Project Number: DXF EB93

Memory Dash - a Multiplayer iPhone Game

A Major Qualifying Project Report:

Submitted to the faculty of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

By

Elisabeth Beinke

Keilin Bickar

Joshua Doyle

Alexander Schwartz

Date: March 6, 2009

Approved:

Professor David Finkel, Major Advisor

Abstract

Memory Dash is a casual, puzzle-based multiplayer game that is available for the iPhone and iPod touch. It is similar to the classic game Memory, but includes real-time multiplayer. *Memory Dash* consists of two rounds, a memory matching competitive multiplayer round and an opinion sorting and guessing cooperative multiplayer round. This game was designed to be enjoyable to play by users of all ages in addition to providing data about users opinions of eBay items for eBay Research Labs.

Table of Contents

Table of Figures

1 Introduction

Note: There will be many mentions in this paper to the iPhone and iPod Touch hardware on which Memory Dash runs. If no further specification is made, assume that iPhone refers to both iPhone and iPod Touch due to the similarity in hardware, software, and compatibility.

1.1 Introduction Summary

Memory Dash is a casual, puzzle-based multiplayer game designed for the iPhone. It was inspired by the classic Memory/Concentration games, but there's a twist: players compete against each other in real-time. *Memory Dash* consists of two rounds, a memory matching competitive multiplayer round and an opinion sorting and guessing cooperative multiplayer round.   In the first round, players move their carts around the screen to different cards, trying to match cards before their opponents can. All the cards have images of eBay items that are pulled from listings. In the second round, four product cards are presented to each player and they must sort these cards based on opinion-gathering categories. Players then have to guess how their opponent sorted the cards, with points being awarded for correct card sortings.

The purpose of creating an iPhone game for eBay was to create a game that would obtain meaningful data for eBay Research Labs as well as be fun and popular, gathering many players. The memory-matching round serves to get people introduced to a fun and competitive game rather than a direct questionnaire of opinions. The sorting round gives players the opportunity to give their opinions on eBay items as well as guess how their opponent thinks. Both rounds of the game together produce a complete game that is addicting and purposeful.

1.2 Background

1.2.1 Purpose

The requirements of the project sponsor were the most important thing to consider in the creation of the game. Before any game ideas could be discussed, it was necessary to determine what eBay wanted from the project. In order to accomplish this, several meetings were held with Neel Sundaresan, the project sponsor for the group and the head of eBay's Research Labs, in order to discuss eBay's expectations of the project. Neel stressed that the game's purpose should be more than just entertainment, but that it should also obtain useful data for eBay or serve some purpose to the user An example of a game employing data mining techniques that Neel suggested is the ESP Game. [1] The ESP game shows two players an image, asks them to come up with tags for it and awards them points for matched tags. As well as having a purpose, Neel also wanted the game to be fun and virally popular, similar to other iPhone apps such as *iBeer* [2], and *Razor - Electric Shaver Simulation* [3]. These requirements and Neel's other requirements of innovativeness, eBay integration, and social networking heavily influenced the final design of the game.

1.3 Development Process

From the beginning, the goal was to create a working, playable demo as soon as possible before moving on to finishing all elements of the game. During the first couple of days, the design of the game was discussed and planned. The core elements of the gameplay were planned out, from which lists of necessary art assets and technical features were created (see Appendix A for the list of art assets). From this list, both sections of the development team (artistic and technical) devised a schedule according to priority (see Appendix B for the art schedule and Appendix C for the tech schedule).

It was very important to get a playable prototype of the game first so potential problems could be resolved early on. The focus was firstly on getting a single-player prototype working before networking could be established. The artistic team created only one level map per district in order to test the functionality of having multiple districts. The art team also created a simple graphical user interface (GUI) for testing purposes. The technical team's focus was on getting basic elements such as touch-input, path drawing and following, and card matching logistics. Once the design could be tested and visualized, the process of building upon this base prototype could begin.

After the basic prototype was complete, priority shifted towards networking, polishing, and overall completion of the game. The artistic team worked to create 3 additional levels for a total of 6 levels, sounds for feedback purposes, complete GUI mockups, and logo designs for submission to the Apple App Store (see Section 4 for a complete analysis of the artistic design process). The technical team focused their priorities on the difficult task of making multiplayer competition a reality as well as fixing minor problems that were impeding gameplay (see Section 5 for a complete analysis of the technical design process).

2 Requirements

2.1 Hardware

2.1.1 Apple Computers

Unlike developing for platforms such as the PC, there are many strict hardware and software requirements one must follow when developing on the iPhone. Most of these requirements are set in stone due to the fact that Apple Inc. has only released Xcode, the software needed for iPhone development, for the latest version of their operating system. Therefore, iPhone development requires an Apple computer. Since Apple created the iPhone, their development tools are restricted to Mac OS X only. Apple's iPhone development requires an Intel-based Mac running OS X Leopard 10.5.4 or higher with a minimum of 2gb of ram. These requirements are essential as there is currently no other way to develop software for the iPhone. Apple has not released a version compatible with the old PowerPC architecture of Apple computers, so Intel chips are required [4].

2.1.2 iPhone/iPod

Although the iPhone SDK does come with a built-in simulator of how a game or application will run on an iPhone, a real iPhone or iPod Touch is also required for true software testing. The first generation iPhone, iPhone 3g or the iPod Touch will work for testing custom applications, but the newest firmware must be installed before doing any SDK work [5]. Without the physical device, it is difficult to predict performance issues when running on native hardware. Also the nature of the iPhone makes it difficult to simulate game mechanics due to the motion sensors and touch screen. Although there are slight variations in the iPhone and an iPod touch, such as the availability of a microphone and the cell phone data networks on the iPhone, the operating system is the same on both devices allowing software to run natively on both without modifications [5]. However, even though applications may run on both systems adequately, there are often slight variations in the computing power that warrant testing on both an iPod and an iPhone.

2.2 Software

2.2.1 Maya

Autodesk Maya is a powerful 3D modeling, animation and rendering solution for the creation of the project's 3D elements [6]. Maya is the tool of choice because of the ease of use of the software as well as the fact that the artists on the team have training with this program. Other 3D modeling tools were considered, such as Cheetah3D, Blender, and 3dsMax, but due to the unfamiliarity with these tools, Maya was decided to be the best option. Both the 2008 and the 2009 editions of Maya were considered, but due to the inability to locate resellers with last year's Maya 2008 edition, the 2009 edition was chosen as the primary 3D modeling program. In addition, Maya project files can be imported directly into the Unity game engine for quick and easy testing cycles.

2.2.2 Photoshop

Adobe Photoshop is a powerful and intuitive image editor for the projects 2D art needs, as well as being the professional standard for 2D art [7]. Similarly to Maya, the artists' familiarity with the program from past work at WPI and its ease of use made it the best tool for the project. Since the older version, Photoshop CS3, was no longer available for purchase and eBay had previously purchased licenses of Adobe software, the CS4 version of Photoshop was chosen to create the 2D art assets for the project.

2.2.3 iPhone SDK

The iPhone SDK contains all the tools a developer needs to develop, test, run, debug, and tune an application for the iPhone [5]. It is a free download for anyone interested in making iPhone applications running the latest version of OS X [4]. The main development tool included with the iPhone SDK is Xcode, which is a powerful integrated development environment (IDE) created by Apple. Xcode includes the ability to edit source code, compile and build programs, as well as the ability to debug the application using a graphical debugger. Most importantly, Xcode can use the signatures given by Apple to registered developers to create authorized applications that can be tested on a device such as the iPod Touch and iPhone. Also included in the SDK is the iPhone Simulator, a program that simulates an iPhone application locally on a Mac; Instruments, a tool for collecting performance data; and Interface Builder, a tool to assist developers in designing a user interface [8].

2.2.4 Development Environment

One of the most important aspects of the planning phase was choosing a development environment. The major requirements were that it be fast and easy to develop in and create quality applications for the iPhone. Several possibilities were investigated before choosing the Unity Engine.

2.2.4.1 Objective C

The first option that was investigated when choosing development environments was coding in Objective C using Xcode. Objective C is the language used to create many programs for OS X and all applications on the iPhone. It is a superset of the language C that uses syntax most similar to the language Smalltalk [9]. The advantages to using Objective C for development are the fact that it is freely available so development could begin immediately, and that there are plenty of examples and tutorials available on how to develop iPhone applications using Objective C. The disadvantages are that no one on the team had experience with Objective C and that most elements of the app would need to be written from scratch, including any 3D graphics engine.

Another option was to use a game-developing program, with a graphics and physics engine included, which could be ported to the iPhone. The Unity engine was decided upon based on one member of the team's previous experience with it, but other engines were investigated for comparison.

2.2.4.2 Stonetrip

One of the engines considered was Stonetrip. Stonetrip is a small French company and their game engine includes support for all of the standard iPhone interfaces. The tool, named ShiVa, has unlimited publishing rights and includes all the features necessary to create a full iPhone game without purchasing additional add-ons. As a bonus, Stonetrip was having a sale on the product for a price of only 750 Euros [10]. Although there were many tutorials and a full reference library, the forums were mostly in French, so this engine was hurriedly rejected.

2.2.4.3 Torque

Another engine proposed was the Torque game engine. The Game Development Club at WPI had used Torque in the past and had dismissed it as difficult to use and learn, in addition to being plagued with problems. Knowing this, the Torque engine was rejected.

