A Puzzle Strategy Game Using Behavior Trees

A Major Qualifying Project
Submitted to the Faculty of
Worcester Polytechnic Institute
in partial fulfillment of the requirements for the
Degree in Bachelor of Science

by

_____
Zackery Mason

_____
Sean MacEachern

_____
Christopher Gillis

Date: 4/28/15

Project Advisor:

Professor Charles Rich

Abstract

Kill the King is a game created to determine the viability of middleware for building interesting artificial intelligence (AI). The game was developed using the Unity game engine in tandem with the Unity AI plugin RAIN. In *Kill the King*, the player takes on the role of an assassin tasked with killing a corrupt king primarily through manipulation of both the king and his knights. Creating complex behaviors for the game was fairly quick and easy using RAIN, but the most challenging aspect of the project was conveying the behaviors to the player.

Table of Contents

Table of Figures

1 Introduction

1.1 High Level Statement

Kill the King is a 3D top-down puzzle game with real-time strategy (RTS) elements and an emphasis on interesting behavior mechanics. Each level of the game consists of three major elements: the units under the king's command, the player's own units, and neutral or hostile units without allegiance to either side. The goal of the game is to plan the assassination of a king using the specialized units given to the player at the start of a level and then set that plan into motion before the king reaches the end. What sets *Kill the King* apart from other similar games in the genre is that the king's units have a set of behaviors associated with them based on confrontations provided by the player or external forces.

***Figure 1**: Kill The King title screen*

The team for *Kill the King* consists of three programmers. The original purpose of the MQP was to explore the benefits of using middleware to develop interesting AI mechanics within the framework of Unity (Unity Technologies) using a plugin known as RAIN (Rival{Theory}). Given the team consisted of only programmers, the work was split communally among the members of the group.

1.2 Background

The original intent of the project was to explore the capabilities of AI middleware towards creating interesting game mechanics. The most used feature of RAIN was its ability to create behavior trees: a kind of hierarchical decision making AI that is further described in section 4.3.4. The motivation was to create a technical piece with similarities comparable to games like the Sims (Electronic Arts) or Prom Week (McCoy et. al. 2012) in order to both provide a strong demonstration of the middleware's capabilities as well as the foundations for an interesting game. Each of these games have characters with multiple attributes that affect how they behave and react to things that occur around them, which appear to a human player as being their "personality". Through various actions, players could manipulate those attributes to get characters act in ways that helped to achieve some end result; the team wanted to use that concept in conjunction with what the middleware provided.

***Figure 2**: Age of Empires (Microsoft Studios) camera reference image*

Once the team was exposed to the underlying functionality of the selected middleware, namely RAIN, several brainstorming sessions were held to discuss the different potential features and concepts to better establish the kind of game that would be most suitable to showcase the team's abilities and the power of RAIN. The topics of political intrigue and espionage were a common theme where the entities therein would make use of behavior trees in

the form of interesting dialogue and actions based on the choices of the player. When confronted with how best to display these sorts of behaviors to the player in interesting ways, the team decided on the idea of a top-down interface where the player had an omnipotent view similar to what one would see in games from the RTS genre such as *StarCraft* (Blizzard Entertainment) or *Age of Empires* (Microsoft Studios). An example of this kind of camera can be seen in Figure 2. The top-down aspect also played to the team's strengths as, without artists, it would allow the team to work in two dimensions and limit the focus on highly detailed assets.

When it came to how the player would interact with the game world, the team debated whether or not the player should have a physical avatar in the game. The original concept of espionage developed into a narrative where the player was an assassin with the goal of killing an individual of high authority which lent itself to the player being forced to make interesting choices in order to get the assassin close enough to pull off the caper. From there, the concept arose of manipulating entities to do as the player asked. Was the assassin able to hire thugs to do his work for him? Was he able to command units to move in certain places to create a distraction? The longer the team discussed the importance and agency of the player within the frame of the world, the more the overall goal of the game seemed to move towards having the player solve a puzzle. Ultimately, the player assassin was removed in exchange for omnipotent commanding of a given swath of specialized units.

With the focus of the team shifting towards centering the game on its puzzle aspects, it was determined that the player could command units in a manner similar to traditional RTS games but with more limited controls to maintain the puzzle-like feel the team wanted for the levels. It was at this point that the team decided the main mechanics of the game would be built around making the setup of assassination similar to a blueprint planning stage. It was agreed that the player, given that the game was now a puzzle, should have control over time and be able to replan their strategies towards assassinating the high-ranking individual if things didn't go as intended.

Procedurally generated levels were briefly discussed but ultimately discarded as the time and effort into developing such methods was beyond what the team thought to be achievable in

the time frame of the project. Tying in the idea of specialized units, the game was designed in such a way that the levels could quickly be built around the various units and explore concepts within them. As the behavior trees created before the project to get the team up to speed were modular enough to replicate and use anywhere, the team was able to fairly quickly craft levels with only minor script writing between them.

While other features were planned to be included into the game, the culmination of the team's efforts still required some planned elements to be left out of the final build. The biggest challenge that came with having an AI-focused game was not the technical problem of creating the behaviors, but the largely artistic problem of developing a good way to represent those behavior to the player. While the lack of an artist on the team made the resolution to this challenge difficult, the biggest idea to condense down was creating a gameplay that fit how the team felt the game should been experienced. This lead to one of the most important lessons learned during this project; the importance of developing complex AI in a game should never exceed the ability of reliably displaying the desired information or behaviors to the player. The final version of the user interface was a culmination of many playtests across multiple iterations that were tweaked and improved upon and to create what is now the ultimate look of the game itself.

1.3 Overview of Document

This report overlooks the team's design rationale throughout the course of the project. In the Gameplay section, the game's mechanics and underlying structure are discussed to give context for the following sections of the document. Following that, the Design section runs through most of the core decisions the team made about the game as well as the reasoning behind those choices. Leading on from there, the Implementation section delves into how the game works on a programming level and describes the patterns and behaviors chosen by the team to fit the final version of the game. Finally, the Post-mortem section concludes the paper with the team's final thoughts and self-analysis regarding the project in its entirety.

2 Gameplay

Kill the King is a combination of the RTS and tower defense genres with strategy elements centered around an attribute system similar to The Sims (Electronic Arts). Each level is also designed with a specific solution in mind that adds puzzle-like elements to the game. The first mockups included references to games like *StarCraft* (Blizzard Entertainment) and *Age of Empires* (Microsoft Studios) in terms of both their visual styles and their control scheme. The behaviors of the units drew inspiration from both The Sims (Electronic Arts) and the team's original reference, Prom Night (McCoy et. al.). The game was designed to have a familiar control scheme and theme to RTS games but with a distinctly different experience goal for the player. The overview of the game is shown in the campaign screen, where the player can see which levels they have completed and those that are available to attempt. Completing a level unlocks any others connected to it, and the player's goal is to complete everything leading up to the final battle clearly marked on the map.

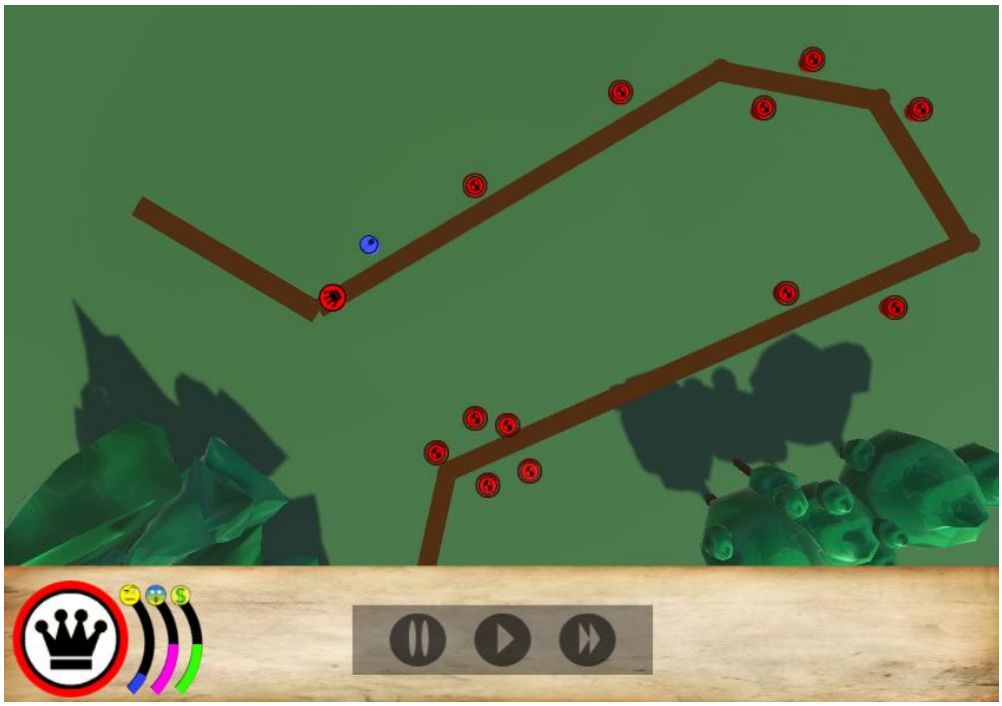

***Figure 3**: Example Level*

Each level takes place from a top-down perspective. Most of the levels take place on one flat ground-plane with a few exceptions. The objective of each level is to stop an enemy commander from completing his objectives. This usually means killing him before he reaches the end of the level. The entire level is visible from the start, and the player can move the camera using the WASD or arrow keys as well as use the mousewheel to scroll in and out for a better view. As can be seen in Figure 3, the HUD is present at the bottom of the screen and displays important information related to the level including the commander's current attributes as well as a panel of buttons to pause or speed up the gameplay. Pausing the game with the space bar or pause button allows the player to see the whole map and outlines important entities like units or the king's path to allow the player to more easily plan their assassination.

The world of *Kill the King* has a wide variety of allies and enemies that can appear in each level. Every unit takes up space, moves in two dimensions, and interacts with other units in specific ways, sometimes with very complex behavior. The game moves in real time, with the ability to pause and fast forward as mentioned above, and most of the strategy involves learning what commands should be issued to units as well as the timing and placement of those commands. The game is mostly deterministic with a few minor exceptions.

2.1 Units

***Figure 4**: King Icon*

King: The king is the ultimate enemy, and the end goal of almost every level is to kill him before he completes his objectives and reaches his escape point. He starts each level with a

clear path built before him and will proceed down it unless interrupted. Interruptions can happen in many ways: seeing a chest, meeting an enemy, or activating a hidden trap, among other events the player may or may not have control over. When the problem is dealt with, he always continues from where he stopped. A walking juggernaut, the king is very strong compared to most of the player's units and can fend for himself, but he travels with an entourage of knights at his side just in case. In most cases, the player can't win by simply having their units engage the king's forces in combat but instead can manipulate the king's behavior by modifying his three attributes: Fear, Paranoia and Greed. Fear represents how much a king believes he is in danger from the environment and enemies; decreasing his fear will make the king charge suicidally into battles while high fear will result in him forcing a knight to walk in front of him to check for traps. Paranoia is a measure of how much he trusts others; low paranoia allows disguised assassins to infiltrate his ranks to kill him while high paranoia will make him fight his own men if he believes them to be disloyal. Greed is a measure of the king's desire for gold; high greed will make him turn over every rock in search of riches, while low greed will make him ignore even the most obvious aid being presented to him. Levels are usually tailored around manipulating the king to act in certain ways by giving the player units to change these attributes, but the king acts consistently between all levels and will perform the same actions every time given the same situation.

