrick Newell

Constructor Summary

`XMLDataDocument(java.lang.String xmlString)` Constructor for the ColorDataXMLDocument class.

Method Summary

`org.w3c.dom.Document`	`getDoc()` Gets the current document object.
`void`	`writeChanges(java.lang.String path)` Writes current data in this class to the XML file that it read its data from originally.

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

XMLDataDocument

`public` **`XMLDataDocument`**`(java.lang.String xmlString)`

Constructor for the ColorDataXMLDocument class.

Parameters:

`xmlFile` - String value of XML filename

Method Detail

WRITECHANGES

```
public void writeChanges(java.lang.String path)
```

Writes current data in this class to the XML file that it read its data from originally. It will make a backup of the original file first, validate the newly written file, and then delete the old XML file.

GETDOC

```
public org.w3c.dom.Document getDoc()
```

Gets the current document object.

Returns:

Document object for XML file

hva.wii.net

Class XMLDataServer

```
java.lang.Object
  └hva.wii.net.XMLDataServer
```

All Implemented Interfaces:

IServer

```
public class XMLDataServer
extends java.lang.Object
implements IServer
```

This is the XMLDataServer class, which implements the IServer interface. It is responsible for writing data to XML files from Flash.

Author:

Patrick Newell

Constructor Summary
`XMLDataServer()` Constructor forXMLDataServer.
`XMLDataServer(int port)` Constructor for WiiFlashServer.

Method Summary	
`java.util.ArrayList<IConnection>`	`getConnections()` Returns the ArrayList of all IConnections that are

	connected to the WiiFlashServer.
`int`	`getPort()` Returns the port that the WiiFlashServer is listening on.
`boolean`	`isStarted()` Returns whether or not the WiiFlashServer is started.
`void`	`killConnections()` Severs all connections to the WiiFlashServer.
`void`	`restart()` This method restarts the WiiFlashServer.
`void`	`start()` Starts the WiiFlashServer.
`void`	`stop()` Stops the WiiFlashServer.

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

XMLDataServer

`public` **`XMLDataServer`**`()`

Constructor forXMLDataServer. Will listen by default on port 4446.

XMLDataServer

`public XMLDataServer(int port)`

Constructor for WiiFlashServer. Will listen on specified port.

Parameters:

`port` - int value of port to listen on

Method Detail

getConnections

`public java.util.ArrayList<IConnection> getConnections()`

Returns the ArrayList of all IConnections that are connected to the WiiFlashServer.

Specified by:

`getConnections` in interface `IServer`

Returns:

ArrayList of IConnections

---

isStarted

`public boolean isStarted()`

Returns whether or not the WiiFlashServer is started.

Specified by:

`isStarted` in interface `IServer`

Returns:

true if started, false otherwise

START
`public void` **`start`**`()`

Starts the WiiFlashServer.

Specified by:

`start` in interface `IServer`

STOP
`public void` **`stop`**`()`

Stops the WiiFlashServer.

Specified by:

`stop` in interface `IServer`

RESTART
`public void` **`restart`**`()`

This method restarts the WiiFlashServer.

Specified by:

`restart` in interface `IServer`

KILLCONNECTIONS
`public void` **`killConnections`**`()`

Severs all connections to the WiiFlashServer.

GETPORT
`public int` **`getPort`**`()`

Returns the port that the WiiFlashServer is listening on.

Specified by:

`getPort` in interface `IServer`

Returns:

int value of port

CLASS XMLDATASERVERPANEL

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └javax.swing.JComponent
              └javax.swing.JPanel
                  └hva.xml.gui.XMLDataServerPanel
```

All Implemented Interfaces:

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible

---

```
public class XMLDataServerPanel
extends javax.swing.JPanel
```

This is the XMLDataServerPanel class. It is responsible for providing a front-end for controlling the XML Data Server.

Author:

Patrick Newell

See Also:

Serialized Form

Nested Class Summary

Nested classes/interfaces inherited from class javax.swing.JComponent
`javax.swing.JComponent.AccessibleJComponent`

Nested classes/interfaces inherited from class java.awt.Component
`java.awt.Component.BaselineResizeBehavior`

Field Summary

Fields inherited from class javax.swing.JComponent
`TOOL_TIP_TEXT_KEY, UNDEFINED_CONDITION, WHEN_ANCESTOR_OF_FOCUSED_COMPONENT, WHEN_FOCUSED, WHEN_IN_FOCUSED_WINDOW`

Fields inherited from class java.awt.Component
`BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT`

Fields inherited from interface java.awt.image.ImageObserver
`ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH`

Constructor Summary

`XMLDataServerPanel()` Constructor for the XMLDataServerPanel.

Method Summary

Methods inherited from class javax.swing.JPanel
getAccessibleContext, getUI, getUIClassID, setUI, updateUI

Methods inherited from class javax.swing.JComponent
addAncestorListener, addNotify, addVetoableChangeListener, computeVisibleRect, contains, createToolTip, disable, enable, firePropertyChange, firePropertyChange, firePropertyChange, getActionForKeyStroke, getActionMap, getAlignmentX, getAlignmentY, getAncestorListeners, getAutoscrolls, getBaseline, getBaselineResizeBehavior, getBorder, getBounds, getClientProperty, getComponentPopupMenu, getConditionForKeyStroke, getDebugGraphicsOptions, getDefaultLocale, getFontMetrics, getGraphics, getHeight, getInheritsPopupMenu, getInputMap, getInputMap, getInputVerifier, getInsets, getInsets, getListeners, getLocation, getMaximumSize, getMinimumSize, getNextFocusableComponent, getPopupLocation, getPreferredSize, getRegisteredKeyStrokes, getRootPane, getSize, getToolTipLocation, getToolTipText, getToolTipText, getTopLevelAncestor, getTransferHandler, getVerifyInputWhenFocusTarget, getVetoableChangeListeners, getVisibleRect, getWidth, getX, getY, grabFocus, isDoubleBuffered, isLightweightComponent, isManagingFocus, isOpaque, isOptimizedDrawingEnabled, isPaintingForPrint, isPaintingTile, isRequestFocusEnabled, isValidateRoot, paint, paintImmediately, paintImmediately, print, printAll, putClientProperty, registerKeyboardAction, registerKeyboardAction, removeAncestorListener, removeNotify, removeVetoableChangeListener, repaint, repaint, requestDefaultFocus, requestFocus, requestFocus, requestFocusInWindow, resetKeyboardActions, reshape, revalidate, scrollRectToVisible, setActionMap, setAlignmentX, setAlignmentY, setAutoscrolls, setBackground, setBorder, setComponentPopupMenu, setDebugGraphicsOptions, setDefaultLocale, setDoubleBuffered, setEnabled, setFocusTraversalKeys, setFont, setForeground, setInheritsPopupMenu, setInputMap,

setInputVerifier, setMaximumSize, setMinimumSize, setNextFocusableComponent, setOpaque, setPreferredSize, setRequestFocusEnabled, setToolTipText, setTransferHandler, setVerifyInputWhenFocusTarget, setVisible, unregisterKeyboardAction, update

Methods inherited from class java.awt.Container
add, add, add, add, add, addContainerListener, addPropertyChangeListener, addPropertyChangeListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getComponentZOrder, getContainerListeners, getFocusTraversalKeys, getFocusTraversalPolicy, getLayout, getMousePosition, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusCycleRoot, isFocusTraversalPolicyProvider, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paintComponents, preferredSize, printComponents, remove, remove, removeAll, removeContainerListener, setComponentZOrder, setFocusCycleRoot, setFocusTraversalPolicy, setFocusTraversalPolicyProvider, setLayout, transferFocusBackward, transferFocusDownCycle, validate

Methods inherited from class java.awt.Component
action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, contains, createImage, createImage, createVolatileImage, createVolatileImage, dispatchEvent, enable, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusCycleRootAncestor, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getForeground, getGraphicsConfiguration, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputContext, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocale, getLocation, getLocationOnScreen, getMouseListeners,

getMouseMotionListeners, getMousePosition, getMouseWheelListeners, getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners, getSize, getToolkit, getTreeLock, gotFocus, handleEvent, hasFocus, hide, imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isMaximumSizeSet, isMinimumSizeSet, isPreferredSizeSet, isShowing, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, postEvent, prepareImage, prepareImage, remove, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, resize, resize, setBounds, setBounds, setComponentOrientation, setCursor, setDropTarget, setFocusable, setFocusTraversalKeysEnabled, setIgnoreRepaint, setLocale, setLocation, setLocation, setName, setSize, setSize, show, show, size, toString, transferFocus, transferFocusUpCycle

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

XMLDataServerPanel

`public` **`XMLDataServerPanel`**`()`

Constructor for the XMLDataServerPanel.

HVA.XML.UTIL

CLASS XMLDOCUMENT