2.2.4.4 Unity with iPhone

The engine found to be the most beneficial for this project was the Unity with iPhone support. Unity is a 3D engine that was developed by Unity Technologies, a company based in Denmark.

However, it also has a small office in San Francisco (about 40 miles away from eBay's offices). It contains most of the same features as Stonetrip with some extra graphics and rendering features including occlusion culling and texture compression. Unity also includes a simulator that allows the iPhone to act as an input device to the computer to let developers run the program directly in the editor with the ability to modify game scripts and variables on the fly [11]. Another main advantage of using Unity is the high level of customer support that they provide, including support from the CEO himself on this project.

The Unity engine comes at a price of $1,500 for the professional version and another $1,500 for the iPhone Advanced add-on needed for developing iPhone games. Although they do not offer a trial version of their iPhone development platform, their staff was very cooperative in working with eBay's software acquisition process and was able to give out special trial keys to allow development to proceed.

2.3 iPhone Developer Account and Program

The iPhone developer account is required for anyone wishing to create iPhone applications. Creating an account is a free and easy process consisting of registering an Apple ID on the development section of Apple's website. Once the account has been created, a wealth of resources including tutorials, examples, and instructional videos are available [12]. In addition to the developer account, it is necessary to enroll in the iPhone Developer Program in order to publish applications. Before proceeding, a yearly enrollment fee of $99 is required, and basic checks are done to verify the creator of the account is a real person (including comparing the credit card address to the computer address). Account activation can sometimes take long periods of time because if the account cannot be verified correctly, an Apple employee must then manually activate it.

The iPhone Developer Program Portal is the section of the developer website specific to the developer program and the utilities included. iPhone developers can manage devices for which software can be tested on, create certificates needed to verify program authenticity, create App IDs to identify applications created, make provisioning profiles that allow applications to be installed on specific devices, and finally submit applications to be distributed by Apple using the iTunes store [13].

3 Game Design

3.1 Game Summary

Memory Dash was inspired by the classic Memory/Concentration games, but with the additional feature of players competing against each other in real-time. In the first of two rounds, the player's objective is to guide a cart avatar around the map by using the touch screen to create a path by dragging from the cart's location to a destination. The cart then moves along this path to its destination. Once the cart has reached a card pickup point, players can tap the card to flip it over. Once players find corresponding cards, a match point is awarded. The more matches a player gets, the higher their score. Once round one is over, the second round begins and both players now have to work together towards a common goal. Four product cards are presented to the player and four to the player's opponent. Players must sort these cards based on the current criteria by dragging the cards around with their finger, their relative sorting representing their opinions on the matter. Once the time is up, players then receive their opponent's cards. The better players can read their opponent's mind and match their exact card sortings, the more points they both gain (see Section 3.3.3 for more information on the scoring system).

3.2 Overview

3.2.1 Concept

Memory Dash had the difficult task of having to fulfill many requirements. The game had to be enjoyable to play and catchy. In addition, the game also had to have an underlying purpose aside from entertainment in order to be useful to eBay Research Labs. *Memory Dash* encompasses both of these objectives, within its 2 rounds of gameplay. In the first round, players are asked to match identical cards together by moving a cart avatar on the screen towards the given card in order to select it. Each card displays a random item listing pulled from eBay's listing database. The chosen listing corresponds to the items normally found in the level's district. For example, electronics and computer items can be found in the technology district.

Once all cards are matched, the second round begins. The purpose of the second round is to gather opinions on eBay items from players. Players are asked to sort four items according to a category (for example: which items do they think are most expensive as compared to least expensive, which items would they be most/least likely to buy, etc), after which their opponent guesses how the items were sorted. More points are given the closer players come to matching sorting choices. Therefore, not only are opinions gathered from users, but also the players are rewarded for working together.

3.2.2 Genre

Memory Dash is a casual, puzzle-based game consisting of two parts. The first part is a puzzle matching game, which integrates a real-time multiplayer aspect. The second part however, is a strategy sorting game that includes a guessing element where the player must predict their opponent's actions to gain points.

3.2.3 Novel Features

Because *Memory Dash* was designed to be played on the iPhone/iPod, there were numerous technical features that could be utilized in the game. One such feature is the touch-screen input. Although *Memory Dash* has the basic structure of a memory matching game, it differs in that players draw paths on the screen towards the cards they want to match, instead of just clicking on cards. In addition, the second part of the game allows for opinion gathering by having players guess their opponent's opinions.

3.2.4 Target Audience

Memory Dash can be played by users of any age. The game is easy to learn and has a bright color scheme, which appeals to players both young and old. However, since *Memory Dash* will be distributed for the iPhone and iPod only, the game was designed to include categories that appeal more towards the technologically inclined user.

3.2.5 Target Platform

Because *Memory Dash* was developed using Unity iPhone and the Apple iPhone SDK, it can only be run on an iPhone or iPod touch. Even if the game were ported to alternate hardware, the input relies solely on the iPhone's touch-screen. Knowing that the hardware was pre-defined, technical limitations were considered in the game design (see Section 4.2) as well as the possible technical features that could be implemented (see Section 5.2.1).

3.3 Game Play and Flow

3.3.1 Part A – Matching

The first section of gameplay in *Memory Dash* is referred to as Game Part A, or the matching game. This concept was arrived at during a Pre-Qualifying Project (PQP) planning and brainstorming session. The matching game's purpose is to create a fun game idea that also incorporated a way to present eBay item listings to the user. The idea of using product cards to show a listing to the user was decided upon, and thus formulated *Memory Dash's* gameplay.

The first thing the programmers implemented was the early cart movement code of the matching game. This movement did not restrict the player to any paths and allowed for the ability to drive anywhere on the screen. At this point, the artists were in the process of creating 3d level designs that would contain paths on which the player could move. Because these scenes were flattened down to textures and applied to a flat plane, the programmers developed a way to keep the player cart on a set track. Restricted path movement made the illusion of a 3-D space complete so that the cart could not drive over the flat image of a table or a wall. Instead, the cart was confined to hallways and pre-set paths on the levels. Once the level art and the rail system were in place, the matching game became playable.

When a player starts the game and begins the matching section, they are dropped into the level at the pre-determined spawn points. These points were pre-determined for balance purposes and reduces the chance that one player will spawn next to a card while their opponent must traverse the map to get to their first card. Dragging a finger on the iPhone touch-screen from the cart's current location toward a destination point moves the cart around the level (See Figure 1 for the path movement on the iPhone).

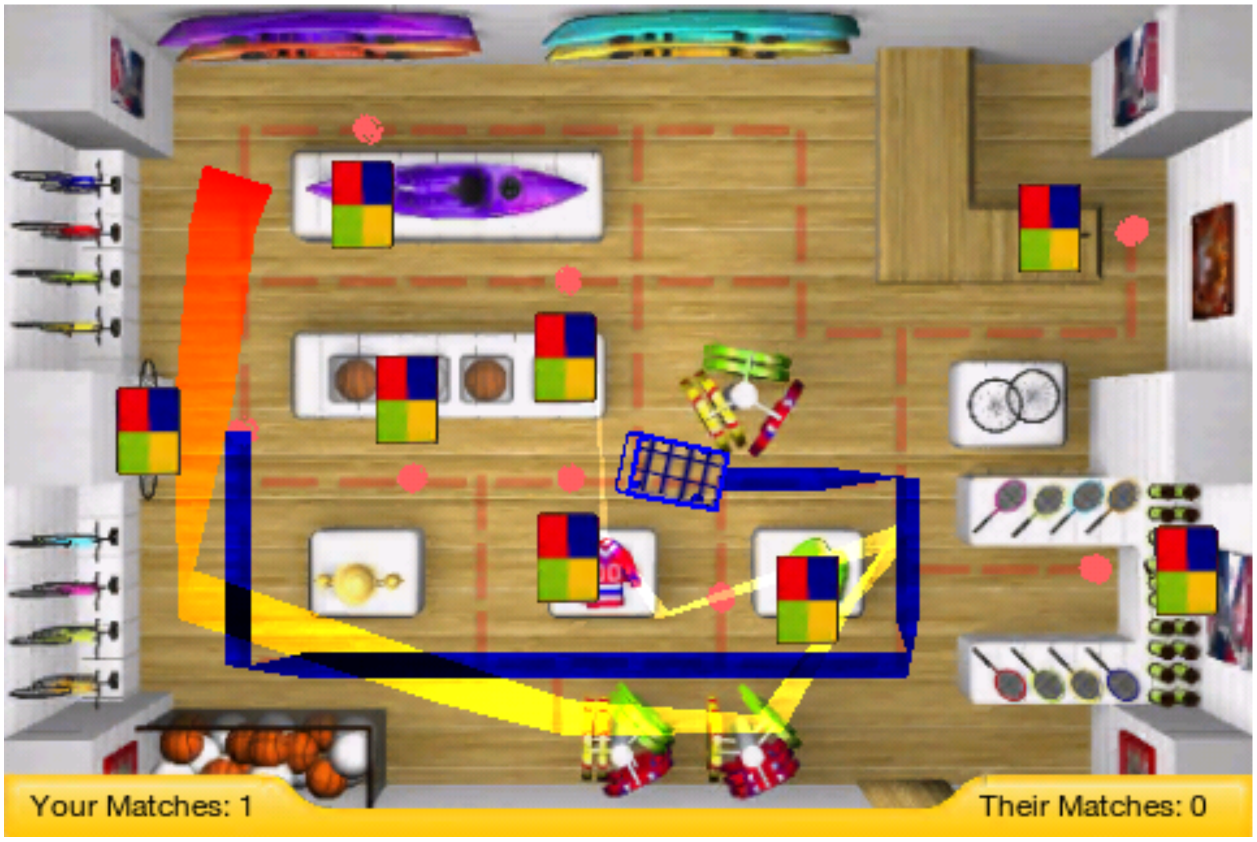

Figure 1: The Sports Store Level

The cart then moves toward the destination by following the underlying rail system. When the cart reaches a card, the player can tap their finger on the card to reveal it. The card then flips over and presents itself to the player by zooming to fit the screen (See Figure 2 for a screenshot of the card selection on the iPhone).