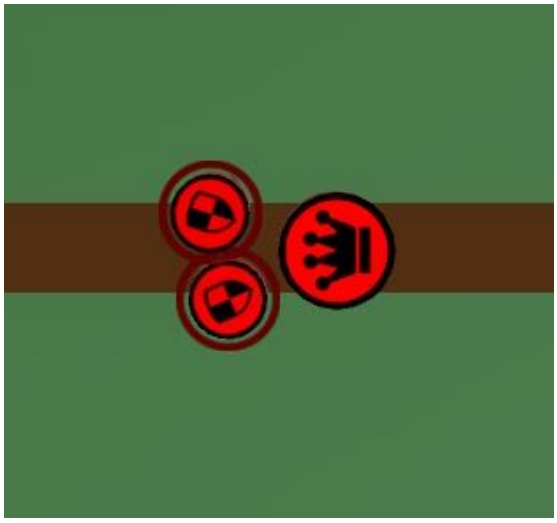

***Figure 5**: Knights following the King*

Knight: What's a king without an army? Well paid and well-armed, the knight is the king's generic lackey, following him without question and doing whatever deeds they're

commanded. The knights are much better trained than the player's ragtag group of rebels and are also more than a match for them. They follow the king on his journey the moment he sees and calls upon them as can be seen in Figure 5. The knights may still patrol without the king, but most of the time they will only wait at their assigned post until the king calls upon them. While they're possible to defeat in combat, there are better ways of dealing with them. Knights also have attributes similar to the king but functionally different: Fear, Hunger and Loyalty. Knights have Fear just like the king does; at low fear, they'll bravely charge into any danger and peek their head in anything interesting, and at high fear, they'll ignore orders and flee from battle or even off the map. A knight's Hunger starts at nothing and slowly rises over time but doesn't do anything until he's reached his limit, after which he takes damage and becomes disloyal due to maltreatment. Loyalty is a knight's devotion to his king; knights are very loyal by default, will follow any order unconditionally, and will gladly fight to the death, but at low loyalty, a knight is liable to ignore the king, perform rebellious acts for his own benefit, or even become a turncoat and join the rebels. The king will usually have enough knights around him to deter any physical threat, so it's essential to learn how to work against the enemy by using means other than direct combat.

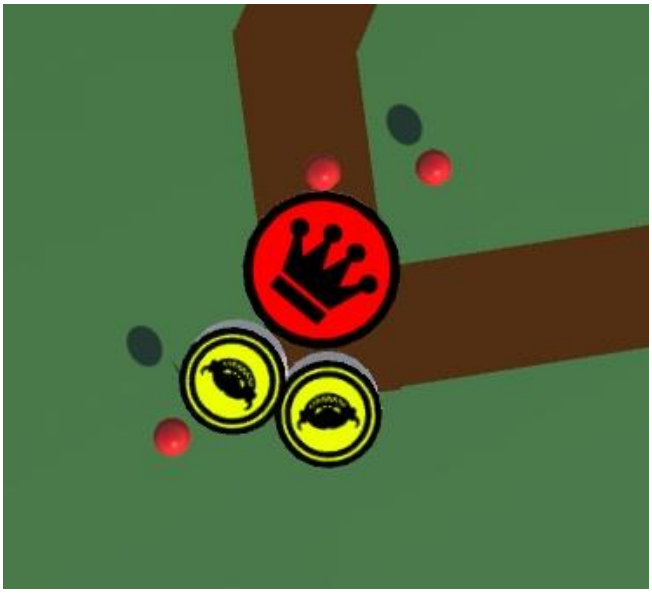

***Figure 6**: King fighting Creeps*

Creeps: The world is of an older age, and the beasts and monsters of the wilderness have not been fully culled. A variety of other enemies are spread throughout the world, ranging from

common goblins, explosive ghouls, vicious bears, spooky ghosts and more. These creatures are a hostile third party that will fight whatever comes close, but rarely have any goals in mind and will wander aimlessly until disturbed. Their uses and threat range widely between levels and, while they can be seen fighting the King in Figure 6, they're just as likely to be a threat to the player's units as well. There's no reasoning with them as is possible with the king and his knights either, being mindless fighting machines.

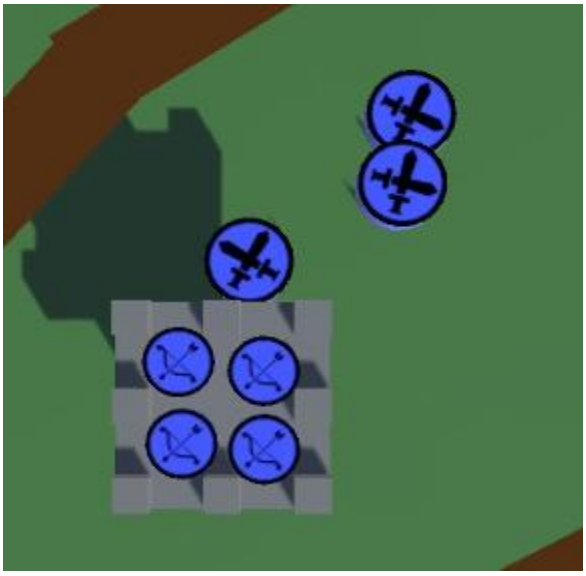

***Figure 7**: Thugs surrounding Archer tower*

Thugs and Archers: The player has a variety of rebels on their side as well, each geared towards a common tactic. For thugs and archers, this tactic is fighting, with thugs being the frontline tanks and archers being the ranged assault force. The rebels almost always have a few of these guys on the field ready to fight against the powers that be, but those powers are often the victor in outright combat. These fighters work best together in groups where they can gang up on weakened or solitary enemies and are one of the fastest units that can be quickly repositioned around the map for ambush. Archers can additionally be put into archer towers as they are seen in Figure 7; if a level happens to have an unoccupied tower, any nearby archers can be garrisoned inside to become invulnerable with extended range at the cost of not being able to move.

***Figure 8**: A Knight checks a Gold chest*

Chesters: Combat often isn't the key to victory; sometimes a little trickery goes a long way. Chesters are units that place the chests in traditional RPGs, only this time they're working with the player, and the prizes inside aren't always the fancy loot that the player has come to expect. While utterly useless in combat, Chesters start out with a single chest that contains one of four traps inside, and he can be ordered to place it anywhere on the map. The king and his knights will stop for any chest they see on their way, except when the king has no greed at all. If the king is greedy enough, he will send a knight to investigate any chests he sees, but he may also check a chest himself if the player manages to manipulate his greed to be high enough. On opening the chest, the one checking it is influenced by whatever he finds; the chest is emptied; and everyone continues on their way. While Chesters cannot place more than a single chest down, they can be moved near exhausted chests to refill them allowing a single chest to be used multiple times if put near a looping path. There are a variety of trapped chests available to the player: Vomit traps that poison guards with uncontrollable vomit that makes hunger rise and the King's greed and health drop; Jacks-in-the-box that terrify anyone near their reveal by raising fear and paranoia and lowering greed; Food bribes signed for by the rebels will make knights disloyal based on how hungry they are but just fill the kings stomach with no effect; Regular chests filled with gold as seen in Figure 8 that elicit the typical response of the opener by drastically raising his greed while lowering paranoia and fear. Too many bad chests in a row will lower the king's greed to the point that he ignores every other chest he sees, while good chests alone will only benefit the king and his knights. Players will need to learn the best situations to

use and place each type of chest while also playing a careful balancing game to ensure the king continues to open the chests at all.

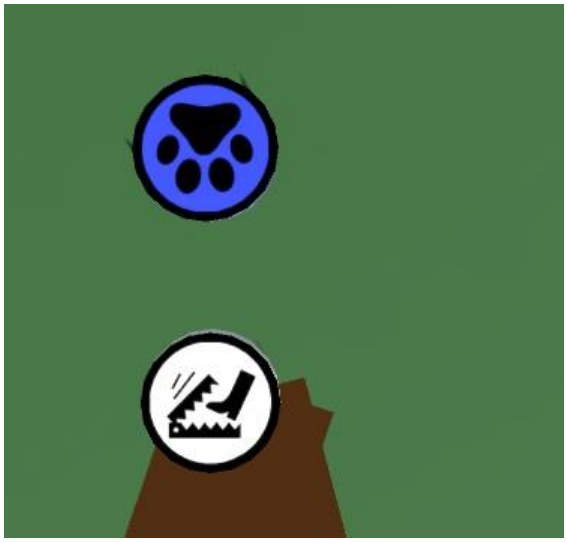

***Figure 9**: A Trapper places a Spike trap*

Trappers: Trappers are very similar to Chesters, in that they have no use in combat but have a one-time-use trap they can place down and rearm. An example of a trapper placing a trap can be seen above in Figure 9. Unlike chests, traps aren't visible to the king and knights and instead activate automatically if any enemy comes close enough, meaning they have to be placed directly where someone is going to walk to be effective. They have a much more potent effect than chests do, but they're just as situational, to the point that giving the king some gold might be more useful than preparing to impale him with a trap on some levels. Instead of greed, traps are used to manipulate the king's and knight's fear. Though even with high fear, the king can't just ignore traps the same way he does chests (even if he wants to). The available traps are spike traps and snare traps. Spike traps cause an explosion of vicious spikes when activated, damaging anyone nearby and pushing them back while causing a high amount of fear. Snare traps will rope in anyone that triggers them, rooting them in place for a long duration that can leave them isolated for a strategic advantage but does no damage and causes a very minor increase in fear.

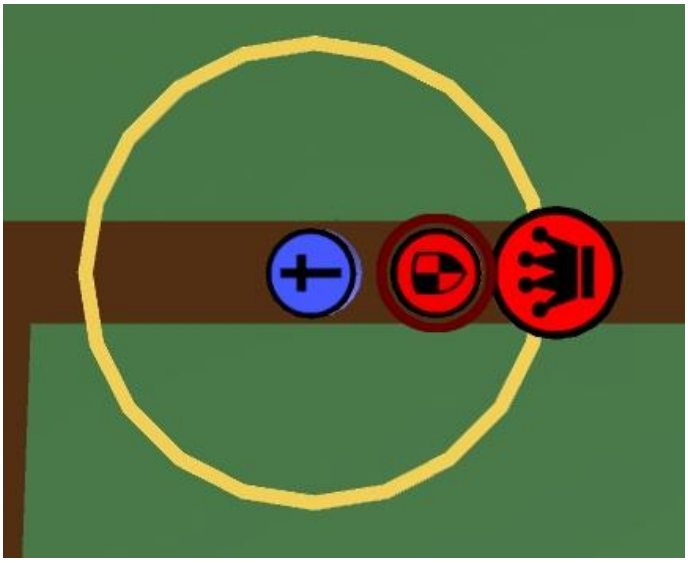

***Figure 10**: A Priest follows the King and his Knight*

Priests: Unlike the more violent classes, Priests can actually heal units they come across by performing a wide area AOE (area of effect) heal every second that they see someone close by who's injured. They can be a huge boost to the player's offensive units but are also a possible tool to use for converting knights; any damage healed reduces their loyalty by the same amount, meaning in prolonged battles priests can keep everyone alive until either the enemy flees or joins the player's side. Being just as weak as Chesters and Trappers, they have no use in a fight beyond healing and will fall quickly from any damage. However, they have another advantage of being a non-combatant; they follow the player's commands like any other unit, but the king and knights won't attack them on sight, allowing the player to move them into enemy lines. They'll follow the king and his knights around just like any other follower as seen in Figure 10 while healing anyone along the way They do so in sight range of the king, however, which will raise his paranoia. If the king gets paranoid enough he'll get suspicious of the units not under his direct command and attack them on sight like any normal rebel.