```
java.lang.Object
  └hva.xml.util.XMLDocument
```

Direct Known Subclasses:

CrestColorXMLDocument, XMLDataDocument

```
public abstract class XMLDocument
extends java.lang.Object
```

This is the XMLReader abstract class.

Author:

Patrick Newell

Constructor Summary
`XMLDocument()`

Method Summary

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

XMLDocument

```
public XMLDocument()
```

HVA.XML.UTIL

CLASS XMLLISTENER

```
java.lang.Object
  └java.lang.Thread
      └hva.xml.util.XMLListener
```

All Implemented Interfaces:

java.lang.Runnable

```
public class XMLListener
extends java.lang.Thread
```

This is the XMLListener class. It is responsible for handling the listening behavior for the XML Data Server. It will listen for XML data to be sent from Flash and in turn will write that XML data to a given XML file.

Author:

Patrick Newell

Nested Class Summary

Nested classes/interfaces inherited from class java.lang.Thread
`java.lang.Thread.State, java.lang.Thread.UncaughtExceptionHandler`

Field Summary

Fields inherited from class java.lang.Thread
`MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY`

Constructor Summary
`XMLListener(XMLSocketConnection xsc)` Constructor for the XMLListener.

Method Summary	
`void`	`run()` Runs the Thread.

Methods inherited from class java.lang.Thread
`activeCount, checkAccess, countStackFrames, currentThread, destroy, dumpStack, enumerate, getAllStackTraces, getContextClassLoader, getDefaultUncaughtExceptionHandler, getId, getName, getPriority, getStackTrace, getState, getThreadGroup, getUncaughtExceptionHandler, holdsLock, interrupt, interrupted, isAlive, isDaemon, isInterrupted, join, join, join, resume, setContextClassLoader, setDaemon, setDefaultUncaughtExceptionHandler, setName, setPriority, setUncaughtExceptionHandler, sleep, sleep, start, stop, stop, suspend, toString, yield`

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

XMLLISTENER

`public XMLListener(XMLSocketConnection xsc)`

Constructor for the XMLListener.

Parameters:

`xsc` - XMLSocketConnection object for communicating with Flash

Method Detail

RUN

`public void run()`

Runs the Thread.

Specified by:

`run` in interface `java.lang.Runnable`

Overrides:

`run` in class `java.lang.Thread`

HVA.XML.UTIL

CLASS XMLLISTENERTHREAD

```
java.lang.Object
  └java.lang.Thread
      └hva.xml.util.XMLListenerThread
```

All Implemented Interfaces:

java.lang.Runnable

```
public class XMLListenerThread
extends java.lang.Thread
```

This is the XMLListenerThread. It is responsible for making a connection to Flash to begin listening for XML data.

Author:

Patrick Newell

Nested Class Summary

Nested classes/interfaces inherited from class java.lang.Thread
java.lang.Thread.State, java.lang.Thread.UncaughtExceptionHandler

Field Summary

Fields inherited from class java.lang.Thread
`MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY`

Constructor Summary
`XMLListenerThread(java.net.ServerSocket ss)` Constructor for the HVAWiiListenerThread.

Method Summary	
`void`	`removeListeners()` Removes all listeners from the server.
`void`	`run()` Starts the main running loop of the server.
`void`	`startListening()` Sets the stop flag to false in the main loop.
`void`	`stopListening()` Sets the stop flag to true in the main loop.

Methods inherited from class java.lang.Thread
`activeCount, checkAccess, countStackFrames, currentThread, destroy, dumpStack, enumerate, getAllStackTraces, getContextClassLoader, getDefaultUncaughtExceptionHandler, getId, getName, getPriority, getStackTrace, getState, getThreadGroup, getUncaughtExceptionHandler, holdsLock, interrupt,`

`interrupted, isAlive, isDaemon, isInterrupted, join, join, join, resume, setContextClassLoader, setDaemon, setDefaultUncaughtExceptionHandler, setName, setPriority, setUncaughtExceptionHandler, sleep, sleep, start, stop, stop, suspend, toString, yield`

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, wait, wait, wait`

Constructor Detail

XMLListenerThread

`public **XMLListenerThread**(java.net.ServerSocket ss)`

Constructor for the HVAWiiListenerThread.

Parameters:

`ss` - ServerSocket object for server

Method Detail

RUN

`public void **run**()`

Starts the main running loop of the server.

Specified by:

`run` in interface `java.lang.Runnable`

Overrides:

`run` in class `java.lang.Thread`

STOPLISTENING

`public void` **`stopListening`**`()`

Sets the stop flag to true in the main loop.

STARTLISTENING

`public void` **`startListening`**`()`

Sets the stop flag to false in the main loop.

REMOVELISTENERS

`public void` **`removeListeners`**`()`

Removes all listeners from the server.

HVA.WII.NET

CLASS XMLSOCKETCONNECTION