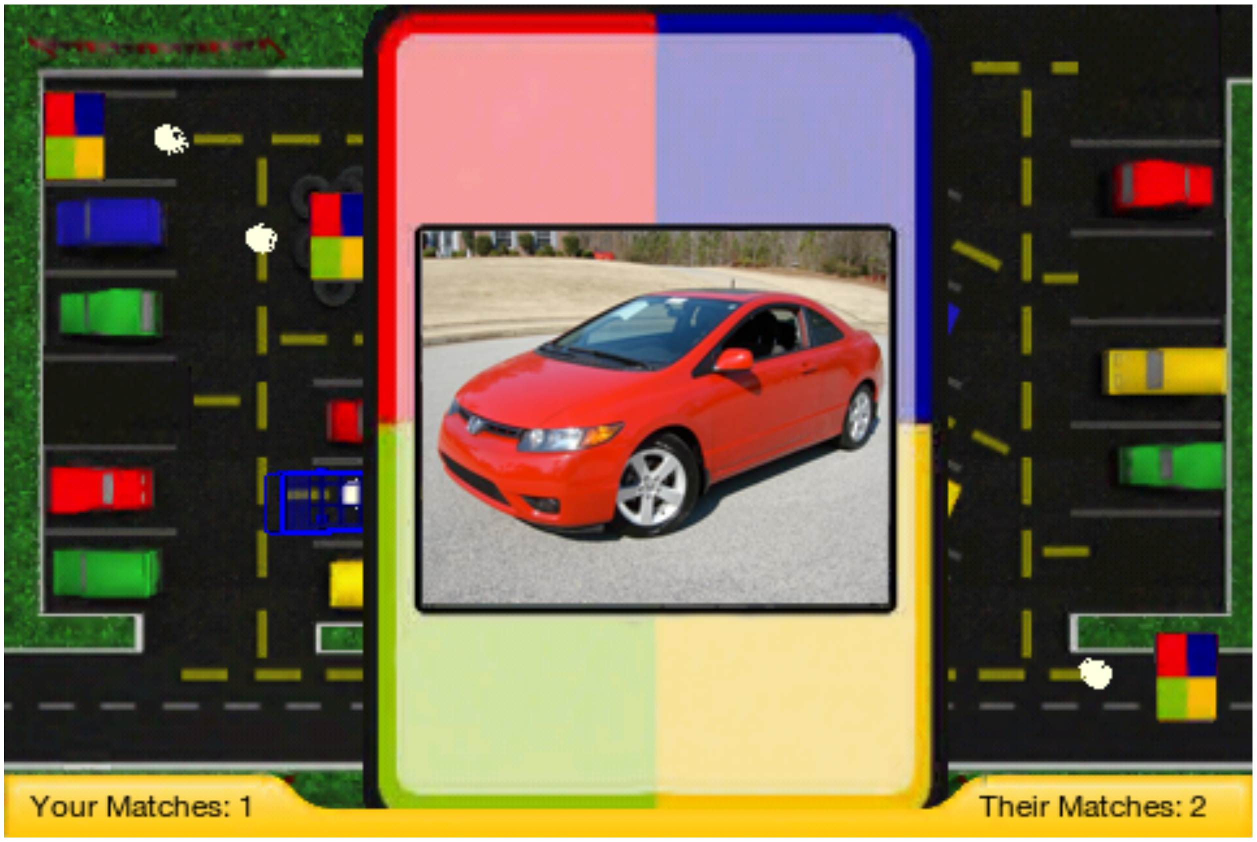

Figure 2: A Selected Card Zooming In

After a quick pause, the card moves back to its location, face up. The player can then direct the cart to another card location and tap on this second card. At this point, both cards will be presented to the screen side by side. If the products on the cards match, the player receives a point and is notified of a correct match via audio feedback. In addition, when a player has collected a pair, the cards disappear, giving visual feedback. The scores of both players are shown in the bottom left on the GUI. When all of the matches have been found, the round ends and the player with the most matches is the winner. In the case of a tie, each player is rewarded with half the score of a full win (see Section 3.3.3 for more information on the scoring system).

This gameplay structure makes for interesting situations such as in the case where one player has a card flipped up and the opponent wishes to flip that same card over to make his match. This conundrum was taken into account when designing the game and a solution was devised to declare a stalemate situation if the game is down to only two remaining matching cards with each player holding one of the cards. In this case, the winning points are awarded to the player with the most card matches.

3.3.2 Part B – Sorting

The programming behind the sorting round was created after the matching round was feature complete for the single player prototype. The first step in creating the sorting game was to implement the dragging and placing of cards in a linear fashion. The dragging around of cards was implemented early, but the actual details of the design of the sorting game were not completed until much later, as there was not sufficient data gathered from the project sponsor. All that was known was that eBay would want to gather certain information from each play session and that some data was more valuable than others. During the first four weeks of development, the project sponsor expressed many ideas through meetings, but nothing concrete was decided upon. The design was not locked down until week 5 of the 8-week development period.

The problem with the design of the sorting round was that two of the project requirements were conflicting. The fun and addictive game side of the project dictated that game part B be less about surveying the user and more about doing something fun and competitive, much like game part A. On the other hand, the other goal of the project was to obtain or ‘mine’ useful data from gameplay. These two requirements were conflicting due to the fact that traditional surveying lacks any semblance of enjoyability and is perceived by users as ‘doing work’. Balancing the gathering of user opinion and having players enjoy the game was difficult as was not making the data mining blatantly apparent. It was determined that the only plausible way to successfully merge these two aspects was to align one player’s opinion with the competitor’s opinion to make a game out of predicting or guessing what the other had decided.

After the difficulties with making the sorting round enjoyable and useful were resolved, the design of this part of the game was complete. When the matching round ends, (regardless of who won, lost, or tied) each player moves on to the sorting round. The round starts by giving four unique product cards to each player. Each player is then asked to sort the four cards based on a certain criteria. An example of one sorting criteria is “Willingness to Buy”. This means that on the left side will be “Least Likely to Buy” and on the right side, “Most Likely to Buy” (See Figure 3 for a screenshot of the sorting round with the criteria, “Rate the Durability”).

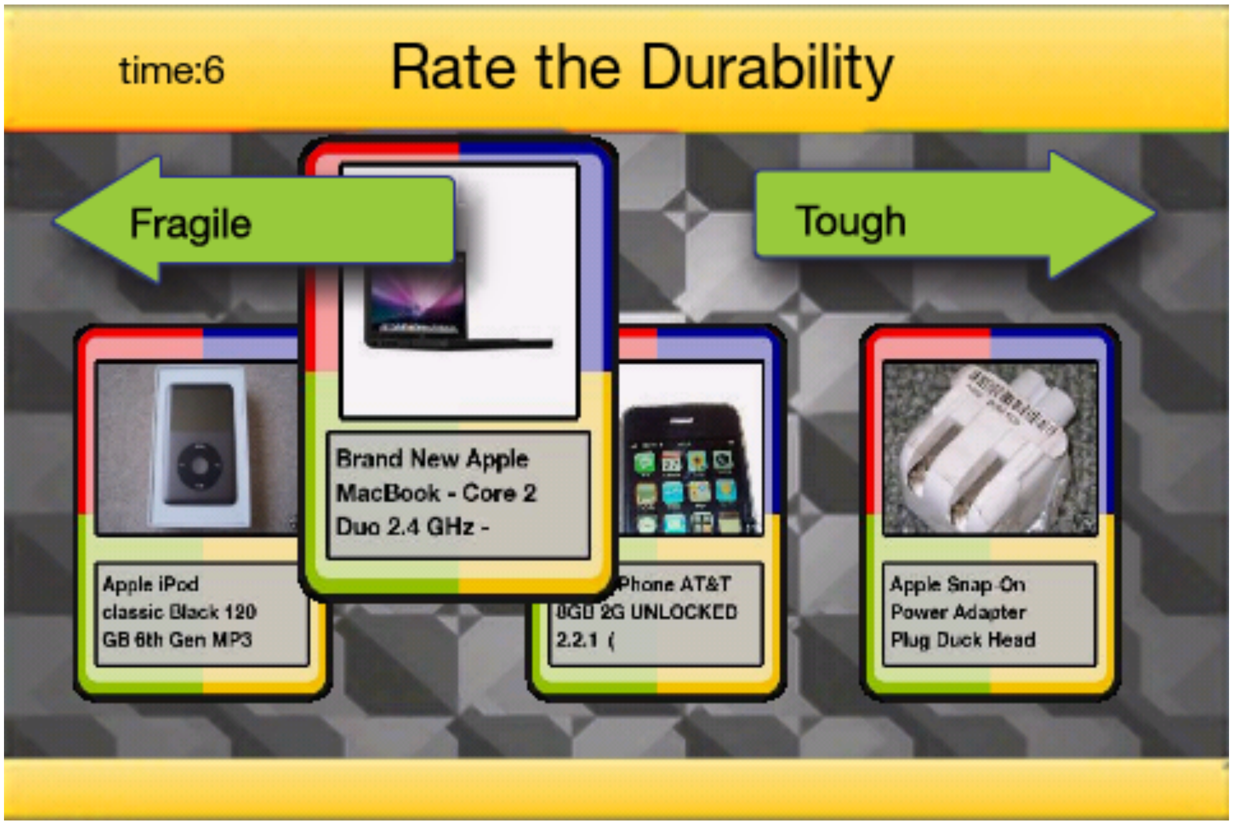
time:6
Rate the Durability
Fragile
Tough
Brand New Apple MacBook - Core 2 Duo 2.4 GHz -
Apple iPod classic Black 120 GB 6th Gen MP3
Phone AT&T 8GB 2G UNLOCKED 2.2.1 (
Apple Snap-On Power Adapter Plug Duck Head