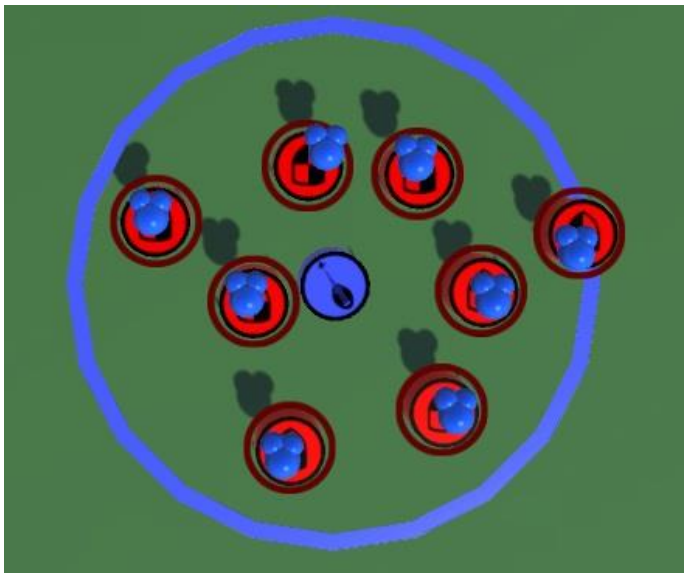

***Figure 11**: A Bard converts a group of Knights*

Bards: Very similar to priests, bards are pacifists who only play songs that cause mental effects in a large AOE. They don't follow people around like priests do, instead preferring to stay in the first place they are directed to and sing their song for as long as they're physically able. Bards have two songs the player can direct them to play: an inspiring tune about how glorious the rebellion is that heals rebels and makes knights disloyal that can be seen in Figure 11 and a haunting tune that terrifies all enemies that hear it. Just like the priest, these mainly affect knights, but if the king hears it, his paranoia will increase to the point he attacks bards on sight as well.

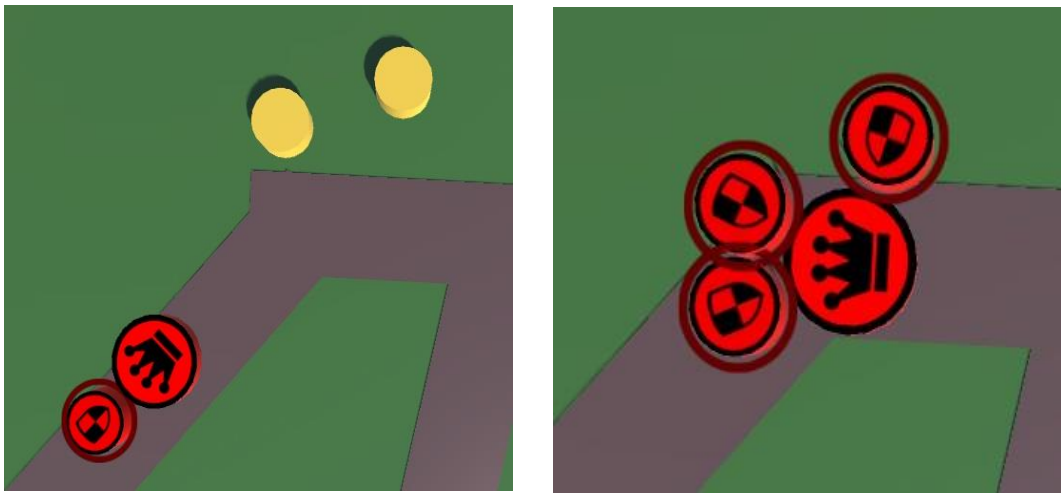

***Figure 12**: Left: A King spots a group of Neutral Units; Right: The neutral units join the King's cause*

Neutrals: Between the fence on all issues, neutrals are the general peasantry that haven't yet picked a side but are very easily convinced. They have nothing to do with either side at the start of a level, but the first side to get a neutral unit within speaking range causes them to be converted to their cause, with monsters obviously exempt. The king gets another knight as seen in Figure 12, while the rebels get another knight or archer. This can turn the tide in anyone's favor depending on how many of them there are throughout the area, but the player must keep in mind to always make sure to command them once they join the player's cause as the enemy often has their sights on recruiting as well.

2.2 Strategies

All unit commands are handled via a pie menu showing the available actions that appears around a unit after the player has selected it. The player can direct a unit to a given location by first selecting an action for the unit to perform and then selecting a place on the map for the unit to go before performing it. For combat units, it is also possible to click and drag the mouse to select multiple units and right click to order all of those units to the selected location. This mechanic is similar to an RTS, but differs in the sense that after an order is given to a unit, they will largely act on their own. For example, the player is unable to micro-manage units that are already in combat to encourage the player to think of each level as more of a puzzle than a clever test of their reflexes.

There are many kinds of exploitable strategies based around how these units interact with each other, though the wide variety is usually limited depending the level. Knights follow their king once they first make contact with him and will remain at their respective stations along the road until he arrives. A group of thugs and archers, while typically no match in a fight against the king's guards, can pick off knights while they're alone with minimal damage, as archers can shoot down a solo target before he has time to reach them and reload before the next enemy arrives. The player is not able to order an ambush on a specific enemy unit, instead only issuing an order to attack a general area that an enemy unit may be in. The player must keep this in mind to ensure that their units don't get caught up fighting an undesired unit along the way.

Trappers can either deal massive unavoidable damage with their spike traps or temporarily stall an enemy with their snares. In most cases the damage is preferable, but snared enemies can be attacked individually as the king will not waste time waiting for a guard to free himself from a snare. After two spike traps, the king will become paranoid of running into additional traps and will send a knight to check the road ahead if he has one available. That knight will activate any other traps placed along the road but will also give the player a chance to pick off an isolated target that the king is depending on to keep the road safe; if the knight is eliminated quickly, another trap can be placed to finish off the king. In addition, the path of the king is visible in Strategy Mode, and if the player notices a loop it could be a good spot for a trap to be placed and rearmed repeatedly.

***Figure 13**: A Chester moving to rearm an opened Chest*

Chesters are the player's best tool for manipulating and taking advantage of the king's behavior, but it's also easy for them to make a single mistake and ruin the player's plans. Two traps or Jacks-In-The-Box are enough to make the king very fearful, while two gold or three food chests are enough to make him very greedy. The player should avoid using anything more than is necessary to bring the king to these extremes and should keep the initial positioning of the chests

in mind as it is necessary to rearm chests to get the most value out of them. It's possible to "juggle" the king's behavior by giving him a sequence of good or bad chests to make him switch from checking things haphazardly to opening a trap and getting hurt as can be seen in the figure above. This kind of "juggling" strategy can be seen in Figure 13 as the player has the Chester place down a Gold Chest, so the King will keep sending his Knight's to open Vomit Chests. The King gets greedy and bounces between the chests intended for him while getting greedier to the point of wanting to check a potentially hazardous trap himself. In the case the king only has a single knight following him, the player can combo traps by having a vomit trap early in the level to increase hunger over time before placing a food trap at the end to take advantage of the knight's hunger to convert him to the rebel cause. Chests affect the king's other attributes besides greed, so manipulating his attributes can make him a lot easier to deal with when there are other trappers, priests, or bards in the level.

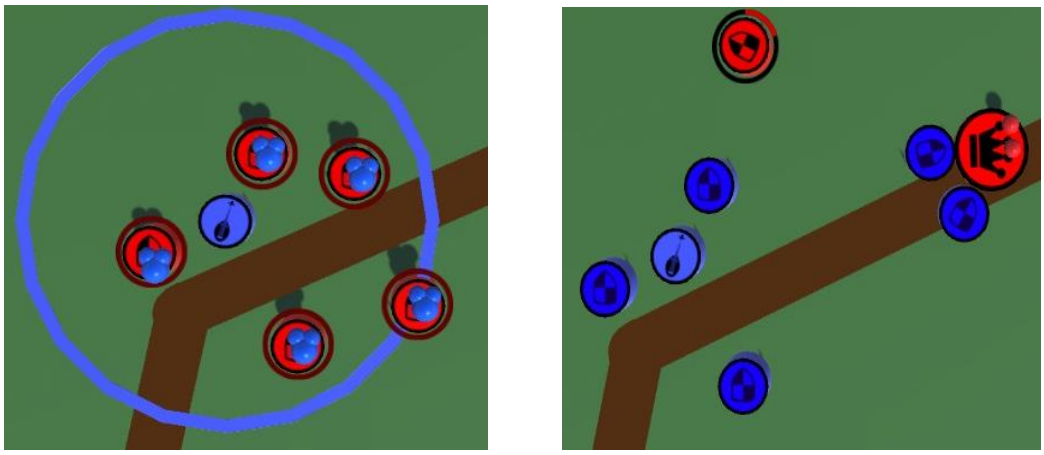

***Figure 14**: A Bard converts Knights to the rebel cause*

Bards and Priests are very similar and are one of the few ways the player can recruit enemy knights to their side. All their abilities affect both the king's and player's units in an area around them, but this also makes it harder to hide their actions from the king. For conversions, it's best to aim for large groups of immobile knights who are still waiting for the king to call upon them they won't react or run away without the king as seen in Figure 14, but they also start with maximum loyalty and take a while to convert. Priests will heal anyone in range, including the player's own combat units. A prolonged battle with a priest in the middle will often result in a conversion followed by a full heal. Bards have a choice between pure conversions or an

advantage in combat, but they require good positioning to be the most effective. The player is encouraged to center battles around these two units when they are available as they can greatly increase the effectiveness of the weaker rebel thugs and archers.

Quite a few levels have their own unique gameplay twists that are explained beforehand so that it won't take players by surprise. Creeps in patrol is a common theme, where it's best for the player to try and avoid fighting needless battles and save their strength for the king's troops, but there are times where leading creeps into knights or vice versa can cause a battle that leaves both sides weakened and vulnerable afterwards. The convertible neutral units are usually a race between allied troops and the king to acquire them; while players get only a thug or an archer, the king gets a much stronger knight, so stealing converts (or at least preventing them from being converted) is a high priority. When traps or chests are already on a map when it starts, it's usually a hint as to what the player should plan around doing in the first place. A few levels start out with a "cutscene" example of how new mechanics work before being given a chance to perform the same kind of actions themselves.