```
java.lang.Object
  └hva.wii.net.XMLSocketConnection
```

All Implemented Interfaces:

IConnection

```
public class XMLSocketConnection
extends java.lang.Object
implements IConnection
```

XMLSocketServer class designed to interface with an ActionScript 2.0 XMLSocket in a Flash application.

Author:

Patrick Newell

Constructor Summary
`XMLSocketConnection(java.net.ServerSocket server, java.net.Socket client)` Constructor for XMLSocketServer

Method Summary	
`java.net.Socket`	`getClientSocket()` Returns the ClientSocket that is listening.
`java.io.BufferedReader`	`getInStream()` Returns the BufferedReader containing the input stream for the client

	socket.
`java.net.ServerSocket`	`getServerSocket()` Returns the ServerSocket that is being listened to.
`void`	`send(java.lang.String message)` Sends a given message over the output stream.

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

XMLSOCKETCONNECTION

```
public XMLSocketConnection(java.net.ServerSocket server,
                           java.net.Socket client)
```

Constructor for XMLSocketServer

Parameters:

`server` - ServerSocket being listened

Method Detail

SEND

```
public void send(java.lang.String message)
```

Sends a given message over the output stream. Automatically terminates message with a 0.

Specified by:

`send` in interface `IConnection`

Parameters:

`message` - String value of message to send

GETINSTREAM

```
public java.io.BufferedReader getInStream()
```

Returns the BufferedReader containing the input stream for the client socket.

Specified by:

`getInStream` in interface `IConnection`

Returns:

BufferedReader containing input stream

GETSERVERSOCKET

```
public java.net.ServerSocket getServerSocket()
```

Returns the ServerSocket that is being listened to.

Specified by:

`getServerSocket` in interface `IConnection`

Returns:

ServerSocket object for this class

GETCLIENTSOCKET