Figure 3: A Screenshot of the Sorting Round

Each player sorts the cards based on personal opinions and is not aware of how the other player is sorting their cards. Each player is given 15 seconds to sort and lock in answers. Once each player has finished sorting, the product cards are swapped, with player one receiving player two's cards, and vice versa. At this point, each player is presented with the sorting question that their opponent just tackled, and the goal is to guess the exact order that their opponent had ordered their cards. Once again, each player is given 15 seconds to complete this task. Once the time is up, all 8 cards are presented and the correct guesses are marked and tallied. This number equivocates to a score bonus for each player. For example, if 6 of the 8 card sortings were correctly guessed, each player would get 6 points added to their total score.

This design was also determined after many discussions about balance and the issues that arise with competitive guessing games. If for example, the game was designed to make game part B competitive in that each player gets points for the correct guesses they make on the other player's sorting, huge problems could crop up. If a player could get a leg up on an opponent by lying and changing answers to opposite opinions, the system would break. Players would have difficulty realistically guessing the sortings of their opponent if their opponent was not truthful. Making this round into a cooperative game ensures truth from both participants. Because the players work together to try to get as many correct as possible, they both receive more points based on how well they work together. This eliminates the potential for bluffing to make the other player lose the round. If both benefit from each other's win, then there is no reason to lie or give false information, thus confirming the information's validity two-fold; firstly, each person is trying to be as honest as possible and secondly, each positive sorting match has been confirmed by two human subjects, thus showing that two people have overlapping opinions on the subject.

3.3.3 Scoring

Memory Dash includes a global scoring system visible in the High Score menu. A player can view the top ten highest-ranking *Memory Dash* players as well as the player's own personal score. Both parts of the game are combined and added to form the game score. In part A, the matching section, the player with the most matches is awarded 20 points for the round, with the loser receiving 0 points. In the event of a tie, both players are awarded 10 points. In part B, the sorting section, players are awarded one point for each matching sorting choice, with a maximum of eight points possible for the round. These points are added to each player's first part score for a total score for the game. This total game score is finally added to the player's global score, which determines the ranking.

4 Art Assets and Art Style

There are many things to keep in mind when planning out a game's core art style and content. The design is a very important aspect to think about. One of the first aspects of art design that has to be kept in mind when deciding on an art style is the company's pre-existing branding. Does the existing intellectual property already have a certain art style to adhere to? In addition, companies often have specific color values that are used in branding. Not only is branding important to consider, but also the outstanding technical restrictions on how to produce the art is important as well. For example, producing a game for the PS3 is very different from producing a Nintendo DS game in both design and development. In addition to developmental restrictions and branding issues, other considerations to keep in mind are copyright issues like whether it is acceptable to use royalty free assets or mention specific company names or logos. Finally, another important consideration when planning out an art style for a game is the issue of deadlines. When does the game need to ship and what can be done to keep to that schedule? All of these factors were considered when designing the game and its artistic style.

4.1 Art Design: Incorporating eBay Branding

4.1.1 Visual Choices

The initial concept for the art style for the game was heavily influenced by eBay's style guidelines. eBay's official colors consist of four primary colors: red, blue, green, and yellow. Not only are these colors shown directly in the company logo, but they are also used throughout eBay's branding. The red-green-blue values of eBay's colors are red (255R, 0G, 0B), blue (0R, 0G, 153B), yellow (255R, 204G, 0B), and green (153R, 204G, 0B) [14]. When designing each art asset in the game, these colors were kept in mind and used wherever possible. It would be contradictory to the eBay style to make a game with many desaturated colors and browns for example.

Significant research was done to examine the style used on eBay's website and emulate the nuances when creating the user interface elements. For example, the gradient background headers on eBay's landing page were examined and used in the design of the game UI (See Figure 4 and Figure 5).

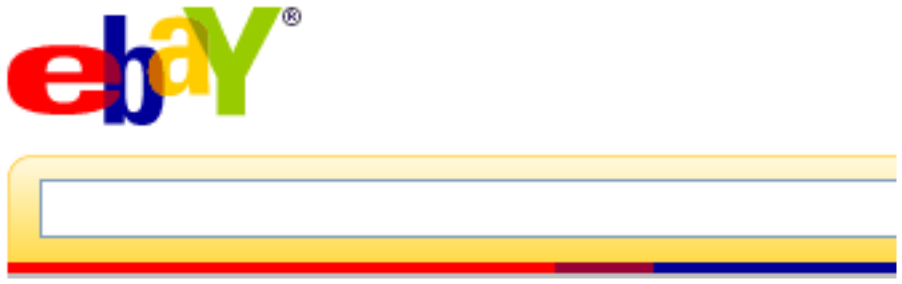

Figure 4: eBay's main header on eBay.com

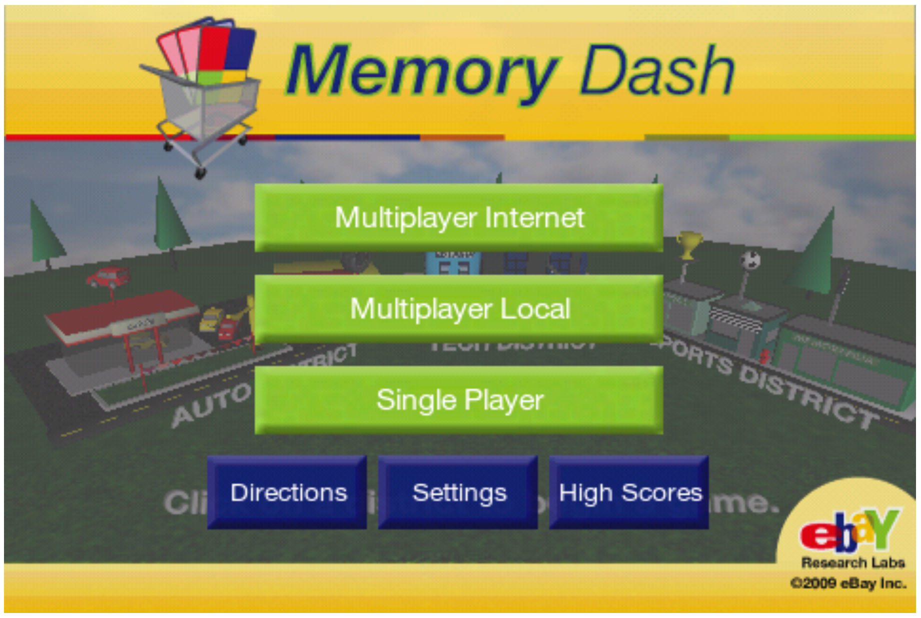

Figure 5: Memory Dash's Main Screen

Other subtle design cues used in the game were also influenced by eBay's front page and were used to suggest eBay affiliation. The use of solid colored outlines on the edges of select objects also mimicked the art style on eBay's promotional materials. In addition, the rainbow-colored thread below eBay's search bar is used in the header of the main screen's user interface. Similarly, the lighting used for interface gradients was also applied in the game.

In addition to visual guidelines, the typeface rules were adhered to. eBay's primary font face is Helvetica Neue [15], therefore this typeface was used within the game. The combination of all of these artistic choices create a game that is not just financially backed by the eBay Corporation, but also an extension of their family of applications.

4.1.2 Use of the eBay Logo

Another concern when adhering to eBay's style guide was the proper use of eBay's logo. Most brands keep specific style guidelines as to how the company logo can be used. For example, most companies will not allow stretching, shearing, or warping of the aspect ratio of their logo in any way. eBay has many guidelines as to the spacing that should be used when using a logo: where in a document a logo should not be put, as well as other guidelines such as no stretching, no drop shadows, no rotations, etc of the logo that need to be adhered to (see [16] for an extended list). Keeping these guidelines in mind, the placement and use of the eBay logo was done in a careful and calculated way. The eBay Research Labs logo was placed in the bottom right of the main menu screen. (see Figure 6).