2.3 Level Summary

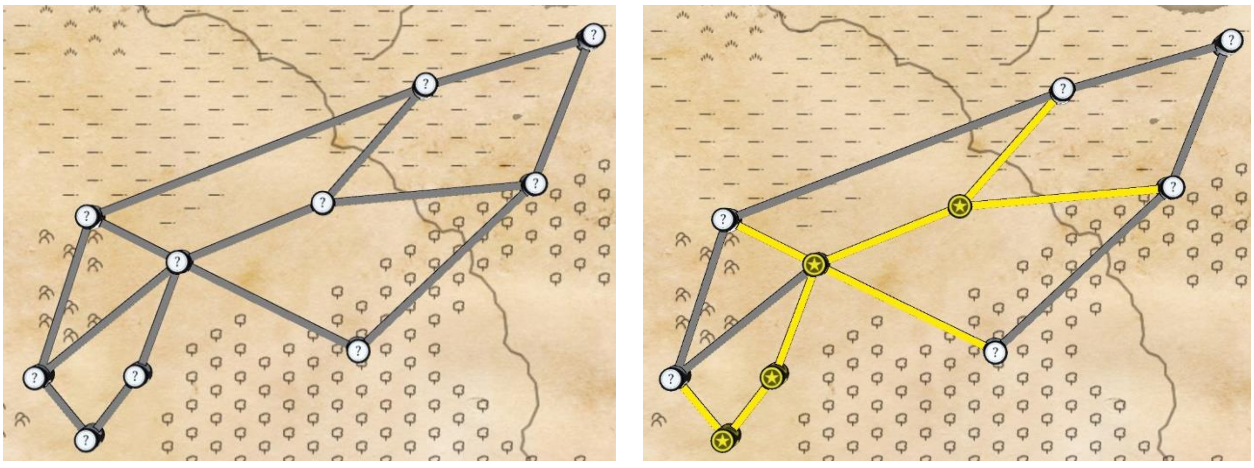

***Figure 15**: Left: The Campaign Map at the start; Right: A partially completed Map*

The campaign is a series of connected levels as seen on the left in Figure 15, with only one available for play from the start. When the player completes a level, connected ones become unlocked for playing, opening up the map as the player progresses into a branching tree-like

pattern that converges into one final ultimatum level. Each level introduces a new mechanic or expands upon the uses for a previously learned one, and the branching design is sectioned to focus on a certain area of gameplay. IE: one series of levels may focus on traps and chests, while another may focus on combat and unit management, so the player can progress down whichever series of levels they want for more in-depth levels based around those mechanics, or switch halfway through if one side becomes too difficult. In addition to just unlocking more levels, sometimes the reward is also directly affecting the unlocked levels themselves, such as adding more units for the player to control or opening up areas that were sealed before. The player's progress is saved and updated on the campaign screen, and previously completed levels can be replayed at will. Figure 15 on the right shows how the map is updated as the player completes more levels, with every highlighted area being available to select and play.

3 Design

3.1 Constraints

3.1.1 Software Constraints

The main goal of this project was to try to make a game with interesting behaviors using a pre-existing middleware to demonstrate that game studios do not always need to build their own AI engines to create interesting characters or enemies. The middleware in questions was chosen to be the RAIN (Rival{Theory}) plugin for Unity that allows its users to easily create behavior trees. RAIN is currently only supported in Unity, so the choice of engine for this project was non-debatable. Likewise, the team's initial ideas for games was built around the idea that they would need to be concepts that would benefit from having complex AI interactions both between characters and between characters and the player.

RAIN is a powerful AI tool that supports its own perception system as well as a way to generate navigation meshes around which characters using a RAIN AI component can base their movement. The biggest problem with RAIN was getting over the initial learning curve required to use it. While Unity has a large community that is ready and willing to answer almost every

problem a developer can encounter, the community around RAIN is much smaller, and it's very possible to run into a problem that no one else has had. Questions about RAIN are answered primarily through a forum on the Rival{Theory} website, but the team behind it could sometimes take days to answer a question if they answered it at all. To get over this, the team worked on smaller, personal projects before beginning this project to develop a working understanding of RAIN and how it uses behavior trees.

As stated before, Unity has a large community around it to answer most questions a developer could have. The main concern with Unity for this team was its unfamiliar way of handling software engineering and object orientation. Unity is great for rapid prototyping which allowed the team to quickly arrive at the core mechanics of the game, but the way the engine handles scripting through components led to some trouble when the team tried to make a robust, maintainable framework for the game. Ultimately, the team appreciated the ease of use Unity provided but felt constrained by Unity's scripting system.

3.2 Art

When the members of the team signed onto the project, there was a problem with the structure of the team that became immediately apparent; all three of the members were either IMGD-tech or double majors with Computer Science with little to no background in art. During prototyping, the team used the default primitive shapes in Unity (such as planes, cylinders, cubes, etc.) to represent the characters with the hope that an artist would be able to be brought in for a term to create assets for the game. However, finding an artist was more difficult than expected, and the team decided to embrace the limitation and build the assets with a simple, iconographic style. Unity primitives (primarily cylinders) are used to represent the units with symbols on the top faces to allow the player to differentiate between them.

***Figure 16**: The King icon*

The iconographic style that can be seen in Figure 16 was found appealing for two main reasons. Choosing to go with icons rather than manufactured assets allowed the team to focus more on the gameplay and complex behaviors rather than having to stress over the look of the game, which during early stages of the project was the primary focus at the time. Not only did icons afford extra time for deeper development, but the style itself led the project towards a universally appealing theme as seen with the minimalistic games such as 2048 (Cirulli 2014) and other sleekly designed puzzle games that have been released in recent years. Later in the project, it would help further emphasize the game as a puzzle rather than a real-time strategy game while allowing a broader audience to interface with the game's mechanics. However, in other respects, maintaining the art style would be a challenge. By focusing the main theme of the game around icons, the project's amount of text was limited to only a few sections namely being the tooltips and the menu. The rest, in order to maintain consistency, had to be icon-centric otherwise it would detract from the art style that was originally intended.

***Figure 17**: Left: A bland, early level; Right: A level with trees as background*

Towards the end of the project, the team realized that many of the levels felt bland as can be seen on the left in Figure 17 and could benefit from having background decoration to detract from the monotony of each level. Some assets, such as simple castles, were made in Unity using a combination of primitive objects. The team had a desire to find additional 3D models to represent environmental objects but did not want to find highly detailed assets that would stand out next to the cylindrical units. While looking on the Unity Asset Store, the team found an asset pack with very simple environmental models such as trees and rocks. Even more fortunately, the models had a low resolution and fit very naturally next to the units and other assets already in the game as evidenced on the right of Figure 17.

3.3 Visual Feedback of AI Actions

At the outset of the project, the team was fairly confident that the biggest challenge of creating the game was going to be the implementation of the complex behaviors of the king and his knights. As the project went on, however, the team found that using RAIN (Rival{Theory}) had allowed the behaviors to be prototyped and refined very quickly. It was during the initial playtests of the game that the true challenge of creating a game with interesting AI presented itself. While the members of the team knew what interactions were taking place, the playtesters had no indication of what was going on. The team had to quickly redirect their efforts away from creating more unit types and furthering the features of the game to ensure that there was a

meaningful and clear method of displaying what each unit was thinking in their behavior tree at any given time.

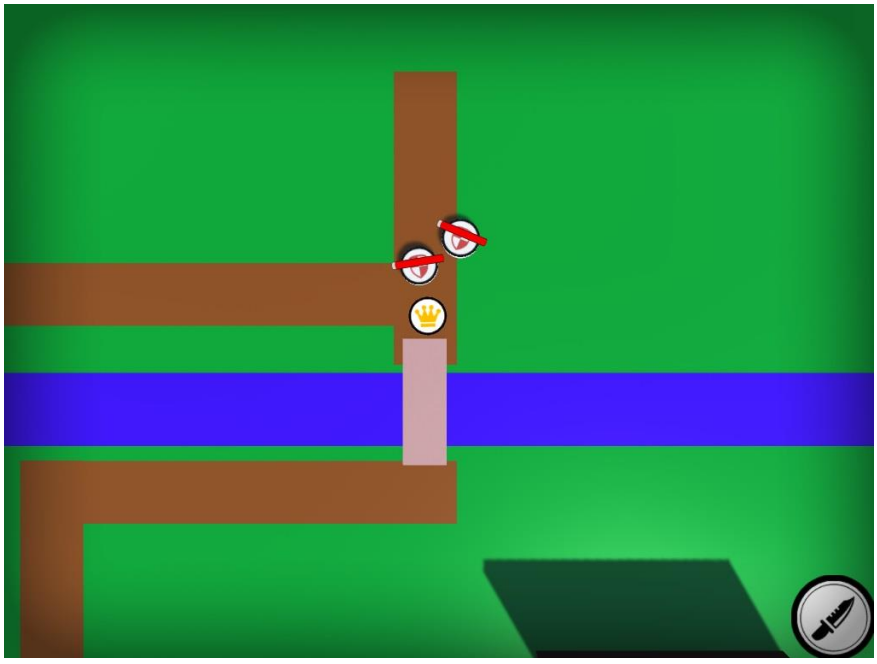

***Figure 18**: The first iteration of the UI*

The first iteration of the user display that can be seen above in Figure 18 consisted only of a set of bars in a section of the screen displaying the King's attributes and health as well as a button on screen that would allow the player to spawn an Assassin token outside of a specified range of the King. This design was favorable at first because it presented a simple solution to the problem of displaying information. While this worked for the team as a starting implementation, early playtesters showed confusion as to what they represented when playing through pre-alpha stages of the game. This led the team to draw further upon alternative ways of showcasing the information and to contemplate how the player might receive or interpret what they were given.

***Figure 19**: The updated, circular version of the UI*

From the early playtests, the next draft of the user interface took the form of arced bars around the units themselves as seen in Figure 19. The main reasoning behind this approach was that it allowed the player to see all the information contained within each unit without the need for a menu or section of the screen to be partitioned off. For the King, the team figured all the attributes could be displayed but be given a particular length around the King in terms of how important they there were which resulted in the health as the most important and visible arc around the king with his attributes each taking a small segment with equal length.

Guards were also included amongst the units that were given arcs to show their intermittent changes to their behavior values as knowing how an enemy unit would react to a command was something that the team considered an important features to visibly broadcast to the player. However, upon playtesting the new addition, players still had trouble determining or even seeing the bars themselves at certain zoom levels. Given that the bars didn't scale, players were not able to tell how to interpret the changes in values among each of the units and thus played the game without demonstrating knowledge of the features attempted to be showcased.

It was clear after the user interface's second iteration that the player needed to be shown directly what the units were thinking, with primary focus on the king, in order to better understand the causes and effects of taking certain actions. In order to solve this problem, the team drew inspiration from the Sims (Electronic Arts). In the Sims, as values for a person's well-being increase or decrease, an emoticon appears once a sim has reached a certain threshold regarding the current state of the person in question. Implementing emoticons into the game would allow for the appearance of bars to be minimized in exchange for visualizing the cause-and-effect style of gameplay that the team desired to achieve. From this, the thought bubble system was created as a way of using the emoticon style of the Sims to show the different states and changes within the King's as well as any other potential unit's behavior. The only con to using such a system was the potential risk of obscuring the top-down style of gameplay. Given the needs of the project at the time, however, the team decided to move forward with the idea.