```
public java.net.Socket getClientSocket()
```

Returns the ClientSocket that is listening.

Specified by:

`getClientSocket` in interface `IConnection`

Returns:

Socket object for client

Appendix XI: Original IQP Proposal

[THE VIRTUAL JOUST]

One of the more unique events during medieval tournaments was the joust. In this event, mounted knights would charge at their enemies using various weapons, with the intention of dismounting them. In an effort to replicate this event, a virtual jousting game will be produced. Four main areas of research will support this project: historical context of the joust, mechanics of the joust, design concepts, and technical concepts. Background research on the joust will be required to set up the historical context and game mechanics, and design and technical research will be required in order to construct the actual game.

1 – General Information

1.1 – Acronyms and Symbols

The following is a list of common acronyms used throughout this paper.

- HAM – Higgins Armory Museum
- HVAT – Higgins Virtual Armory Team
- IQP – Interactive Qualifying Project

1.2 – Introduction

At the moment, there are four main time frames for this IQP. The frames include: brainstorming for a project [Term A], project planning and research [Term B], project development [Term C], and project finishing [Term D]. Term A involves seeking the advice of HAM staff in order to decide on a project that would enhance the museum's collections. Term B involves outlining dates for deadlines of project work. Term C involves developing a functional application. Finally, Term D involves fixing up the application for public viewing and use. Currently, the brainstorming for a project phase has been completed, and the project planning phase is in the works.

For Term A, each group member was assigned one of the following areas of research: HAM material, HAM visitorship, design concepts, and technical concepts. During this phase of the project, interviews and meetings were conducted with HAM staff in order to determine a project direction that would benefit the museum. In addition to these interviews and meetings, independent research was done on the museum itself and on projects other museums have pursued. In order to select a project that would be feasible in the time frame our group was allotted, design and technical research was performed. We have approximately seven weeks to construct an application to keep the attention of participants for about three to four minutes.

For Term B, each group member was assigned one of the following areas of research: historical context of the joust, mechanics of the joust, design concepts, and technical concepts. The plans of work are varied, but everyone is expected to be able to contribute their respective parts before the beginning of Term C. During this phase, the individuals researching the historical context and mechanics of the joust should be frequenting the HAM library to obtain material for application context, parameters, and functions. The individuals working on the design and technical aspect should be brainstorming possible graphics/sounds and implementation methods (e.g. internal structure of program, design for possible extendibility, user input devices and control, etc.) for the application.

For Term C, everyone will consolidate and contribute their share of research in order to reach a consensus for the final application. Upon reaching this accord, the group will begin materializing the application in a joint fashion. For Term D, the application will be shown to and reviewed by HAM staff and possible outside beta testers. From the input provided by the staff and beta testers, final touches and fixes will be applied, and the end of Term D should bring forth an application that is ready for viewing by clients of the HAM.

2 – TERM A: BRAINSTORMING FOR A PROJECT

2.1 – HAM INFORMATION

This section contains information about how we will perform the background research that goes will go into the virtual application. More specifically, this section will list what we currently known about the Higgins Armory Museum and how we will get some of our information about the joust.

2.1.1 – CURRENTLY KNOWN INFORMATION

The Higgins Armory Museum carries a collection of arms and armors from various ages in a gothic castle setting. Regarding its European collection, it has 3000 armors and components, 1000 weapons and accessories, 500 swords and daggers, 100 firearms, and several other artifacts. In addition to this impressive European collection, the HAM also boasts over 1000 non-western arms and armors (African, Islamic, Indian, and Japanese).

A jousting simulation will be the product of this IQP. Currently, it is known that jousting is a sport where mounted knights charge at each other with a large lance, attempting to knock his opponent to the floor. The armor involved with the joust is specially designed so that there is a large shoulder guard for where the lance is to hit. Also, the helmet is designed so that the opening is near the top; this way, when the lances and armors collide, the splinters will not fly into the knight's unarmored visage.

2.1.2 – RESOURCES

Higgins Armory Museum (2002). The Age of Armor. Virginia: The Donning Company Publishers.

- Book that describes the history of armor, in a narrative style. It also features many items that could be found in the Higgins collection.

Collections-Public Database.

- A collection of numerous items from the Higgins Armory in greater detail. This would help significantly in the graphics collection part of the project.

HAM Bibliography.

- A large collection of sources in the Higgins library that may be of assistance for the upcoming project. This would help significantly in collection of primary materials by providing us with foresight into which resources would be helpful without spending too much time browsing.

Higgins Website

- This source can provide background information on the HAM. This includes the armory's mission, collections, and goals.

Modeling the Joust IQP (+ sources cited in this paper)

- This will help a lot in the research phase, providing background on the joust as well as documentation for further research.

2.1.3 – Possible Contacts
Jeffrey L. Forgeng

- Supervisor of the project... may also provide materials related to the project on a regular basis. Can help much in directing students toward the right people/resources once a definite project has been determined.

2.1.4 – Task List
The following is a task list generated based on the current condition of the project:

Resource Gathering

- Look at previous IQP's involving the HAM and/or the joust.
- Navigate around the Higgins site for relevant information.
- Get oriented with the Higgins library; gather relevant materials from the Higgins library.
- Search for additional information from outside sites, if not enough information was gathered.

Using Information

- Enlighten group about the mechanics and context of the joust.
- Gather media for the project (pictures, sounds, etc).

Other

- Assist with technical aspect of project.

2.2 – HAM Visitorship
This section details information about who is visiting the museum. It also will help with learning about our future users, which will be preteens.

2.2.1 – *Currently Known Information*

Based on the interview with Nikki, we know that most people that come to museum are younger kids or school groups.

- The younger kids that are coming are heavy into knights, dragons, and princesses. This is the primary reason for them coming to the museum; they want to see what a knight actually looks like.

The museum might have a small survey of what people thought of it overall. This would be useful to find and review.

Based on the statistics given to us, school groups account for ¼ to 1/3 of museum visitors. Children coming per year not in a group averages about 6000 a year while school groups are about 10000.

- Most likely due to weakening economy, school group attendance has been falling. It was 18000, but now is about 10000.

2.2.1.1 – Visitorship Report

Overall the average number of visitors per year is 50000. However that number seems to be slowly going down. The greatest loss is from school groups. This is most likely due to the economy. The lack of funding at schools is preventing them from coming to the museum. There is no means that we can provide to reverse the problems with the economy to improve this number.

Despite this, school groups are still a sizeable piece of the visitors. On average, a school group will come and there will be 1 adult chaperone for every 6 kids. It would be useful to know what age group the children in these school groups full into. The age of the school group would be useful to know so that it would be easier to determine what the average age of a child at the museum.