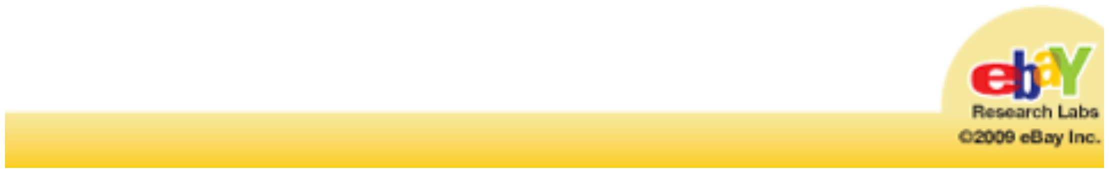

Figure 6: The footer GUI element in the main menu

4.2 Restrictions

4.2.1 Artistic Restrictions

4.2.1.1 Low Poly and Performance Restrictions

Throughout the development of the iPhone game, many technical and physical restrictions became apparent, the first of which regards the processing power of the iPhone itself. While there are slight differences among the different revisions of the iPod Touch and iPhone, all of these devices have effectively the same processing power. Due to the low power of the GPU and CPU of the iPhone, art assets had to be budgeted strictly. The first aspect of a 3D mesh that required careful thought was the vertex count. Unity states that, "to achieve 30 fps on the iPhone, you should keep per frame visible vertex count below 7000" [17]. The number 7,000 may seem high, but with the complexity of recent 3d scenes, it is low by comparison. The Unreal 3 engine, for example, allows for environments of 500,000 to 1.5 million [18]. For this reason, careful forethought was necessary when crafting each art asset in order to keep within the budget.

Certain low polygon modeling tactics were used in the creation of the art elements. Alpha-textured planes are one example of a technique used to keep polygon count down. This term refers to the technique of using flat planes to represent complicated structures. On these flat planes, an alpha texture is used to create areas of transparency and opacity in the image, giving the appearance of a complex shape. This technique was used for assets such as store signs and small triangular flags in the auto district, in addition to the metal bars on the shopping cart avatar (see Figure 7).

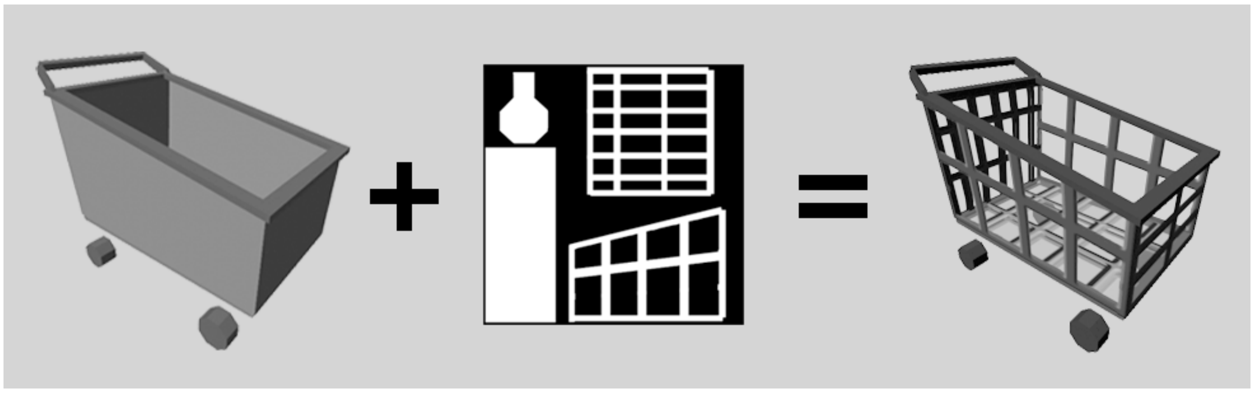

Figure 7: Cart model with alpha channel

If the 3d cart model had used actual polygons to represent every metal bar, the polygon count would have increased the vertex count immensely. Alpha-textured planes are used regularly, even in current PC and console games, and are a must when working with the iPhone due to the hardware restrictions.

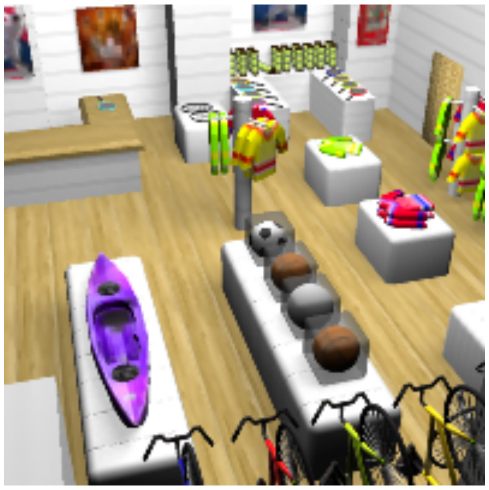

Figure 8: A 3D Render of the Sports Shop

Another way the artistic members of the team dealt with the vertex limit of the iPhone was to render complex 3d models as flat textures to be used in game. For the actual game levels involving cart racing, the backgrounds were created using Maya, the chosen 3d modeling software (see Figure 8 for a render of the 3D scene).

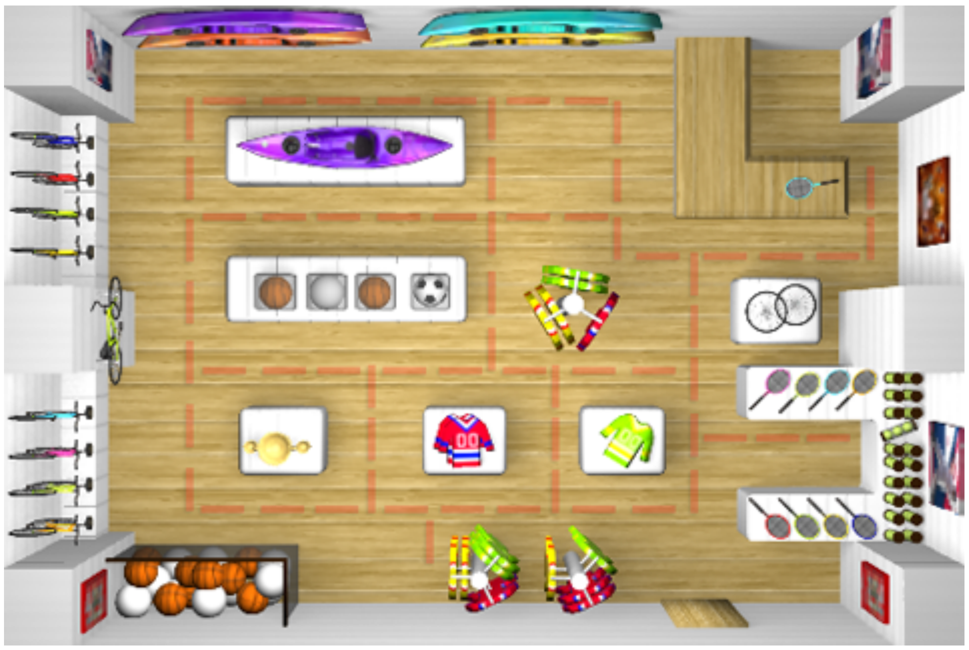

Figure 9: Static Texture Image of the Sports Shop

Due to the static, top-down camera angle and the nature of the gameplay, it was decided that it would be best to render the level as a single texture and apply it to a flat plane in Unity (see Figure 9 for the final map texture used in the game). This way, the cart would be the only 3d model being rendered by the iPhone, thus reducing the polygon count on-screen.

4.2.1.2 Visibility at Low Resolutions

Another restriction when developing for an iPhone is the low screen resolution. The resolution of the iPhone screen is 480 pixels wide by 320 pixels tall [19]. This is relatively low compared to the sizes of common computer monitors, which have screens with a resolution between 1024x768 and 1600x1200, so the iPhone's screen size is a definite constraint when it comes to visual screen real estate. Due to this fact, many different precautions had to be taken. Firstly, readability was a priority for all text presented to the user. Careful testing was done to confirm that all text on-screen was legible and readable for the user. Different font sizes were implemented and compared on the iPhone internally for their relative legibility, readability, and overall aesthetics on the small screen. In addition, this testing of adequate font sizes was a particular challenge when used on cards in game.

Each of the product cards displays an eBay product listing pulled directly from the Internet, along with its listing title. For the sorting bonus round in the game, four of these cards are simultaneously displayed on-screen. This was a challenge when balancing the font size of the text, for readability, and the space for other cards. During the design process, four cards were chosen to be displayed on the screen because more than four would clutter the screen immensely and reduce visibility to an unacceptable level. During the sorting round, four full cards are displayed on the screen. However, in the matching round five matches, or ten cards are displayed very small on the map. The problem arose that it would be very difficult to show ten cards on the screen at a large enough size to discern the different cards. Therefore, the game was designed so that the card zoomed to the center of the screen when selected, presenting the card to the user at the highest resolution possible. When two cards were selected, both appeared side-by-side on the screen for comparison. This effectively solved the problem of small card sizes on-screen.

Another issue arose from the relatively low screen resolution of the iPhone. Certain art assets that looked clear and visible on the computer screen became too small to visually distinguish on the iPhone. One major challenge pertained to the aesthetics of the shopping cart model. Due to the high percentage of transparent sections, the shopping cart literally disappeared at high distances on the iPhone. In addition, the cart was viewed solely from a vertical angle in game, so it was difficult to distinguish as a shopping cart. Special considerations were made for this model to increase visibility. The top rim was reinforced with a thick border, allowing the model to be viewed clearly from above. In addition, the bars on the sides were scaled up in size, increasing visibility. After these changes, the cart model became much more visible, but the textures lacked the clarity of other objects that were viewed head-on. Therefore, research on this problem was done. It was discovered that applying anisotropic filtering to the model's shader clears up the blurriness of the texture when viewed at steep angles. This was the first of many texture issues presented on the iPhone.