***Figure 20**: An example of an icon sentence*

Since the game was mostly iconographic, the idea for developing an iconic syntax or statement within the thought bubbles came to light. The concept would allow for the King to show not only his response to changes in behavior but also to changes in his environment such as if he spotted a potential enemy or was signaling for his battalion of guards to take care of a task for him. One of the icon sentences used in the game can be seen above in Figure 20. These sentences are broken down into three parts: the actor, the action, and the receiver. The example above shows that the King is planning to attack an archer that has come into view with the actor being the King, the action being fighting, and the receiver being the archer that he sees.

Moving along with the idea, the next part of the design was whether to include guards into the mix of units that could display thought bubbles. Upon realizing the risk of cluttering the screen with too many thought bubbles given that several levels with numerous amounts of guards

were planned, the team opted to only have the King display a thought bubble. Being able to predict what the King would do in response to traps or ambushes set by the player was something to be considered highly valuable leading to developing the system to primarily focus on the King's internal mechanisms.

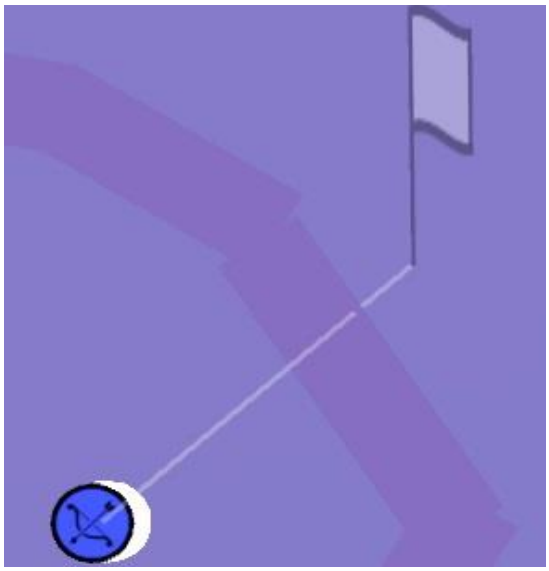

***Figure 21**: An example of a flag appearing at an Archer's set destination*

Following the ideals of the second design iteration, another element of the user interface was added to help clarify a unit's movement. The problem with the original design involved the lack of visual feedback for when a player selected a unit, gave them an order, and then sent them to a specified location to complete a given action. For inspiration on how to reach the solution within the frame of *Kill the King*, the classic *StarCraft* (Blizzard Entertainment) series was notated as being a prime example to follow where a unit's specified location upon command was given in the form of a marker. For *Kill the King*, this came in the form of a flag with a line connecting the unit to the destination as can be seen in Figure 21. These flags would gradually fade over time allowing the player just enough time to know where the unit was headed while avoiding too many lines on the screen cluttering the visibility of gameplay.

Once the thought bubble system was realized, the team's attention drew back to the original issue of whether or not curved bars were necessary to display aspects of the king to the player. After a bit of discussion, health remained one of the elements that was favored to keep on

screen at all times while the personality aspects of the guards and King could be abstracted into some other form. However, what this new form would take generated a series of discussions that would last well into the third term of the project. The next set of problems to tackle involved when the player confronted the user interface of the game. Some elements would overlap or interfere with the readability of the thought bubbles alongside the issue of being able to discern where the player's controllable units were in the level. In order to solve both tasks, the following conditions needed to have been met: The king's attributes would have to be constantly displayed and the indication of other units in the level would have to be presented in such a way that didn't overlap direct gameplay. To find an answer for the first dilemma, the standard real-time strategy user interface was looked upon. By sectioning a part of the screen for information relevant to the player, the issue of having a place to keep the King's attributes on constant display found resolution.

***Figure 22**: The final global HUD*

Not only would it solve the placement of the King's attributes, but it would allow for additional functionality from other parts of the game such as the play and pause buttons to be in an easily accessible location rather than be hidden within the pause menu or in a corner of the screen. The second dilemma was solved by looking back to early space flight games where the positions of enemies came in the form of markers that hovered along the edges of the screen. A similar construct was used for units controlled by the player except the chosen implementation not only showed what units were available but also allowed the player to be able to focus the camera on that particular unit with a simple click of the respective marker. The final version of the UI that incorporates all of these features can be seen in Figure 22 above. Handling the problem of displaying controllable unit locations to the player allowed better intuition for what strategies the level emphasized while also placing less impact on providing a miniature map for the player to reference as this new functionality allowed quick maneuverability about the playable landscape.

3.4 Design Choices

3.4.1 Initial game design

The first concept for the game was to have the player fight against a merchant king who would send waves of enemies to fight the player in an RTS style game. This king would react to the player's fighting tactics by sending units to counter the player's current army layout. While this idea was refined throughout the development process, the core theme of the game was focused around having an enemy that would react intelligently to the player's actions.

3.4.2 Unit design

Prior to the start of the year, the team worked on individual personal projects to develop an understanding of RAIN (Rival{Theory}) and how to create behavior trees using it with the concept of intelligent AI that reacted differently depending on its mental state being the end goal. The primary choice when designing units was whether the king would be the only intelligent entity or if his knights would be able to make decisions beyond what the king ordered them to do. The initial idea was to have the king command his units from some secure, off-screen location and change his behavior depending on how many of his guards completed their objectives and returned safely. This kind of hive mind AI didn't directly interact with the player and would merely adjust his parameters based on hearsay from the troops. Groups could be intimidated into reporting imminent danger, bribed into falsely saying quests were completed, or killed so a king expecting them on time would get worried. The only direct confrontation would be at the end of the level, where the king would be off guard or defenseless through whatever occurred during the level, and the player could kill him with their army. The main worry was that the player would feel disconnected from the action in the game, as a protected hive mind makes one feel like they're only dealing with the middleman for the whole ordeal. Because of this, it was decided that the king would take a more active role in the game.

The next layout was a connection of towns, barracks and guard posts the king would travel between while aiming for a goal destination. All places of interest would be connected and the king would change his route based on the mental state and status of his army such as if the king wanted to collect taxes for more money or find an inn to rest his units after a fight. These

places would have region specific units within that had their own mental states as well, and while they would default to being loyal to the king, they would be able to be bribed, threatened, or otherwise influenced to the player's side. For example, towns would have a mayor who would usually pay taxes upon the king's arrival but, if coerced to the player's side, would instead sets up an ambush. The king would respond in turn by adjusting his own mental state to remember such occurrences, likely not trusting the next encampment. After some mockups for how the movement and repathing would take place it was decided this plan would be too difficult to display in a fair way. The aim was for shorter levels at this point, and the thresholds prototyped for the king to switch paths all seemed a bit too spread apart; most levels would only have one or two chances for a major route change, and those changes would have to be telegraphed far in advance for to give the player time to change their own strategy. On top of that, the range of ways for the player to influence these areas to their benefit was either too simple (e.g. move allied units into those areas to claim it, hire that place in a point/click fashion) or too non modular (e.g. complete a subquest to convince the area to join the rebels, have the king perform certain actions that area disagrees with) to make villages something desirable to implement in the majority of the game. Many of these concepts would be kept in the final version but on a lower scale.

The focus of the project then moved towards the core concept that this project wanted to create: an intelligent, adapting king with complex interactions between both its own followers and the player. Initially, all enemy units had a few attributes to conceptualize their mental state: health, hunger, fear, etc. Player units could affect these attributes which in turn would cause enemy units to change their behavior at key moments. As an example, knights would flee from battle at high fear and would mutiny against the king at high disloyalty. This design provided a more immediate sense of agency to the player while still focusing on highlighting the complex interactions between units.

Playtesting among the team helped to refine the ideas around unit personality attributes. Initially, the units under the player's control also had an AI of their own, but this proved to be more annoying than fun, as it took away too much agency from the player. The initial plan for enemy units was to have multiple different classes such as scouts, cavalry, and healers that

would each have different weaknesses that the player could exploit to manipulate them. This was a difficult concept to convey, however, and was riddled with the same problems as the town's idea. As such, this was also scrapped in favor of a more consistent, single knight unit. In the end, it was decided that the best way to highlight the complex behaviors was to focus on only the behavior trees of the king and his single type of knight. The player would be able to learn more about how the king would react under certain stimuli and would develop more of an understanding and appreciation for the AI.

After the team had arrived at what units would have personality affecting attributes, the discussion moved to what those attributes should be and what behaviors they would initiate. The team decided that having dozens of attributes similar to the entities in The Sims (Electronic Arts) and Prom Night (McCoy et. al. 2012) would be too much information for any player to reasonably learn about with the limited UI. Because of this, the team limited the scope to a few key attributes: Health, Hunger, Fear, Loyalty, Paranoia and Greed. At first all the enemies had each of these statistics, but this caused some of the potential behaviors to be counterintuitive such as a King being disloyal to himself. The King then had his Loyalty stat switched Greed and his Hunger stat switched with Paranoia. This helped to emphasize the King's and Knight's different roles in the game and gave players a way to imagine them as distinct entities when formulating their plans.

3.4.3 Commanding Units

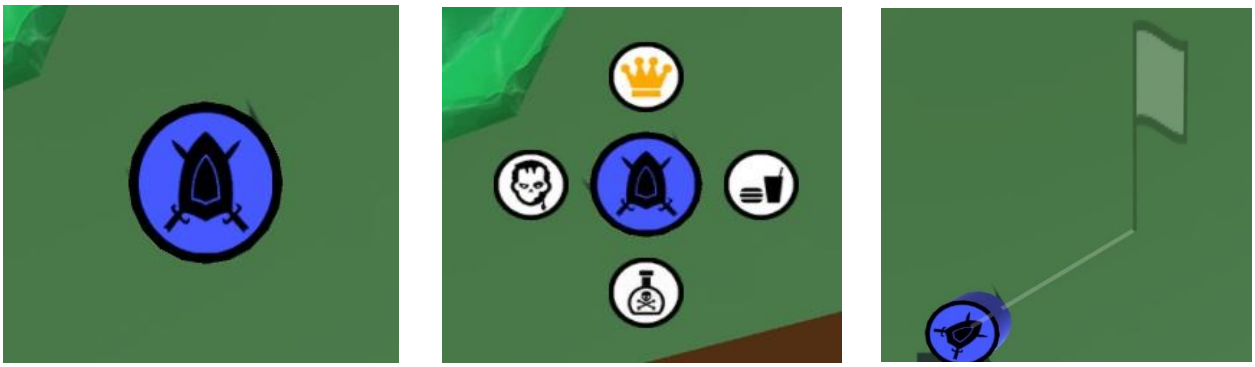

***Figure 23**: Left: A Chester; Middle: A pie menu appears upon selection; Right: The Chester is assigned an action*

After deciding how the units were going to look, the next imperative to the game's design was giving the units a way to receive commands. The team had decided early on that the way to command a unit needed to be quick and simple to allow for the bigger elements of the game, such as the behaviors, to gain more focus. The idea of pie menus were presented and soon expanded upon within the early prototypes of the game. Pie menus allowed an interesting way of giving each unit their own set of commands while keeping the accessibility of each command within a short distance of the mouse which the team considered beneficial for developing the game to fit a puzzle-like theme. An example of unit assignment with pie menus can be seen in Figure 23. Another idea thought of in contrast to the pie menus were a list of commands that would display either next to the unit or at some specific position on the screen. However, the idea of a list was ruled out in favor of the pie menu as it appeared clunky while not entirely fitting the minimalistic style the game was steering towards.