Other organizations and events do not pull in any major numbers for visitors. Very on average, it seems that between all other groups there is only 5000 to 6000 in visitors. These numbers are appear to be lower than average which is mostly likely due to the economic outlook. People do not have spare money to spend on trips to museums.

The spreadsheet provided does not specific families, which has been mentioned as one of the three major sources of people that come. There is an adults and children section on the chart so averages can be used to make some conclusions. If the standard family is 4 members, 2 parents and 2 kids, this would result in an average of 3000 family visits per year. This would mean an average of 7000 adults come of their own accord to view the collection.

While the provide chart is useful, there are areas we can question. It would be useful to know if they have any information about what the average size of a family is. Does one parent bring the kids to the museum by themselves making the family 1 adult and 2 kids on average? Another category, labeled free, does not specify the age group. There is no way to know who is getting in free. Are these very young kids that are getting in for free? If so, there is a large number of babies that are coming. Inferring that families have their kids at 2 year intervals, this would mean that most families that do come have very young kids. This would match the information that we have been given already.

The last area to get more information about would be the outreach section. What is this outreach? In 2006, it had a loss of 50%. Is this due to marketing or something else? It would also be useful to know what the age group breakdown in outreach is to get more detailed information.

2.2.2 – Resources

http://www.eduweb.com/company.html

- Research on what makes an interactive work and what makes it fail.

http://www.aam-us.org/museumresources/ic/index.cfm

- Has information about committee that works on interactive and media technology.

http://www.accesswave.ca/~infopoll/tips.htm

- Basics on how to write a good survey

http://www.ehow.com/how_16596_write-survey-questionnaire.html

- More information on how create a survey or questionnaire.

http://www.cc.gatech.edu/classes/cs6751_97_winter/Topics/quest-design/

- Data on questionnaire design

http://www.ssdd.bcu.ac.uk/learner/writingguides/1.05.htm

- General information on questionnaire writing.

http://www.surveybounty.com/articles/write.html

- General survey writing tips. Also has advantages of online surveys and determine sample sizes.

http://www.streetdirectory.com/travel_guide/5094/marketing/20_top_tips_to_writing_effective_surveys.html

- Top 20 tips for how to write effective surveys

Bradburn, Norman (2004). *Asking questions : the definitive guide to questionnaire design : for market research, political polls, and social and health questionnaires.* San Francisco: California. http://library.wpi.edu/cgi-bin/Pwebrecon.cgi?BBID=226926

- Book on questionnaire design.

Caulton, Tim (1998). *Hands-on exhibitions: managing interactive museums and science centres".* London: New York. http://library.wpi.edu/cgi-bin/Pwebrecon.cgi?BBID=202917.

- How to managed interactive in museums and idea of what to do

Dillman, Don (2000). *Mail and internet surveys : the tailored design method.* New York: http://library.wpi.edu/cgi-bin/Pwebrecon.cgi?BBID=225339

- More information on surveys

Froddy, William (1993). *Constructing questions for interviews and questionnaires : theory and practice in social research.* New York: New York. http://library.wpi.edu/cgi-bin/Pwebrecon.cgi?BBID=201664.

- Information on how to conduct interviews and how to use questionnaires

Punch, Keith (2003). *Survey research: the basics.* London: California. : http://library.wpi.edu/cgi-bin/Pwebrecon.cgi?BBID=268043.

- How to do survey research of people

Roberts, Lisa (1997). *From knowledge to narrative: educators and the changing museum.* Washington D.C. http://library.wpi.edu/cgi-bin/Pwebrecon.cgi?BBID=154888.

- How has the museum evolved in the past. Important information for the future.

Janet, Collins (1997). *Teaching and learning with multimedia.* London: New York. http://library.wpi.edu/cgi-bin/Pwebrecon.cgi?BBID=286762.

- How to use multimedia as a teaching tool.

Simkins, Michael (2002). *Increasing student learning through multimedia projects.* VA: Alexandria. http://library.wpi.edu/cgi-bin/Pwebrecon.cgi?BBID=223748

- Information about creating interactive that increase student learning.

Higgins Armory Attendance Report: Compiled by the Higgins Armory Museum Staff. 2008

2.2.2.1 – Methods to Obtain more Information

Interviews

We have currently only interviewed one member of the Armory Staff. The interview with Nikki provided a good insight to the type of people that come to the museum. By interviewing the other staff members, it might be possible to extend this insight.

Surveys

Surveys about the museum experience are useful for knowing about the people there. Understanding their needs and wants is what we are after. In particular, a survey specifically aimed at younger visitors might be good. Currently the museum might have a generic survey that is normally answered by parents. We need to know how the kids felt.

Observation

Two interactive exhibits are being installed. By watching how people interact with these exhibits it is possible to extend the good parts and repair the bad. This would require prior set up and require someone to be there to watch.

2.2.3 – Possible Contacts

Nikki Andersen

- E-mail: nandersen@higgins.org
- Extension: 22
- Extensive knowledge gained from work at Indianapolis Children's Museum. Has experience with the public relations section of the museum. Useful for setting up various surveys or getting visitor information from the museum.

Devon Kurtz

- E-mail: dkurtz@higgins.org
- Extension: 15
- As director of education, will have general knowledge about people at the museum. Might have other information as to what type of educational experience he would like to have the interactive provide. He might also have more detailed information about what types of learning activities have been successful in the past.

Linda Woodland

- E-mail: lwoodland@higgins.org
- Extension: 23

- As project manager of exhibits, will have knowledge about which exhibits have failed and which have been successful. Might have information regarding how people react to different types of exhibits and what the public has issues with while they are at the museum.

2.2.4 – Task List

Requirement Gathering

- Determine what the museum wishes to have for an exhibit and which age group they want targeted for the exhibit.
- Using currently installed interactive exhibits, take surveys to determine what the people using them liked and what they didn't like about an exhibit.
- Do a mock up user interface and seek others opinions on it from the museum.
- Meet with Higgins staff after final requirements are laid out. Confirm with them that this is what they are after.

Intermediate Tasks

- Get surveys of how people perceive the museum. What did they like. What would they like to see?
- Look into possible sources of target age group for testing of the product before it is finally released.
- Look into creating testing dates for the project to see how what problems the users might be encountering with the current design.

Interactive Specific Tasks

- Continuously keep Higgins staff in loop of project development. This will ensure that the aim of the project stays in focus to both the development group and the client.

2.3 – Technical Details

This section covers information on the technical details for the project. We currently plan on running our virtual jousting simulation in a kiosk-based environment. Other goals of this section is to ensure that the project is developed in a way that will allow for extendibility, not only on a software level, but also by extending the simulation onto the web through web services and applications.

2.3.1 – Possible Technical Options

2.3.1.1 – Programming Languages

This project will require some form of a programming language in order to write the necessary scripts and procedures to develop the software level of the project. Below are some potential languages to work with:

Java **

This language is very verbose but is mature and very powerful. It allows for object-oriented programming, which allows for increased extendibility and can also be deployed to almost any environment running the correct Java virtual machine.

Actionscript ***