4.2.1.3 Texture and Audio Restrictions

Dealing with textures in Unity and on the iPhone was a process with many explicit restrictions. First, all textures imported into Unity were suggested to be sized in square powers of two. This restricted the texture sizes to be in pixel dimensions of 128x128, 256x256, 512x512, and so on. Initially, for the background scenes, images of 480x320 pixels were used to keep with the exact sizing of the iPhone screen. It was later learned that the math involved in working with textures that are not powers of two makes the hardware work much harder. For this reason, the background textures were changed to 512x512 pixels, leaving black bars on the top and bottom where the texture would be cropped off by the screen size. In addition, there was also the general restriction of the iPhone's video memory. Texture sizes of 1024x1024 pixels were simply not possible. Instead, texture sheets were kept very compact using texture resolutions of 128x128 pixels or 256x256 pixels. Lastly, some of the texturing procedures the artists were comfortable with, such as bump maps or normal maps, were not available for use on the iPhone [17]. Due to the heavy processing power needed to compute normal maps on a 3D model, special texturing techniques such as normal mapping, were not used and instead, flat-shaded and solid-colored shapes were used. This worked quite nicely for the game due to the fact that it saved precious iPhone resources and also helped in adhering to eBay's style guidelines.

In contrast to graphical restriction, sound effects were an area of relative ease, specifically, Unity supports .aiff, .wav, .mp3, and .ogg files [20]. The ability to import .mp3 audio was important because this type of compression allows for the use of sound effects without needing to worry about the large size of using uncompressed audio in a downloadable game.

A list of audio selections used in the game was added to a list of the items in need of legal consultation and sent to eBay's legal department for approval.

4.2.2 Copyright Restrictions

4.2.2.1 Art Assets

Often, when working on indie or small games, the tendency is to use free and royalty-free assets to help reduce the workload for artists. In the case of *Memory Dash*, it was necessary to operate under the restrictions of a large corporation. Because of this fact, public art resources, such as game textures were not freely used. Often, sites like cgtextures.com or other free texture sites are turned to for a quick art asset such as a brick wall texture or wooden fence reference picture. In the case of *Memory Dash*, these types of images in the game were not used due to copyright restrictions. The same problem came up when free sound effects were found to put in the game. Although many of the sites claimed that the sound effects available for download on their site were both completely free and royalty free, the validity of these claims could not be confirmed until checking with eBay's legal department. Because of these issues, all art assets, besides sound effects, were created by the art team and thus, completely original.

4.2.2.2 Logo and Branding

In addition to physical art assets, the legal issues of using another company's logo and branding were also a concern during the development of the game. One example of this problem occurred when designing the computer store level in the tech district of *Memory Dash*. The idea of the level was to emulate the look and feel of an Apple Store. This aesthetic included the use of wooden tables, white walls, gray floors, Mac laptops and iPods positioned around the store, and other Apple-specific styles. When weighed with the fact that the use of a specific brand name such as "Apple Stores" would possibly cause legal troubles in the future, it was decided that the level would be stripped of any specific references to the Apple brand and label it as a generic computer store.

4.3 Timeline

4.3.1 Art Asset Listing

One of the most important ways to prepare for development of a game is to carefully plan out every art asset that will be needed in the game before jumping into asset creation. During the first few days of planning, an art asset list for reference and scheduling purposes was developed. For the complete art asset list, see Appendix A.

4.3.2 Art Asset Schedule

This list was useful in making a proper 8-week timeline. This helped by not only organizing the assets into manageable chunks, but also in assessing whether an 8-week development cycle was enough time to complete all artistic elements. For the complete art schedule, see Appendix B

This timeline was designed in order to help effectively produce the quickest working prototype of the game. The first two weeks were critical to make sure that the game would work and end up being fun and manageable. The only way to ensure this was to get a playable demo as soon as possible. The organization of elements beyond the playable demo were sorted by importance, tackling the must-have features first and then moving to the assets that would be completed if there was time. The schedule was also organized in a way that would leave ample time for bug fixes, problem solving, and playtesting near the end of the project. This time period of playtesting and bug reporting is critical in making the game stable and cohesive as a whole.

5 Technical Design

Programming a game requires a well thought out plan and design in order to avoid difficulties in the implementation. When planning the design of the game, there were several major considerations that needed to be taken into account. Chief among these concerns was the platform being used. The iPhone presented several unique difficulties in development including its relative low performance compared with modern computers as well as its unique touch based input. In addition, networking on the iPhone also proved to be a major concern for development. However, careful planning helped to minimize these potential difficulties during game development.

5.1 Technical features timeline

Before beginning any major programming project, it is important to have a clear idea of the requirements of the project. To this end, before any code was written, a list of technical features was drawn up to aid in the planning of development. This list included all of the features of the game believed to be hurdles in the development of the game, such as networking and path finding.

In order to ensure the success of the project, an iterative development process was used. Keeping this in mind, a timeline (see Appendix C) was created that allowed for creation of a playable version of the game as soon as possible so that more features could be added to as development continued. The initial focus of development was on Game Part A, the matching game (further explained in Section 3.4.1), which contains the majority of the core gameplay. Part B, the sorting game, was developed later on as part of the iterative development once the major features of Part A had been implemented. In order to make a working demo as quickly as possible, early development focused on creating a single player version of the game that was expanded into multiplayer once networking had been implemented. The timeline also left several weeks at the end of the development cycle for debugging, play testing and any unexpected delays in development. Scheduling plenty of time for this is especially important, due to the constrained time frame of the project, in ensuring a complete and cohesive product at the end of development.

5.2 Implementation

5.2.1 iPhone specific features

5.2.1.1 Controls

An important difference in developing for the iPhone as opposed to another platform is that all the input for the iPhone is done via touch screen. Interpreting the touch input from the iPhone and turning it into useful data in Unity was the first major programming hurdle. Luckily, Unity iPhone provides functions for retrieving and processing iPhone touch data [21]. Although these functions allowed easy access to input data, this data still needed to be interpreted from touches on a screen to the user's intentions in game. Besides giving screen coordinates for each touch, which needed to be translated into the coordinate system used in Unity, a phase was given which told when the touch was started and when it ended. This information proved invaluable in interpreting user input for path drawing as well as clicking.

5.2.2 Cart Movement and Path work

One of the technical hurdles was how to code movement of the cart around the level. If the graphics of the levels were actually rendered by the graphics engine based on a 3D world instead of pre-rendered onto the background, then the movement of the cart could be based on physical contact with the surroundings and actual movements much like any first person shooter game. However, due to artistic restrictions (see Section 4.2.1.1), the level was pre-rendered as a static image. Given that from that cart's point of view the level is a big flat plane, it was necessary to create another system to confine the player to the set spaces of the level.

5.2.2.1 Node-based

The solution to the problem of restricting cart movement was to create a node map for each level consisting of the available points for the cart to go on. Each node contains an x and y position relative to the screen (values 0-320 pixels in the y-axis and 0-480 pixels in the x-axis), an x and y position relative to the Unity coordinates of the level (screen values offset and scaled by 1/37), a list of connections consisting of the numbers of the neighboring nodes, and number defining which side of the node a card could be placed (or 0 if the node could not have a card).

The advantages of this system are that it greatly simplifies the number of points on which the cart could move and that it defines a set network for the cart to traverse. It also allowed cards to have easy reference points to be placed on, which made generating levels much simpler.

The disadvantages are that the cart has a more limited movement around the level and that the network of nodes needs to be created for each level. The first problem was partially solved by putting enough nodes on the map that the cart could go to most areas, and through path following techniques (described in Section 5.2.2.2). The second problem was made easier through a better data entry system.

The first step of making the node maps was for the artists to label all the points on the level with the numbers of the corresponding nodes drawn onto the image for reference. This was useful not only for creating the initial node maps, but later for debugging problems that occurred from either data entry mistakes or code problems. The next step was for the artists to enter the relevant data into an Excel spreadsheet with four columns: the x and y locations of the node relative to the screen, the list of nodes that the node connects to, and the card position, with each row corresponding to a different node. The artists would then export the spreadsheet as a Comma Separated Values (CSV) file and send all the files involved in the level to the programmers.

Once the programmers had received the files from the artists, they were converted into a form that could be read easily in JavaScript by Unity. This was done by running a custom made Python script on the CSV file that converted the data into a collection of StoreMap and Node data structures that are read directly by the program. These objects could then be used for all the cart movement including finding paths and placing cards.

5.2.2.2 Finding Paths

In a perfect world, users would only try to move the cart exactly from node to node, but it was discovered through early play testing that this is not the case at all. In the first versions of the game, there was no path following at all, just a ribbon that appeared wherever the user touched the screen. This was kept in the game to give some feedback to the user and show where people actually were trying to draw a path.

The actual path is stored in a 200-element points array that works as a queue continually wrapping around as new input is received. Input is only received from the user every 0.1 seconds in order to not fill up the points array with duplicate values. If the points array is ever completely filled, it will no longer accept input from the user until more space is available, but it should be noted that it takes 20 seconds of input to fill the entire array. The points in the array are the numbers of the nodes that the path takes.

To determine which point to add next into the array, the game takes the last point (white dot in Figure 10) and looks at its list of connections (light blue dots in Figure 10). It then takes every node on that list and adds on their connections (dark blue dots in Figure 10) resulting in a list of every node within two nodes of the previous point. For each point in the list it compares the distance from the point to the position of the user's finger and chooses the closest point. If the closest point is more than one node away, the intermediate node is added to the list first, then the closest point. This method works better than directly taking the closest node because it can take input twice as fast if the user is moving his finger very quickly, and it improves the path finding around corners in situations where the user is bad at following the node paths.