3.4.4 Player Avatar, Economy, Visibility

As the core gameplay was based on standard RTS controls, the player character was first designed as merely being an omnipotent character who could interact with any unit on the map at any time. One of the first units designed was an assassin who would be able to instantly kill the king, and it was discussed briefly that this assassin could be present in each level and serve as a physical player avatar in the game. The player would need to move the assassin next to other units in order to give commands. The motivation behind this idea was to try to make the player more of an equal to the king as the king needed to get close to his knights in the game world in order to enlist them. In practice though, playtesters had enough trouble even telling which of the units on the board they were able to command at all, and the need to move a physical character around to issue commands to the rest of the army felt like an unnecessary obstacle towards setting a plan into motion.

One area where the game does differ from its RTS roots is the distinct lack of an economy needed to purchase units. The team had prototyped factories to allow the player to select which units they wanted to use to complete a level, and an economy was needed to balance each level. In the end, it was decided that the player experience could be more easily scripted if the number and types of units for each level was predetermined. The factory idea, while allowing

players to use their favorite types of units, was too open-ended for the final game and was left out along with the economy.

Visibility was another obstacle involving both game mechanics and the lack of an artist. Before deciding how best to display the personality attributes of the various units, the team first needed to decide how much of the map was going to be visible at any given time in order to know where to display that information. While many RTS games have a "fog of war" that limits player visibility, there was mutual agreement that it did not complement the puzzle-like structure of the levels if the player could not formulate their strategies with full information. Likewise, a similar idea of having an "Oracle" character that the player would interact with to learn more about enemy attributes was also scrapped in order to make it easier for the player to plan their actions.

3.4.5 Level Design Choices

After the initial concept for the game had been finalized, the team began to discuss how the game would be structured in terms of its levels. Initially, the team wanted to create one, big level that would represent the entire game world. This level would change permanently each time the player started a new campaign based on the choices and actions of the player on their previous playthroughs of the game. These changes would include a path being blocked because of a rockslide caused by the player or a village being overrun with knights due to the king responding to an uprising the player incited. The team felt that this sense of continuity would make the game world feel more alive and would more easily sell the player on the idea that the king was an actual, intelligent threat to overcome. The problems brought up with this kind of game world was the amount of time that would be needed to balance it and the difficulty in conducting playtests as the testers would need to invest a significant amount of time to see and give feedback on all of the features.

On top of statically changing levels, the team also briefly discussed having procedurally generated levels. This kind of level structure can ensure infinite replayability for the player while giving each person a unique experience. However, the team felt that this kind of content

generation would be difficult to implement well within the timeframe of the project and decided to leave it as a possible stretch goal after many of the levels had already been completed.

It was decided that having multiple, smaller levels benefitted the project the most based on a number of factors. Unlike the large, changing level, the team could easily conduct playtests across a wide audience at this school without needing to ask for a large time investment that many students are not able to give. With smaller levels, the team could also more easily define what the player experience would be as well as explore different experience goals on a level-by-level basis. One of the strengths of the Unity engine is the ease with which a team can rapidly prototype ideas, and smaller levels supported this more than the other ideas for level structure. This structure also supported the team's own timeframe by allowing the team to create as many levels as possible after the core mechanics of the game were established and the complex behaviors of the units defined.

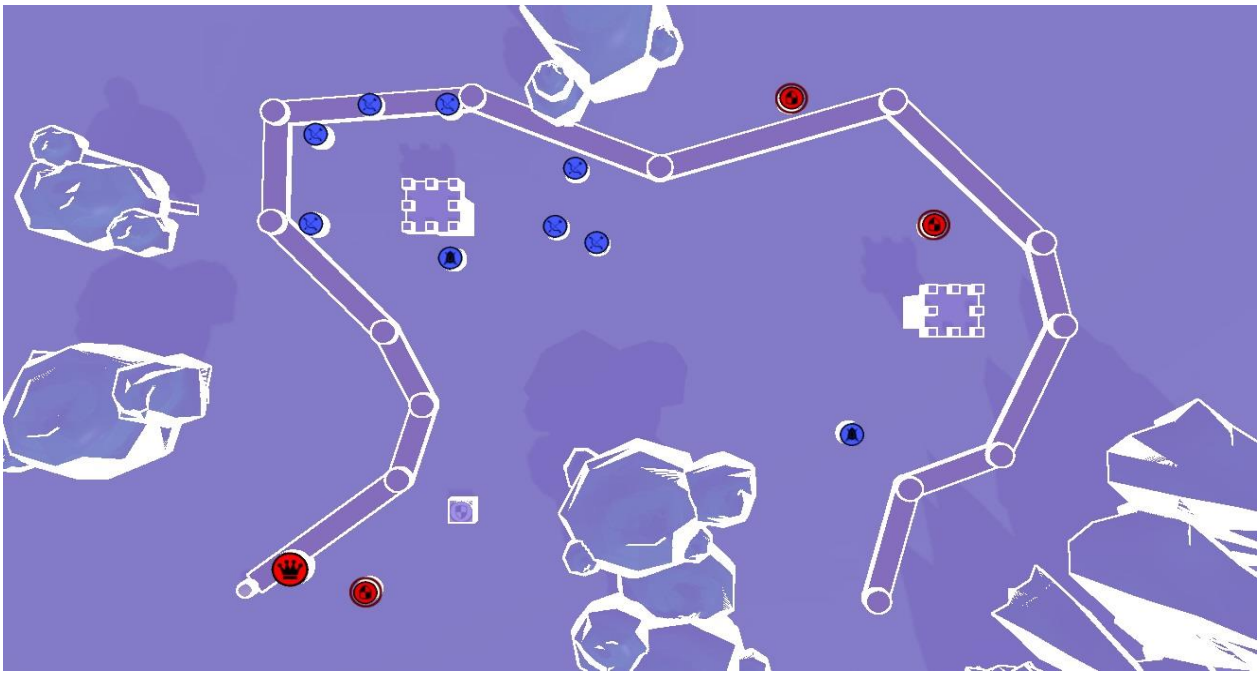

***Figure 24**: An example level with multiple unit types*

With multiple levels becoming the focus of the game, an example of which can be seen in Figure 24, the team also needed to come up with the order in which the player would experience those levels. The most obvious way to accomplish this would be to have a linear level system

where each level (except for the first and last levels) was preceded and followed by exactly one level. While this worked well, the team was still interested in the idea of having the player's actions affect the game world and wanted to put in a system to change levels if the player met certain objectives in the levels before it. Specifically, the team was discussing making the final level very difficult and having each level leading up to it having some side objective that would offer benefits such as a goblin attack to distract the king's units or more peasants joining the rebel cause offering the player more available units.

To complement that idea, the team discussed a tree-like structure of levels where the overworld of the game would have multiple branching paths. Completing a level on one branch would result in unlocking the next level only along that branch with the exception that each of the branches would be able to unlock the final level at the end of them. With this tree-like structure, a benefit towards the final level would only be bestowed upon the player after the finished all of the levels along one branch. The player would be able to unlock all of the benefits on the final level by completing all of the levels on each of the separate branches. This way of granting bonuses to the player allowed the bonuses to be more impactful, and ultimately, the team decided on this level structure for that reason.

A summary of the various levels can be seen below.

- **Tutorial 1**: Introduces the king and his knights, shows how combat works, how traps work, how the king responds to his increasing fear and how the player can control his units.
- **Tutorial 2:** Trappers introduced, player shown the other types of traps and how to rearm them.
- **Tutorial 3**: Thugs, Archers and Archer Towers introduced, player taught how to drag click to select multiple units.
- **Tutorial 4**: Bards and Priests introduced, player shown how to manipulate knights' fear and loyalty to remove them from play, king's paranoia shown.
- **Tutorial 5**: Chesters and their various chests introduced, King's greed and multiple other attributes explained.
- **Tutorial 6**: Neutrals introduced, shown how they can be recruited by either side.
- **Save the Bridge Builder**: A peasant is being attacked by goblins, but the king refuses to help. The player can easily kill the king with their army, but is saving the peasant worth the risk?

- **Scouting**: The king is headed towards a stockpile of gold, and his knights are sweeping the woods. The player must quickly corral his army to focus on the king before the enemy converges.
- **Fleeing**: The king and a few guards are patrolling the woods, unaware of a group of bears tracking them down. His superior forces can be stalled long enough to be mauled using snare traps.
- **Migrants**: The king is beckoning neutral troops from all over the land, and they're headed towards his meeting ground. Convert them before they reach him, then stop his new army as they begin to march on.
- **Bard Conversion**: Several knights are out partying in the woods, and the rebels snuck in some bards. Have them convert a small army before the king stops them in the act.
- **True Path**: The king is headed down a well-protected but widely branching road. There are enough traps to kill him with ease, but only if the player can find out which path he's going to take.
- **Choice**: While preparing an ambush, some trappers and chesters are stuck in a cave with goblins closing in around them. The player's forces only have time to save one group before the king arrives.
- **Horror**: An undead monster is lurking within a guarded area and is said to hold back a hoard of its brethren. Navigate past the king's defenses and unleash the hordes on the nearby King.
- **Shield**: One of the king's knights stumbles upon a legendary artifact: an arrow-blocking shield. With only archers available for hire, the knight has to be separated from the king.
- **Ghoul Assault**: The rebels have been digging through a cave expansion but unleash a horde of undead. Priests are needed to keep the rebel army alive while they push through to the king
- **Rearm**: The king is turning on lamp posts that are conveniently near bushes filled with poisoned berries. Use a single chester to continuously re-arm a gold chest that will make the king greedy enough to eat all the poison himself.
- **Spooky Forest**: The king and his knights are passing through a haunted forest, and within lies a boogeyman that eats kings. Scare off the knights using bards and chesters to leave the king undefended.
- **Hero**: A single thug stands against overwhelming odds, but he has an entourage of bards with him. Have them sing his praises and strike fear in the hearts of his enemies to pull through.

- **Battle**: It's an all-out war between the king's army and the rebels. Place the bards and priests for maximum effectiveness and stop the raid before the king retreats.
- **Waves**: Rebels lie in wait for the king on a bridge, but the tide is rising. The player must carefully manage their army to avoid the crushing waves until the king arrives.
- **Snipers**: The king is crossing near a mountain, with abandoned defenses on top. Creeps and guards must be eliminated to let archers take potshots at the king from the high ground.
- **Picking Off**: The king is calling his knights to him from around his encampment, but they're coming in separate waves. Overwhelm the isolated guards before they can merge into a regiment, then eliminate the king when he tries to run.
- **Finale**: The king of kings is within his heavily armed castle, but the events from other levels are leaking in to threaten him. Eliminate his royal guard, then stop him before he can escape.

3.4.6 Game Mechanics Choices

As the mechanics of the game were being fleshed out, the team found that they were continually adding mechanics that were making the game feel more like a Real-time Strategy game (RTS). The actual levels, however, were being structured as if the game were a puzzle with a select number of units available to the player and a few set ways to complete each level. This choice of what kind of game was being made was a continuing point of debate for the team for the duration of the project.