This is the programming language used in unison with Flash development. If the graphics are to be developed in a Flash environment, a significant amount of Actionscript programming will most likely be needed.

Python *

This language is extremely object-oriented and is fairly quick to develop with. It can be primarily used as a scripting language and is also very useful in web development.

PHP *

This language is similar to Python, as it is another scripting language that is very popular in the web-development world.

2.3.1.2 – Software

Flash

This would be the most feasible means by which to generate graphics and animation required to generate a virtual jousting simulation. The limitation to Flash would be that generating a 3D environment will require a significant amount of work.

Sustainability: Using this software would allow for relatively easy maintenance compared to the other options available, as it is fairly easy to learn and use with some technical experience.

Maya

This is an advanced 3D modeling engine that would be necessary for creating actual 3D models if a true 3D environment was to be created.

Sustainability: Using this software would make the project somewhat difficult to maintain, as future developers would have to learn 3D modeling concepts and understand how they would be integrated into our project.

Various 3D Game Engines

Created an actual 3D game engine for the user to interact with would most likely require more development time than there is allotted, so looking into other 3D game editors might be useful. Games such as Half-Life, Unreal Tournament, and Warcraft III tend to be very developer friendly game engines with their own customizable development software.

Sustainability: Using this software would also be somewhat difficult to maintain, as it would require future developers to have to learn a specific set of editing tools, which may not have the capabilities for future extensions.

2.3.2 – Resources

Dai, Naci; Mandel, Lawrence; Ryman, Arthur. Eclipse Web Tools Platform.

- This book provides excellent information and tutorials on developing java-based web applications.

McGrath, Mike. PHP 5 in Easy Steps.

- This book provides a basic understanding of PHP as a web development tool and discusses the basics behind how it interfaces with MySQL on an Apache web server.

McLaughlin, Brett. Head Rush Ajax.

- This book provides an excellent understanding of the concept of asynchronous Javascript and XML (Ajax), which is a key feature of modern web applications.

Palmer, Daniel W; Steinberg, Daniel H. Extreme Software Engineering.

- This books provides information on good software engineering and development practices.

Sebesta, Robert W. Programming the World Wide Web.

- This book is an excellent resource for the various technologies used in web development.

2.3.3 – Possible Contacts

Professor Gary Pollice

- Professor of Computer Science that has a great knowledge of software development practices. He also teaches the "Webware" course, which may be relevant to this project.

Professor Robert Lindeman

- Professor of Computer Science that specializes in virtual environments. He may be very useful in helping us develop hardware interfacing.

2.3.4 – Task List

Analyze the project to determine system requirements:

Server/Platform system requirements (processing power, persistent memory, RAM, video, audio)

- This project should require a desktop computer with about 1.8 ghz of processing power, 1 gigabyte of RAM, and 128 megabytes of VRAM at a minimum for this project.
- Any modern CRT or LCD monitor would be sufficient for this project at a minimum. If more of an immersive approach was to be taken, a smaller LCD screen that would act as the front of a helmet would be useful. Also, a secondary screen to allow spectators to view the joust from a 3rd person view would also add a layer of depth.
  - Small LCD Screen (15"): approx. $130.00
  - Small CRT Screen (15"): approx. $90.00
  - Mid-Sized LCD Screen (19"): approx. $150.00
  - Mid-Sized CRT Screen (17"): approx. $175.00
  - Large LCD Screen (22"): approx. $180.00

- A 2.0 or 2.1 speaker configuration would be sufficient for this project.
  - 2.0 speakers: approx. $5.00
  - 2.1 speakers: approx. $20.00
- More accurate specifications will be created when development begins.

Non-standard input devices (joysticks, gamepads, gyro-mice, etc)

- The cheapest device to be used in this project would most likely be a mouse. Extending the interactivity to more advanced devices, such as mice with gyro-sensors (similar to a Wii-mote) or joysticks would be more immersive, but would cost more both financially and in development.
  - Standard Mouse: approx. $5.00
  - Mouse with Gyro-Sensors: approx. $60.00
  - Wii Remote: approx. $25.00

Networking specifications (bandwidth, network speed, routers, etc)

- This will most likely not be needed for the bulk of the development for this project, but should be considered for future extendibility or future proofs of concept.

Third-party software (JVM, Apache, MySQL, etc)

- There should be no third-party services required to be run, unless Java is used, which would require a JVM to be installed. If Flash is used, then a Flash player will be required to use the software.
  - All speculated third-party services are free in price.

Other Tasks

- Analyze the project on a deeper level and develop requirements and user stories for the software. Modeling would also occur at this stage if need be.
- Determine a development environment and determine the programming language(s) to be used.
- Iteratively develop the software by completing user stories that encapsulate the system requirements and constantly test on a development level (this will be the where the bulk of the work should occur).
- Perform extensive integration-level tests.
- Deploy software to production system and perform extensive production-level tests.
- This would include public beta-testing.

2.4 – Design Details

The graphic design domain will encompass the end-user interface. It will require close coordination with the technical domain in terms of producing said interface, and will be primarily responsible for generating graphics, video, and related ideas. In this manner, it will be necessary for the graphics design domain to learn technical aspects of graphics generation, including animation techniques and graphic software use. That said, a foundation in design is a must-have, and initial research will develop such a foundation.

2.4.1 – Resources

http://www.flashkit.com/

- Web community-based flash help site. Has resources like tutorials, examples, sound effects and loops, and forum-based help.

http://www.graphic-design.com/Photoshop/index.html

- Contains tons of tutorials on photoshop and illustrator.

http://www.learner.org/interactives/

- List of interactive for educational purposes. Good examples and application for the most part, but reading-heavy

Ballinger, Raymond A. *Layout and Graphic Design*. Call number Z253.5 B28 1970

Heller, Steven; Pomeroy, Karen. *Design Literacy: Understanding Graphic Design*. Call number NC998 H45 1997

Hashimoto, Alan. *Visual Design Fundamentals: A Digital Approach*. Call number TA174 .H313 2004eb

"The Art of Computer Animation" (video). Call number TR897.7 A78 1988

Morris, Dave; Hartas, Leo. *Game Art: The Graphic Art of Computer Games*. Call number T385 M663 2003

Modeling the joust. (Library call number 02C032I).

- This IQP involve students modeling the medieval joust from a physics viewpoint. Information contained in the IQP include equipment and history. Should be a good starting point for research into "Rules of the Joust" and "Game Mechanics."

Royal Air Force Museum exhibit and presentation design. (Library call number 99D242I).

- This IQP involved students creating an exhibit for the Royal Air Force Museum that would explain the concept of metal fatigue to 12 year old children. Should be able to give us a preliminary list of resources, especially since our target audience is also high elementary – middle school.

Animating an Elizabethan suit of armor. (Library call number 03D057I).

- This IQP involved students creating a multimedia exhibit in which visitors could view a suit of armor from multiple angles. This should be a good resource from a technical and design perspective, as it would be possible to glean what "works" in 3D animation.

2.4.2 – Possible Contacts

Dean O'Donnell

- Acting Director of IMGD at WPI
- As the director of IMGD, most likely has valuable contacts and advice on interactive in general. As a professor at WPI, reasonably easy to reach.

Rob Lindeman

- Assistant Professor at WPI
- Interest in irregular input devices and incorporating multiple senses into game design.