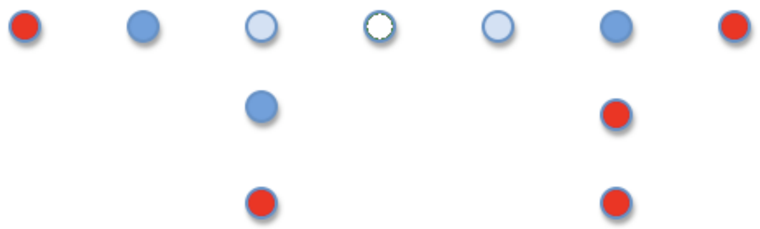

Figure 10: Sample Node Structure

When this method was implemented, it was discovered that tapping on the opposite side of the level would just draw a path automatically from the cart to the point tapped. Since part of the design of the game was to draw the path around the level, a check was added at the beginning of the user touching the screen to make sure that the first point added is actually connected to the previous path or is on the cart if there is not a previous path.

After the points have been inputted into the points array, the final step is to smooth out the path. Often when a user is drawing a path straight through a crossing of nodes (such as straight along the top in Figure 10), he will have his finger slightly off the line so at the crossing, the path will go one node down and then back up making a T shape in the path. This is not a desirable behavior; so any backtracking in the path is removed.

Example:

Starting Path:	(1,2,3,9,3,4,5)
Resulting Path:	(1,2,3,4,5)

This also works for longer paths; the nodes are removed one at a time until it is a smooth line, allowing the user to backtrack by just following the path in the other direction without having the cart trace the entire route.

Example:

Starting Path:	(1,2,3,9,15,20,20,15,9,3,4,5)
Smoothed Path:	(1,2,3,9,15,9,3,4,5)
Smoothed Path:	(1,2,3,9,3,4,5)
Resulting Path:	(1,2,3,4,5)

The final step in finding the path is to make the ending point better if possible. During internal testing in the development stage, many users had trouble getting the cart exactly on the node next to a card (required before the card can be flipped over) because stray movements when lifting the finger would often shift the cart to the next point. This was solved by searching the connections of the point where the touch input ended for a node with a card, and adding that to the points array if found. Not only did this make it easier to get to the cards, but it also made the cart not jump around if the user missed the card and tapped on a node next to the card instead.

5.2.2.3 Drawing Paths

The path of the cart is stored in the points array when the user draws it, but it is also helpful for it to be displayed on the screen. Several methods were tried to achieve this in Unity. The first solution was to create a Trail Renderer (similar to the one following the finger) attached to the input object that follows the finger on the node network. Although this was visually appealing, there is no way to modify the trail once it has started, so the path would not show modifications due to smoothing or backtracking, and it would often fade out before the cart reached it or remain after the cart had already passed.

The solution used was to create a series of primitive objects spanning from one node to the next node on the list. These objects did not look as good as the trail renderer as they are created and destroyed in a more jerky manner, but they have the ability to be added and removed easily by destroying the object if the path changes or the cart passes over one. As objects also can be modified to show different colors, the paths could be customized to the color of the cart that would be following them, even though it was decided not to show the paths of both players at the same time to avoid cluttering the screen.

5.2.2.4 Following Paths

Getting the cart to smoothly follow the path generated posed to be another challenge. As a gameplay decision, it was decided that the cart should always move at a constant speed, meaning the normal methods of moving objects would not work since the nodes can be any distance apart from each other. Furthermore, the card needed to face the direction it was moving, in order to look realistic.

After much experimentation, the following of paths was accomplished through two objects. The first object was an invisible block that would trace the path given by the points array. This block moves in the direction of the next node with a normalized vector so it moves at a consistent speed at all times. However, when the cart was directly linked to this object, it proved hard to identify the direction it was moving, especially when there were duplicate nodes in the points array, so the cart would often flip around and look jerky. Having the actual cart object follow behind the block using the linearly interpolating "Lerp" function solved this problem. The same function was also applied to the turning so the cart would always make the full turn instead of flipping directly to the next direction. As an added side effect, this also made the cart slow down as it arrived at the final point, which added realism to the cart's movement.

5.2.3 Networking and Data from eBay

One of the great touted features of the iPhone and iPod Touch by Apple is its network connectivity. Network connections can be obtained either through the cell phone networks on an iPhone or using a local wifi connection for either the iPhone or iPod. *Memory Dash* was designed to take full advantage of this in order to improve the user's experience and produce more valuable data for eBay (see Section 3.4.2 for more information). The network connection is used both for downloading new content from eBay servers and connecting users in multiplayer games.

5.2.3.1 Card Images

The game was designed such that each card would pull a separate image from an eBay listing, therefore making the matching cards different in each game. The first step in applying eBay-listed items to cards in the game is to find the items that will be on the cards. This is fairly simple as eBay provides a free public API that allows developers to do queries on auctions and get a list of results in an XML format [22].

In order for the iPhone to be able to pull listings from eBay's API, a web server running Apache and PHP was setup that could make calls using the API to get results. The server does all the processing necessary to produce an image of an item on a card using the PHP GD image-processing library. The first step of the process is to load a blank card stored on the server into the script (see Step 1 in Figure 11). The URI returned from the eBay Dev API call is used (see Step 2 in Figure 11) to download the full sized image from eBay's image server. The image is then resized into the center of the blank card image (see Step 3 in Figure 11) and a small rounded border is drawn around it (see Step 4 in Figure 11). Finally, in the fifth step, the image is resized to 256x256 pixels and saved to a file for multiplayer games, or returned as an image to the caller of the script for single player games (see Step 5 in Figure 11). The images are stored as files for multiplayer so that both players will receive the exact same items.

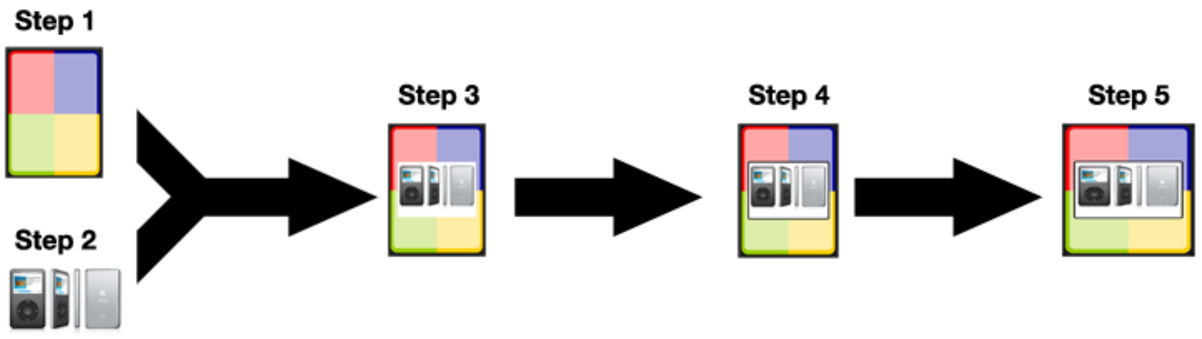

Figure 11: Process of creating a card image (images not to scale)

The use of a web server for the images was chosen because of the ease of importing images using Unity's WWW class. The WWW class can take the results of a web request and convert it directly into a texture (if desired) that can be applied to an object such as a card. The WWW class also handles all the details behind the scene for making the connection as well as for timing out the connection. Since an Internet connection is not always available, several images are stored on the device permanently in case the image server cannot be reached, or if the downloading of an image is not completed within five seconds.

5.2.3.2 Matchmaking

In order for users to be able to play against each other on the Internet, they need to be able to find someone to play, hence the creation of a matchmaking server. Although some matchmaking services will match players based on location or experience, *Memory Dash* simply matches any two players that are playing the same district. This was also done using a web server to take advantage of the WWW class.

The process for matching players is quite simple. The first player who wants to join an Internet game sends a game request to the server. The server then checks for recent open games and finding none, will create a new game in the game database and return the game ID back to the player. When the second player sends a game request to the server, the server sees that another game has been created recently on the same district and returns the same game ID as the first player. At this point the players have been matched and it is up to the gameplay server to start the match.

5.2.3.2 Main Gameplay

Although web connections work great for simple requests, the actual gameplay needed to be something that would work faster and be optimized for messages that the game communicates. Unity includes a clever network class that makes any network communication between players easy. Unfortunately that class was not included in the iPhone version at the time the app was made. This meant that all communications between devices needed to be coded using raw socket connections available in the .NET library.

In order to simplify communications, a standardized message class was developed. Each message sent contains a four byte game id, a one-byte message type, and a one-byte sub-type/action. This allows any device that is in contact with a packet to easily identify what kind of message it is without heavy processing. These message types are listed in Figure 12.

Type	Sub-Type	Description
0	#	Network connection
0	0	Keep alive
0	1	Join internet game
0	2	Broadcast search for other clients
0	3	Reply to broadcast, request to join
0	4	Game Part A Over
0	5	Game Part B Over
1	#	Gameplay
1	1	Start game on first map
1	2	Start game on second map
2	0	Cart movement
3	0	Starting list of card locations
3	1	List of cards ordered
4	#	Card flipping
4	0	Notify flipping card
4	1	Confirm flipping card
4	2	Card flip accepted
4	3	Card flip rejected

Figure 12: Message types

The standard communication packets are converted into an array of bytes before they are sent, but the protocol used for communications needs to be considered. The game design calls for the game to work over the Internet and on local networks, with many people who would be on a private network without an external IP. With this in mind, there are two protocol options: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP), each with its own good and bad points, and both without the ability to traverse private networks easily.