Initially, the player could only command units by selecting them individually and choosing which actions they wanted the unit to perform. For levels with larger numbers of player units available, however, individually selecting units become very tedious. To combat the tedium, the team implemented functionality for the player to click and drag to select multiple units at a time and give orders to units as an aggregate instead of on an individual level. This saw immediate improvement in player enjoyment of the game during playtests, but the team worried that the game was losing its identity as a puzzle game.

One of the game's main features to come out of this discussion was the idea of a strategy mode. In this strategy mode, the game would be frozen and be shaded in such a way that mimicked an engineer's blueprints. In order to cement the idea of puzzle-like mechanics, the

player would be able to interact with their own units in this frozen time space and reorder units to take different actions before leaving the strategy mode to watch their orchestrated plan unfold. Being able to freeze the game in such a way also allowed for quick re-planning should the game move in a way that the player didn't expect thus giving an element of tactical decision-making to each level.

The team discussed adding more features that were similar to other RTS games to take advantage of the transferrable skills from that genre. The worry was that if the game was too much like an RTS, the levels would be focused around careful unit micromanagement instead of trying to find what the team intended as the solution to teach the player about some aspect of the king's behavior. It was for this reason that the decision was made to not allow players to order a unit out of combat after that unit has already engaged an enemy. This also helped to reaffirm the idea that the units in the game had minds of their own as a person is unlikely to turn their back in the midst of a sword fight regardless of what their commanding officer says.

4 Implementation

4.1 Tools

In order to build the game, managing the scope and gameplay elements that the team wanted to incorporate took precedence. Maintaining an achievable level of scope became a task that would surface again and again while the user interface was redesigned after every bout of playtesting.

The core of *Kill the King* is built upon the Unity (Unity Technologies) game engine, most renowned for its cross-platform capabilities as well as its flexibility in producing a wide variety of games. Unity was required for the completion of the project as the core middleware RAIN (Rival{Theory}) could only be used as a plugin for Unity's interface. As a result, the entirety of the project revolved around using RAIN within Unity's framework to develop the complex behaviors and interactions that the team strived to achieve.

4.2 Version Control

For version control, the team needed a software that would allow easy and quick versioning between different features for speedy playtesting while providing an intuitive interface to rely on. While other version control systems such as Perforce and Subversion were considered, Git became the chosen method for two main reasons. First and foremost, the project itself did not revolve around art assets and thus using Perforce or Subversion would have created unnecessary overhead considering the size of the art pipeline for the game's direction. Following the need for fast iterations and the ability to divide the number of desired features among the team, Git was chosen as the prime method of source control for the project.

4.3 Game Implementation

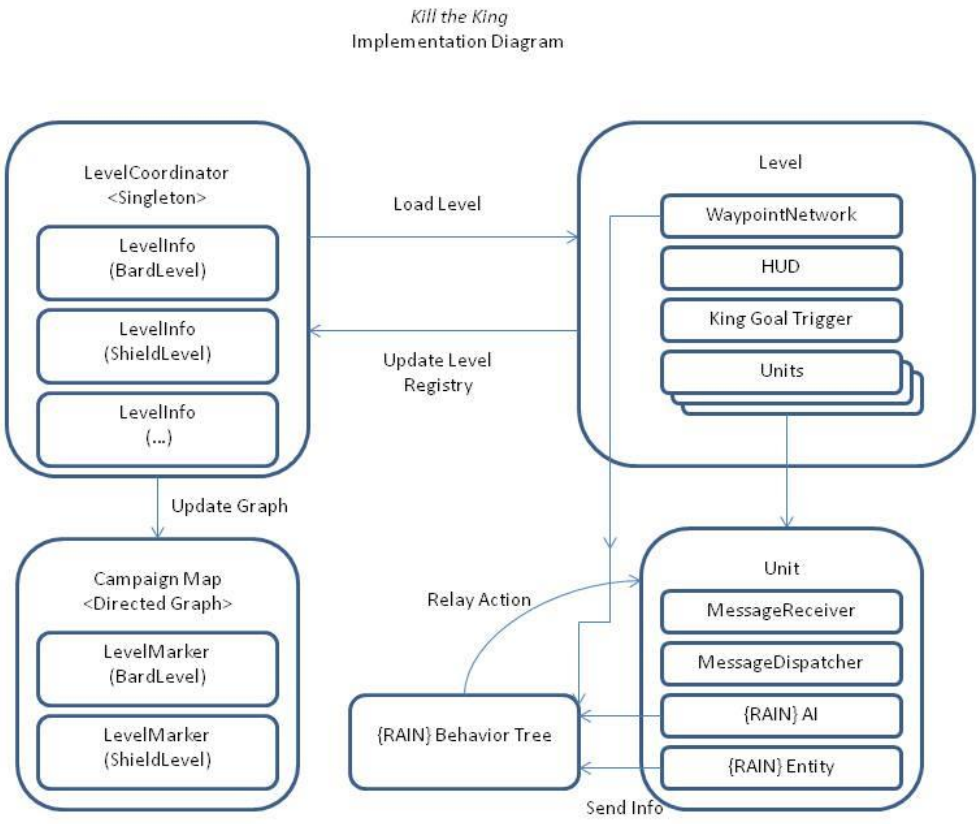

***Figure 25**: Implementation Diagram*

4.3.1 Level Markers

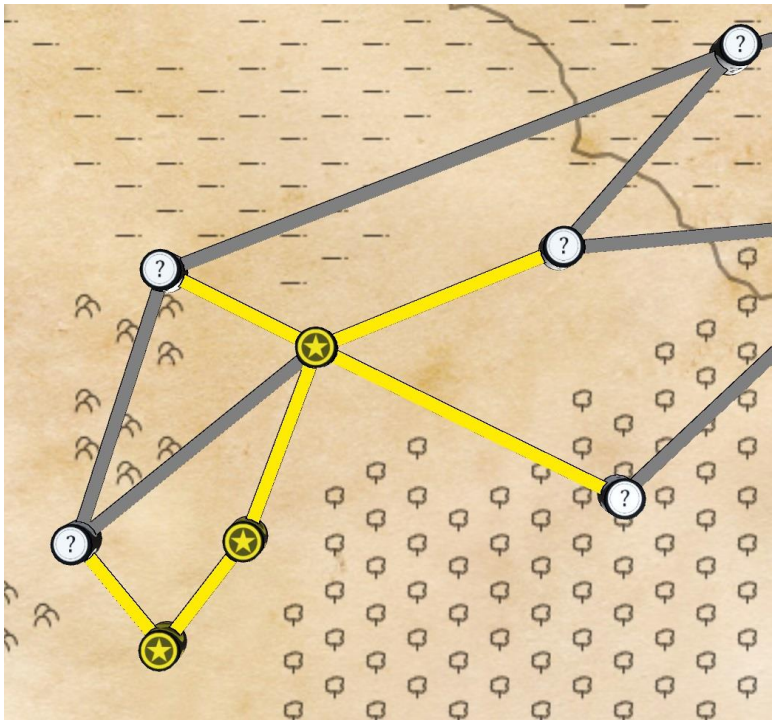

Figure 26: Partial Campaign Map

The structure for *Kill the King* begins with the Campaign Map in the bottom right of Figure 25. The final version of the Campaign Map can be seen in Figure 26. The Campaign Map is a directed graph where the individual nodes are Level Markers. These classes contain information as to whether the particular level indicator on the map is available to enter, what levels are required to access it, whether or not the marked level has been completed, and if the level has a subquest that has been completed. Using the Unity Editor, the Campaign map is arranged by hand to allow a designed intent for how the player progresses throughout the variety of levels available to them. The reason for implementing level markers like these was to provide for an easy way of crafting a custom campaign map that fit the progression that the team saw as necessary for the player to become familiar with the mechanics before attempting future levels where one or more relevant game subjects were showcased.

4.3.2 Level Coordinator

While level markers control the visibility and accessibility of the levels themselves, the Level Coordinator is what allows for the linkage between markers and the core levels themselves. The Level Coordinator is a singleton that contains a dictionary of levels as well as the information stored within them as seen in the top-right section of Figure 25. Information important to each level is whether or not the level has a completed side-quest and whether or not the level is completed, as well as their descriptions and any additional metadata. Every time the player loads the campaign map or returns from playing a level, the coordinator runs through the graph of level markers and updates the information relevant to displaying the opened edges available for the player to traverse. The implementation of the level coordinator was necessary because it allowed for information to be passed between levels should there be an event within another later level that depended on the completion of a previous subquest. In addition, the level coordinator wrapped the Unity level loading tools into a static collection of methods that would allow for easier handling of the project's scenes.

Leading into the next level down from the level coordinator are the levels themselves which are composed of a few important elements as seen in the top-left of Figure 25. The waypoint network is the collection of waypoints throughout a level through which the king traverses with his squad of knights. Upon initialization, the King uses RAIN's navigation system to locate the nearest node in the network to begin his journey throughout the level. Upon collision with an end trigger, which takes the form of a box collider, the game enters a losing state where the player has the option of retrying the level or returning to the campaign menu. Coupled with each level is the user Heads-Up Display (HUD) which displays information concerning the king's attributes, the game speed controls, and an additional space for tool-tip text when the player hovers over a particular unit.

4.3.2 Units

The real core of the implementation comes down to the units themselves. Every unit is coupled with a message receiver, a message dispatcher, and the RAIN (Rival{Theory}) components necessary for communication with an assigned behavior tree. The message dispatcher allow for units to send out information between either a selected ground, range, or

type of unit without the need for implementing methods for each type of message there is to send. The receiver on each unit accepts this sent message and chooses an action to perform based on the information sent. For example, uses of these elements can be seen within *Kill the King* when multiple units are selected. A dispatcher informs the units of the action and the receiver reveals a circle of indication around the unit to confirm activation.

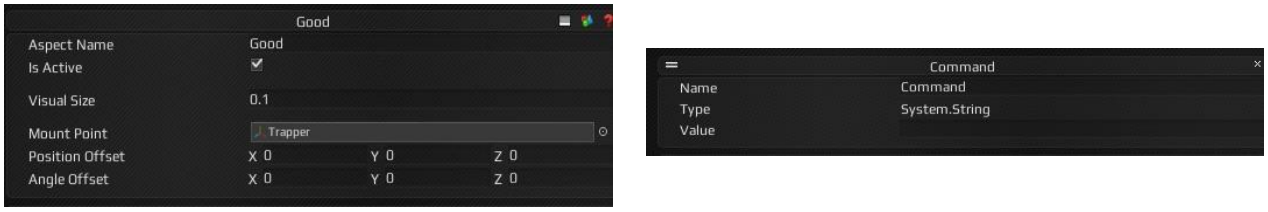

***Figure 27**: Left: A RAIN entity; Right: A RAIN variable*

In order for the unit to communicate with its respective behavior tree, it needs to be given an AI and Entity components that can be seen in Figure 27. The Entity component acts as an identity tag that allows other units upon perception to recognize what the unit is within the frame of RAIN. The AI component contains the visual and auditory perceptrons necessary for detecting the presence of other RAIN entities as well as some ingrained variables that can be fluidly changed and modified as the game enters runtime. Every unit has its own behavior tree within the game which the team manually crafted and adapted to fit the flow of gameplay.