Operand

- Professional interactives company. Perhaps it will be possible to draw inspiration from their portfolio / philosophy. Web site at http://www.operand.com/.

Robert W. Thompson

- E-mail: rwt@wpi.edu
- Faculty advisor for Royal Air Force Museum IQP

Douglas W. Woods

- E-mail: dwwoods@wpi.edu
- Faculty advisor for Royal Air Force Museum IQP
- Seems like the IGSD faculty.

2.4.3 – Task List

General Tasks

- Preliminary design (sketches, "story"boards). Aiming for at least a level 3 interactivity (Participatory), design and pseudocode a game that will satisfy to our goals. Along with this, primary graphics generation and an estimation of time required to animate and incorporate all resources should be done.
- Basic search for available resources. If there has been 3D models or graphics already generated somewhere for public use, try to employ as much as possible.
- Graphics generation / animation as needed. If there are elements missing from initial search for public resources, it will be necessary to create original works to cover those gaps.
- Scripting / synthesis (working closely with technical person).
- Test phase. Contact multiple people with alpha and / or beta releases. Debugging as necessary, encourage testers to "break" program.
- Publication.

3 – Term B: Project Planning and Research

3.1 – Historical Context of the Joust

This section is responsible for researching the historical and social context of the joust. Items such as knowing how the joust began and how its significance changed throughout the different periods of time will be examined. This knowledge is necessary for our project since we need to include some historical information in the project.

3.1.1 – Primary Deliverables

- Research documents regarding the history of the joust

3.1.2 – Plan of Work

Week 1

- Initial research and source gathering. Gain basic understanding of the joust and find different sources that will be useful for continuing research
- Submit: an annotated bibliography and a list of possible subtopics that can be researched in the upcoming weeks

Week 2

- Start research on the beginnings of the joust. This will have information up to the 13th century detailing how the joust came to beginning and how it became popular
- Submit: outline of researched information that has been found out so far

Week 3

- Finish research on the beginnings of the joust. This will have information up to the 13th century detailing how the joust came to beginning and how it became popular
- Submit: write up of research done on the beginnings of the joust

Week 4

- Begin research up to the 15th century context of the joust. What rituals surrounded the joust and what the impact the joust had on society
- Submit: outline of the researched information that has been found out so far. Do refinement of the beginnings of joust paper

Week 5

- Finish research up to the 15th century context of the joust. What rituals surrounded the joust and what the impact the joust had on society
- Submit: write up of research done on the beginnings of the joust and any other refactoring of previous work

Week 6

- Begin research on the 16th and 17th century section of the joust
- Submit: outline of the information that has been found so far and any refactoring of previous work

Week 7

- Wrap up. Finish all research. Put together all researched material and place into one paper

3.1.3 – References

- Anglo, Sydney (2000). *The martial arts of Renaissance Europe.* New Haven: CT. Information about jousting with the heavy lance.
- Anglo, Sydney (1991-1992). *Jousting: The Earliest Treatises.* Basics of jousting
- Barber, Richard (1989). *Tournaments: Jousts, chivalry, and Pageants in the Middle Ages.* New York. The joust as part of the tournament.
- Barker, Juliet (1986). *The Tournament in England, 1100-1400.* Wolfeboro: New Hampshire. Social customs including information on the tournament
- Clephan, R (1919). *The Tournament: its periods and phases.* London: Methuen. Information about the various periods of the tournament.
- Clephan, Robot (1967). *The Tournament.* New York. Basic information on the joust.
- Cripps-Day, Francis (1918). *The history of the tournament in England and in France.* History of the tournament in England and France.
- Crouch, David (2005). *Tournaments: The Medieval Sport of Battle.* Information on the tournament
- Randolph, Adrian (2002). *Engaging Symbols: gender, politics, and public art in fifteenth-century Florence.* Information about the joust as a symbol of society.
- Young, Alan (1987). *Tudor and Jocobean tournaments.* Dobbs Ferry: New York. Information on these types of tournaments.

3.2 – Mechanics of the Joust

This section is responsible for researching the mechanics of the joust. This includes looking at the rules, armor, lance play, and horses involved with the joust. It also includes background research into the different types of jousting that took place during tournaments. This background information is important in order to accurately replicate the joust in the application.

3.2.1 – Primary Deliverables

- Research documents regarding the mechanics of the joust
- List of game variables and functions with respect to jousting mechanics

3.2.2 – Plan of Work

Week 1

- Go through list of references and annotate them
- Brainstorm list of subtopics for major research areas in this field
- Submit: list of subtopics and annotated references

Week 2

- Research rules + equipment of the joust
- Use HAM library, online sources, and contacts
- Submit: outline of rules + equipment of the joust

Week 3

- Continue research started on Week 2
- Submit: rough draft of paper on the rules + equipment of the joust

Week 4

- Research lance play
- Use HAM library, online sources, and contacts
- Submit: outline of lance play

Week 5

- Continue research started on Week 4
- Submit: rough draft of paper on lance play

Week 6

- Research horsemanship
- Use HAM library, online sources, and contacts
- Work on final paper for the mechanics of the joust
- Submit: outline of horsemanship

Week 7

- Develop a working model for the application (parameters and functions used)
- Submit: finalized paper for the mechanics of the joust (including the physics)

3.2.3 – Contacts

Jeffrey Hedgecock

- jeffrey@historicenterprises.com
- Jouster

Luke Binks

- luke@historicenterprises.com
- Jouster

Jenna Reed

- jlrr@comcast.net
- Research on medieval equestrianism

3.2.4 – References

Rules and Equipment of the Joust

- Ashdown, Charles Henry. *European Arms and Armour*. New York, NY: Brussel & Brussel, 1967.

- Barber, Richard and Juliet Barker. *Tournaments: Jousts, Chivalry and Pageants in the Middle Ages*. New York: Weidenfeld and Nicolson, 1989.
- Blair, Claude. *European Armour*: *Circa 1066 to Circa 1700*. London: B.T. Batsford LTD, 1958.
- Clephan, R. Coltman. *The Tournament, its Period and Phases*. London: Methuen & Co. Ltd., 1919.
- Edge, David and John Miles Paddock. *Arms and Armor of the Medieval Knight.* New York, NY: Crescent Books, 1990.
- Sydney Anglo (1968). *The Great Tournament Roll of Westminster. A collotype reproduction of the manuscript.* Oxford: Clarendon Press.
- Sydney Anglo (1991-92). "Jousting: The Earliest Treatises." *Livrustkammeren: Journal of the Royal Armoury, Stockholm* 18. 3/23/2008.

Lance Play

- Sydney Anglo (1988). "How to Win at Tournaments: The Technique of Chivalric Combat." *Antiquaries' Journal* 68. 248-64.
- Sydney Anglo (2000). *The Martial Arts of Renaissance Europe.* New Haven: Yale University Press. JLF.
- Charles ffoulkes;E.C. Hopkinson (1938 (Facsimile Edition, 1967)). *Sword, Lance and Bayonet.* London: Arms and Armour Press.

Horsemanship

- (1981). *Glorious Horsemen: Equestrian Art in Europe, 1500 to 1800.* Springfield: Museum of Fine Arts.
- Frederick Henry Huth (1887). *Works on Horses and Equitation: A bibliographical record of hippology.* London: Bernard Quartch.

3.3 – Design Details

This subsection is responsible primarily for the graphical design of the end-product. It will encompass designing the storyboard as well as the visuals, leading the team in coming up with a flow that will appeal to the target audience. The subsection will also be responsible for the primary generation of graphics through Flash or other media.

3.3.1 – Primary Deliverables

- Sketches of jousting and non-jousting scenes
- "Screen stills" of various scenes
- Developed samples of art and animation for the game

3.3.2 – Plan of Work

Week 1

- Design and submit user storyboard. This will provide a sort of flow chart for the interaction between the users and the interactive, giving an idea on what the first steps can be
- Report on art design-related book (number 1)
- Bring analog "reference samples" for comparison

Week 2

- Iterate storyboard
- Report on art design-related book (number 2)
- Bring more reference samples

Week 3

- Iterate storyboard
- Report on art design-related book (number 3)
- Sketches of various scenes

Week 4

- Iterate storyboard and sketches
- Sketches of components (jouster, horse, etc.)
- Submit: rough sketch of interface for review

Week 5

- Iterate storyboard and sketches
- Refine jousting sketches
- Sketches of user interface
- Begin animation samples

Week 6

- Iterate animation for jousters
- Refine components
- Submit: "screen stills" of various scenes

Week 7

- Continue to refine art work
- Begin animation on multiple scenes

3.4 – Technical Details

This subsection is focused on developing a functional and extendible design for the application. This area of work will produce a framework for the team to develop the project, as well as a detailed system diagram or a set of diagrams to help the team obtain a better understanding of the framework.

3.4.1 – Primary Deliverables

- System diagram of framework
- Multiple demos featuring different modules

3.4.2 – Plan of Work

Week 1

- Generate and submit an initial draft of the system in the Unified modeling Language (UML). Key objects and components should be described in this diagram in order to achieve a better understanding of the system from a large-scale view
- Begin research and development of framework for input devices in flash

Week 2

- Determine final input device to use for development. This should be open to extension to allow different types of input devices to be used in the future.
- Develop and submit a user-input demo that will capture and demonstrate all possible user input commands. This should utilize and demonstrate the user-input component of this project
- Modify system diagram as needed
- Submit: system diagram

Week 3

- Research and develop collision detection techniques in a generic form (i.e. develop algorithms to calculate two objects approaching each other and when they meet)
- Develop a demo that shows the functionality of the advanced collision detection component of the system
- Modify system diagram as needed
- Submit: demo and system diagram

Week 4

- Research and develop advanced collision detection techniques to detect lance hits on various parts of the jouster's shield or the jouster
- Develop a demo that shows the functionality of the advanced collision detection component of the system
- Modify system diagram as needed
- Submit: demo and system diagram

Week 5

- Fully integrate both of the collision detection sub-systems and the user-input framework to develop a proof of concept for the general core mechanics of the joust. Research from other groups should be taken into account for this demo
- Develop proof-of-concept demonstration for the integrated subsystem described above
- Modify system diagram as needed
- Begin work on sound output framework
- Submit: demo and system diagram

Week 6

- Finish development of sound output framework and demonstrate its functionality

- Develop a demonstration of user customization (i.e. the user should be able to change the type of armor, horse, weapon that is used)
- Modify system diagram as needed
- Submit: demo and system diagram

Week 7

- Finalize the jousting simulator system diagram. The diagram should detail every aspect of the system from a framework standpoint. At this point, it should be the road-map by which the group should be able to follow in the development of the project
- Finalize all modules of the framework. Ensure all code and other work if properly commented and documented
- Submit: modules and system diagram

4 – Term C: Developing of the Application

4.1 – Primary Deliverables

The primary deliverables expected after Term C include:

- A functional jousting game
- Updated project proposal
- Portfolio of materials submitted during the term
- Personal statement from each team member highlighting personal accomplishments during the term, as well as critical assessment of these accomplishments

4.2 – Plan of Work

Week 1 – 3: Jousting Simulator Development

- Derive and implement game mechanics.
  - Player variables and statistics
  - Win/Loss determinants
  - Scoring system
  - Joust parameters (time limit, number of passes)
- Implement Story Board Aspects
  - Training/Tutorial
  - Story of Joust
  - Final sequence of events
- Finalize graphics and animation
  - Implement final cut-scenes
  - Implement final game-play animations and graphics
  - Implement final player-customization graphics
  - Implement final animations and graphics for storyline

Week 4 – 5: Administrative Component Development

- Design interface for administrative console
- Implement administrative customization
    - Customizable timeouts and number of passes
    - Game content editing
    - Enable/Disable any web features
- Design and implement data persistence model (XML or Relational Database)

Week 6: Hardware Development and Systems Deployment

- Hardware development (Some components likely to be completed earlier if needed)
    - Build lance (Wii Remote, configure IR LED array)
    - Build helmet (incorporate sound system)
    - Build horse (Wii Balance Board)
    - Build and configure desktops (Servers and Clients)
    - Setup and configure displays (Touch-screens and monitor)
    - Create security measures to safeguard all hardware developed
    - Initial designs for kiosk setup
- System Deployment
    - Software deployment documentation
        - Installation and configuration procedures
        - Maintenance procedures
        - Troubleshooting procedures
        - Un-installation procedure
    - Error reporting/fixing services
        - Implement bug tracker system
        - Documentation on reporting bugs
    - Develop installation and software

Week 7: Web Functionality and Beta Testing

- Web Functionality (may be developed earlier if possible)
    - Web interfacing for user customization (client/server communication)
    - Data acquisition from stored data (accessing user's data from jousting application)
    - Implement data persistence model to be compatible with web interface
    - Score-card emailing
    - Multiplayer
- Beta Testing
    - Integration testing for cross-platform performance
    - Public testing for player-base input
- Demonstrate system to all parties interested.

5 – TERM D: FINALIZING THE APPLICATION

5.1 – PRIMARY DELIVERABLES

The primary deliverables expected after Term D include:

- A functional jousting game that is ready to be shown to HAM visitors
- A final report detailing the research and work involved with this project; this will also contain documentation for the code of the application for future projects interested in extending on it

5.2 – Plan of Work

Week 1

- Collect input from staff and beta testers
- Apply suggested changes and fixes that seemed reasonable
- Release for testing again

Week 2

- Collect input from second test run
- Apply suggested changes and fixes that seemed reasonable
- Work on the introduction of the project report
- Submit: introduction of the project report

Week 3

- Touch up on the interface if needed
- Release for testing again if previous changes and fixes appear major
- Work on conclusion of the project report
- Work on team bios and photos
- Work on project appendices
- Submit: conclusion of the project report, project appendices, and team bios and photos

Week 4

- If testing was done again, collect input and decide if fixes/changes need to be implemented
- Work on project abstract and acknowledgments
- Submit: project abstract and acknowledgments

Week 5

- Put together the final IQP report
- If necessary, add in final touches to application
- Submit: final version of application and project report

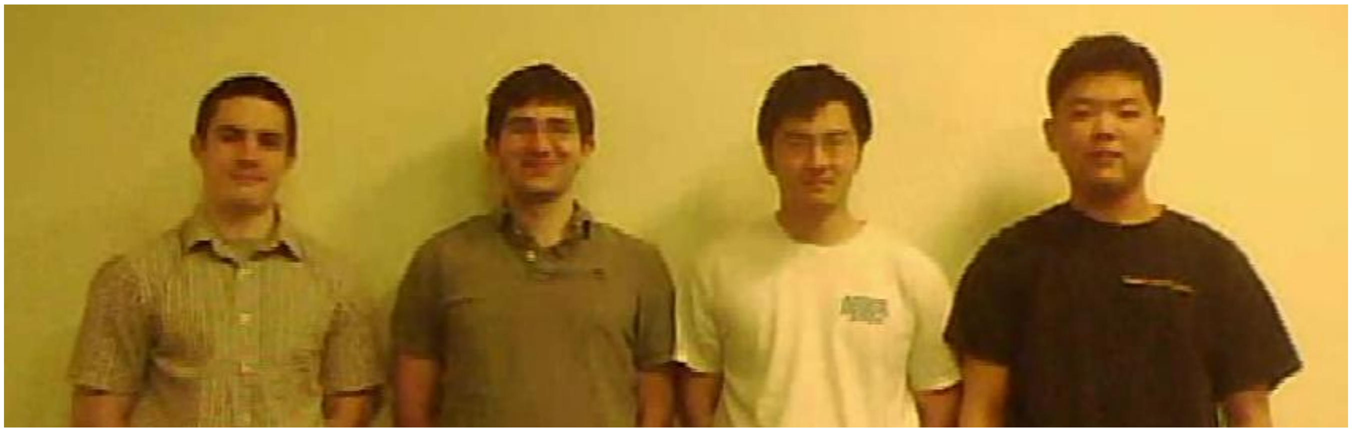
Week 6

- Consolidate all electronic materials to burn onto DVDs
- Submit: DVDs containing the complete project report, all electronic materials generated by the project, and the finalized application

Week 7

- Find and fill out CDR forms
- Print out and bind project report (2 copies)
- Create a portfolio of all materials submitted this term
- Work on updated personal statements
- Submit:
    - One CDR form per team member with personal information and abstract filled in
    - 2 bound hard copies of project report
    - 2 DVDs containing the final application
    - Documentation of all permission letters sent and received
    - All loaned material
    - Portfolio of materials submitted during the term
    - Updated personal statement from each team member

TEAM BIOGRAPHY



Left to right: Patrick Newell (CS ’10), Steven Shidlovsky (CS ’10), Justin Liu (CS / Biology ’10), Hyungjoon Kim (Biochemistry ’10),