4.3.4 Behavior Trees

While the navigation and perception features of RAIN were useful in the final project, the reason the plugin was its behavior tree editor that allows users to easily prototype and refine the behaviors of the entities to which it is attached. As stated earlier in the paper, behavior trees are a kind of hierarchical decision making AI. Starting at a root node, the behavior of the AI traverses down various prioritized branches or subgoals until it reaches some specific action that it will do at that point in time. The nature of the treelike structure of behavior trees also allows an entity using them to quickly replan their behavior to react to the environment. The flexibility of these trees is part of the power of this style of game AI; any given node in the tree can lead down to an individual action, a different decision branch, or even an entire other behavior tree. The ability to repeat actions and incorporate other behavior trees as subgoals allows for the rapid development

of new unit types with very little effort. They are also often more manageable and easier to debug than other kinds of behaviors as the exact reason for why an AI is performing an action can be determined by simply moving up the branch from the current action.

***Figure 28**: Section of King's behavior tree*

RAIN conceptualizes behavior trees by breaking them down into decision nodes and action nodes. Action nodes can be seen as the leaves on the trees that define what a unit should do after it decides the current state that it's in. Decision nodes are the intermediate nodes that define various kinds of checks a unit can perform to determine the action it should take. RAIN

comes preloaded with standard kinds of decision and action nodes but also allows users to create their own custom decisions and actions. In Figure 28, it can be seen that the king is deciding whether to check a chest or attack an enemy. Each of those goals is further broken down based on the king's various attributes and whether he has a guard who can use.

One of the most useful elements of RAIN is its visual debugger built into its behavior tree editor that displays all of the branches that have succeeded or failed as well as what branches or actions are currently running. This allows a user to see details of behavior trees on a unit by unit basis even when the units are sharing the same behavior tree. The ease with which behavior trees can be created is only furthered by the ease with which they can be fixed if any errors are present.

4.3.5 Strategy Mode

One of the most prominent features is the ability to freeze the game. The ability runs independent of the game loop and temporarily disables the RAIN AI components within the units in the level to create the illusion of pausing. Upon hitting the space key while in the strategy mode, the AI components are re-enabled and follow whatever branch of their respective behavior trees that they paused upon. This allows the game to run in realtime without having to stress the tampering of the game loop. In addition to not having to worry about pausing the game loop, freezing the AI components allows for the loop-dependent code to run to allow on the spot changing of unit orders and actions.

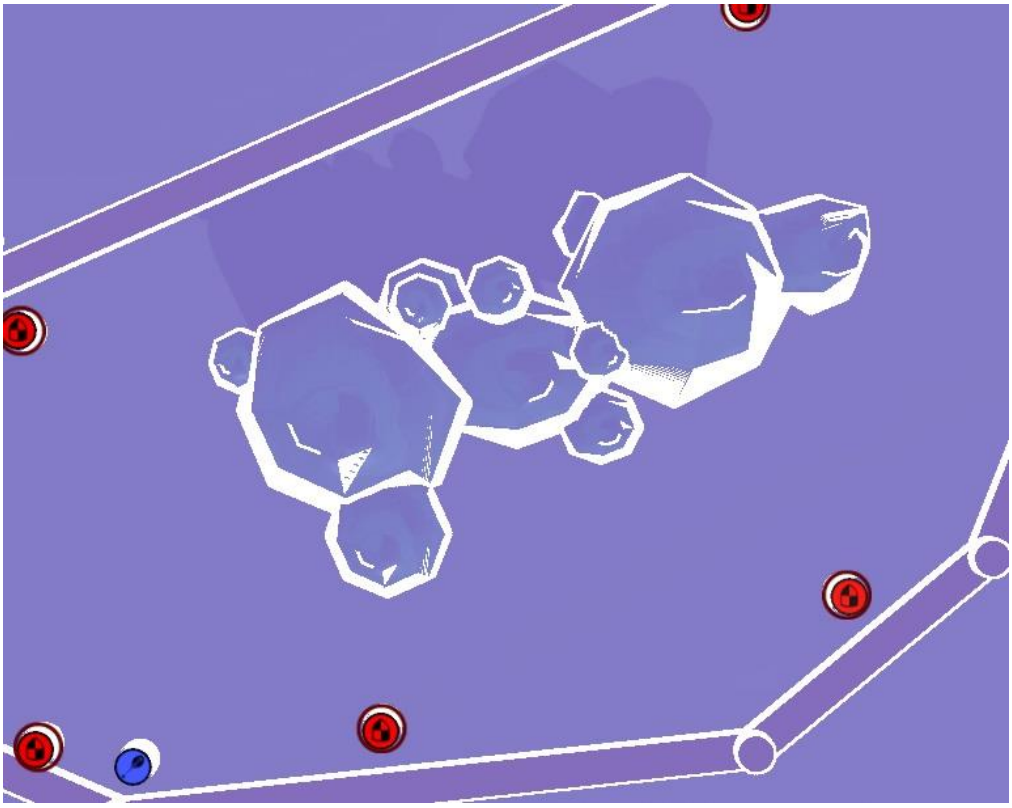

***Figure 29**: Example of strategy mode*

In order to achieve the distinctive blue-white effect highlighted in Figure 29, a custom edge shader was applied over the camera. How the shader works is that it takes geometry within a defined depth from the camera lens and gives weights to any edges that it can detect. The higher an edge's weight, the more likely it is to be revealed in a color specified by the developer within the editor. The resulting effect within the game is a result of fine tuning the edge shader with gradually tested values until the desired aesthetic was achieved.

4.3.6 Pie Menu

***Figure 30**: Example of Chester pie menu*

The Pie Menu consists of two major script components, namely the Pie Menu Manager and the Pie Menu component. An example of the Chester's pie menu can be seen above in Figure 30. The Pie Menu Manager controls which Pie Menus are to be shown on the screen and functions as a singleton for the Pie Menu components to reference. In addition to controlling the display functions, the manager also controls how the Pie Menu elements appear within in a developer-specified radius around a unit or object as well as govern the sizing of the individual menu buttons. While the manager can be attached to an object within the scene, the actual menu components control what commands are operated upon given the player's input. Each menu component comes with the ability to add and remove commands, a functionality that the team saw necessary when trap makers disposed of their ability. Upon initialization of the game, the menu components feed their list of developer-specified commands and icons to the menu manager which displays the given customization as implied by the particular characteristics of that respective menu component.

5 Post-mortem

5.1 Successes

5.1.1 Consistency in style

The team did achieve a very consistent feel for the game both mechanically and artistically. Late in the development process the team wished to add background decorations to break up the monotony of many of the levels. Highly detailed assets would look jarring next to the simple, minimalistic style shared by the rest of the game. Fortunately, the team found a low-poly environment asset pack on the Unity Asset Store that added a breath of fresh air and a sense of depth to many of the existing levels for relatively little effort.

Kill the King also follows the standard puzzle level progression system of highlighting one mechanic or AI interaction in each level to slowly build the player's knowledge of the game. The king will always react similarly to the same stimuli, so the player will gradually be able to beat levels that require manipulation of all of the king's attributes. While a unique mechanic is being introduced almost every level, the player should still feel like everything he learned before is still there and possible to reproduce, even if it's not relevant to their current strategy.

5.1.2 Creating complex behaviors

Using middleware allowed the team to begin prototyping the behaviors of the units at the beginning of the project, whereas studios who create their own AI engines are forced to wait until much later in the development process to begin building their AI and must therefore go through the struggle of integrating it into their existing project. With RAIN, the complex behaviors were allowed to become the core mechanic of the game in and of themselves, and most of the units' behaviors were finished within the first few months of the project. While other behavior trees were added and the existing trees expanded upon, the king and his knights were complete enough to allow the team to begin testing by the middle of November.

5.2 Failures

5.2.1 Code Modularity

Unity's handling of scripting the functionality of objects was very foreign to the team coming into the project compared to other game engines used and previous projects worked on in other languages. This led to a code base with a relatively low degree of modularity; however, this was largely enforced by Unity itself. This does highlight the need to understand the environment a project is being developed in and to look at other projects to see the best practices in that particular environment.

5.2.2 UI Understandability

It was during early play-testing that the real challenge of this project first presented itself. RAIN had allowed the complex behaviors to be developed extremely quickly, but none of the playtesters understood any of the interactions between units or that they, as the player, could manipulate the king's behavior to their advantage. It was obvious to the team what the intended solutions of the puzzles were, but the core concept of the game was not intuitive. The playtesters quickly fell back on their knowledge of RTS games and simply used their combat units to dispatch the king and his guards.

The focus of creating enemies with interesting AI was subsided away during these playtests to allow more light towards developing an interface that was intuitive to the player. The simple, iconographic style of the units, while effective in conveying the purpose of a unit, contrasted sharply with the idea that the player was attempting to fight against an intelligent entity. Through holding other playtests, the team gradually designed and implemented a system of displaying the interactions of the AI that they were content with.

5.3 Final Words

There are a few key lessons that should be taken away from this project. The creation of the kind of complex behaviors this project set out to achieve was largely a technical problem, and it was very fairly easy, especially with a middleware such as RAIN (Rival{Theory}), to

develop units with interesting behaviors. However, it is important to acknowledge the strengths and weaknesses of a middleware when deciding to use one.

RAIN itself seemed more suited to squad-based shooters due to its built-in navigation mesh generation and perception system. The most used feature of RAIN in this project was the behavior tree editor, and while some use was found for the perception system, many of the more interesting features were left untouched. The team still found RAIN to be indispensable to the creation of *Kill the King*, but a more bare-bones middleware with an equally good behavior editor may well have been just as helpful. That being said, the community behind RAIN seemed very passionate, and while the documentation from the developers could have been more fleshed out, it's hard to imagine another AI plugin having as large of a following.

The largest takeaway that the team has taken away from this project revolves around the importance of communicating features to the player rather than set focus on building complexity into the game itself. Through iterative playtesting, the reason for developing an intuitive interface became paramount as understanding why units acted in certain ways in response to certain stimuli was a core aspect of the game. By building tutorials, running through several designs for the interface, and constantly sharing the game with peers and friends was the final version of the HUD in a state that the team was adequately satisfied with.

6 Bibliography

Age of Empires [Video game]. (1999). Redmond, WA: Microsoft Corp.

Cirulli, G. *2048* [Video game]. (2014, March 9).

McCoy, J., M. Treanor, B. Samuel, A. Reed, M. Mateas, and N. Wardrip-Fruin. *Prom Week* [Video Game]. (2012, February 17) University of California, Santa Cruz.

RAIN (Version 2.1.3) [Computer software]. (2014, October 20). Rival{Theory}. Retrieved August 30, 2015, from http://rivaltheory.com/rain/

StarCraft [Video game]. (2013). Irvine (Calif.): Blizzard entertainment.

The Sims [Video game]. (2002). Redwood City (Calif.): Electronic Arts Inc.

Unity (Version 5.1.3) [Computer software]. (2015, August 24). Unity Technologies. Retrieved August 30, 2015, from http://unity3d.com/