5.2.3.2.1 TCP vs. UDP

Before deciding which protocol to use, significant research was done on the major advantages and disadvantages between TCP and UDP. TCP establishes a connection between the two machines and sends a continuous steam of bytes without distinction of boundaries. It will automatically resend packets that are lost and adapt to congested network connections, making it very reliable. UDP is a connectionless protocol that sends datagrams from host to client without any knowledge of what happens to them. Packets may be lost or arrive out of order to the client without any correction methods built in [23]. In essence, TCP can be compared to making a phone call to someone where both parties need to be on the line and talking, and UDP is more like sending letters in the mail where neither party knows anything about the transfer.

Memory Dash makes use of both protocols in order to take advantage of good attributes from each. As stated in Sections 6.2.3.1 and 6.2.3.2, the card images and matchmaking are done through a web server that makes use of the TCP protocol. Transfers of image files are very reliant on having the packets all successfully transferred, and all received in the correct order. The image transfers also were not under a serious time constraint when downloading as the transfer is done within the several seconds that the game is launching. In addition to transferring image files, TCP is also used for transmitting the results of the game to the server because, in order to have useful data, the results need to be correctly ordered.

Although *Memory Dash* uses TCP for many of its transfers, UDP is also used. In the main gameplay, UDP is used for communications. The unreliability of UDP is not such a pressing issue for the main game, as the next command will typically overwrite it anyways. For example, the movement of the player’s cart is sent as an array of points to the other player (see Section 5.2.2.2). Since the cart will always follow to the last item in the array, a missing message will just mean that it will skip a point and go directly to the next one without issue. In comparison, if the arrays were sent using TCP, then a packet loss would delay the next message until the lost one was successfully transferred, adding lag to the gameplay.

Some items such as the card flips, which required a more reliable communication, also incorporated acknowledgements using UDP to make sure that the action happened. Having this also allowed a solution to when the two players tried to get the same card at the same time, as it is designed so that one chosen player will be given the card while sending a negative acknowledgment back to the other player that this player cannot select the card.

The other advantage to using UDP is its ability to traverse networks. Although there are some firewalls that will block UDP packets sent and received on strange ports, those are usually only on computers. Since after testing, this was not a problem on the iPhone, firewall problems were not a concern. Due to the fact UDP does not need to maintain a connection between client and host, it is possible to trick routers into passing traffic by getting introduced by a server to the other client's public IP, and then both sending to each other's public address. As the router has just sent a packet to the other address, it assumes incoming packets from that address should be routed back to the same location and a "connection" will be made [24]. While this works great in theory, in practice there are many routers that are not configured correctly, so this method does not work in situations with a lot of routers along the communication path.

5.2.3.2.2 Multiplayer Internet Gameplay

One of the gameplay modes included in *Memory Dash* is multiplayer Internet gameplay. This mode is aimed at players who want to play a multiplayer game, but are not within the proximity of another player. This mode uses the matchmaking server to find players that will be in a specific game. In order to make this mode work for as many players as possible, all communications go through a central gameplay server with communications directly between clients.

When a player receives a game ID from the matchmaking server, they send a message to the gameplay server containing their name (the name of the iPhone) and the game ID that indicates they are ready to start. As soon as both players have notified the server that they are ready, the server sends a game start message out to both clients, which starts the game. The server then has no involvement in messages going between players other than relaying them to the other player until the game ends and the server is notified that the game has finished.   The flow of messsages is shown in Figure 13.

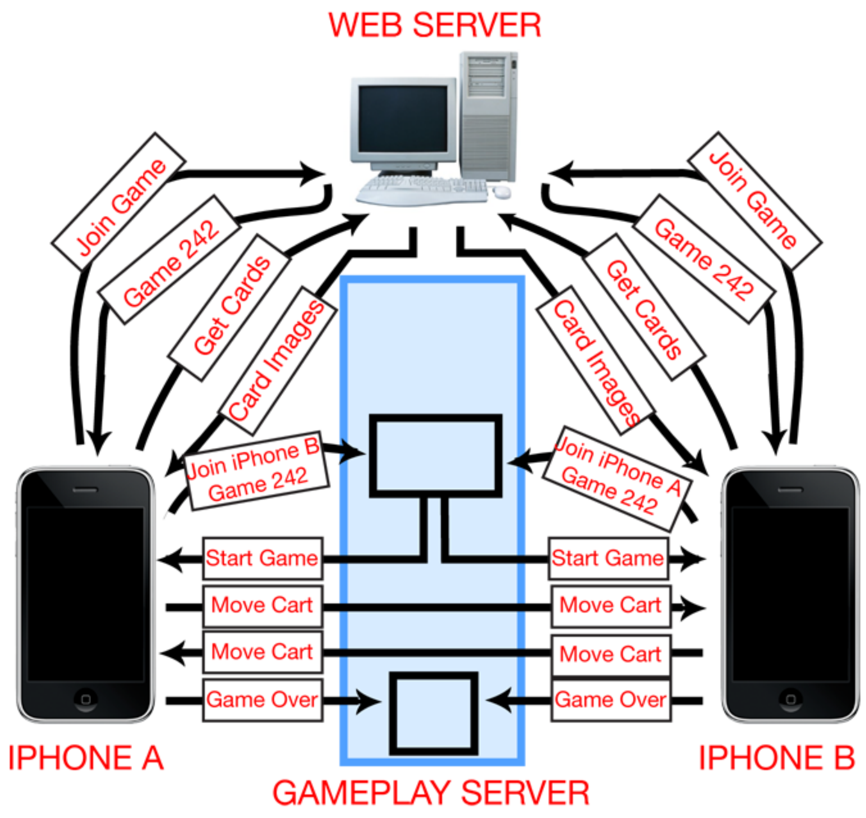

Figure 13 Network interactions between iPhones and server

The other action unique to the multiplayer Internet mode is that the scores are reported back to the matchmaking server for the high scores list and the results of the sorting are reported to eBay for analysis. Both these items are submitted through a simple web interface to the server.

5.2.3.2.3 Local Network Gameplay

The local network gameplay mode is very similar to the Internet gameplay mode with the exception that it does not utilize any servers other than the image server. This mode is designed for players that are on the same local network, such as two people using the wireless network at their local coffee shop. It can also work on a network completely disconnected from the Internet such as someone setting up a wireless network using a laptop on a train.

Since local play does not use a matchmaking server, players need to be discovered in order to connect to each other. This is done by broadcasting a UDP message out to everyone on the local network, notifying them of a device looking to play a game. When a second player looking for a local network game receives this message, the second player's iPhone sends back a reply to the first player asking to join a game. The initial sender then takes the first response and sends the launch game message in the same way the Internet server does, with the name of their device. When the second player receives this, the person's iPhone knows that it's a part of the game and sends the launch game message back to the initial player with the iPhone's name, thus starting the game for both players. Once both players are connected, all communications are sent between the devices in the same way they are in Internet play, just without the server relaying.

Although the iPhones do not need to be connected to the Internet, a network is still necessary. There is an undocumented technical limitation of the iPhone that prevents it from creating an ad-hoc network between two devices, leaving no method of connecting two players without relaying through a router.

6 Project Post-Mortem

6.1 Result

Overall, the project yielded a feature-complete and fun game that will be available for free on the Apple app store (see Section 6.3 for details about app submission). It is a fully working game that includes all elements of the original game design, satisfies the project requirements, and is essentially a fun and marketable product.

6.2 Successes

Memory Dash followed the original game design very closely throughout development. The original game design called for two separate sections of gameplay: game part A, or the matching round; and game part B, or the sorting round. In the final game, users can compete against each other in the matching round in order to get as many matches as they can out of the possible five. Then, users work together to try to guess how the other player thinks in the sorting round. Both of these rounds serve to satisfy the initial project requirements.

All of the major project requirements were met in the final game. *Memory Dash* serves to be an enjoyable and engaging game that will immerse the user in competitive gameplay. In addition, it also serves to mine data about users' opinions that will be useful to eBay Research Labs. For more information, see Section 6.4.2.

In addition to having a feature-compete game that meets all requirements, the schedule devised in the beginning of development served as a useful guide that was followed fairly accurately. Both the tech and the art teams adeptly planned for the last three weeks to be a buffer for any overflow of work that was unfinished, new work that was necessary for completion, and any playtesting and bug fixing needed. Scheduling this padding of time at the end of the project was a successful decision because there were found to be many problems and bugs that needed to be fixed in addition to additional assets that needed to be created. This served to help the development team in producing a working and marketable product that will be released to the public.

6.3 Problems

Although the development for *Memory Dash* was completed on time and the schedule was followed, there were a few major unforeseen issues that arose during the development of the game itself. Overall, these issues included difficulties in obtaining software and hardware in a timely manner, iPhone latency issues, low memory issues, and several multiplayer issues.

6.3.1 Obtaining Hardware and Software

In order to develop *Memory Dash*, several pieces of hardware and software needed to be obtained. Although intensive research went into choosing optimal hardware and software for development (see Section 2 for more detail) and the list of requirements were specified before the project began, actually obtaining these elements quickly proved to be difficult. Since Unity Pro licenses were not received until one month after the project began, the team worked with Unity Technologies' local branch in San Francisco to obtain trial versions of Unity Pro with iPhone Advanced that were used until the purchase order was completed. In addition, licenses for Maya 2009 took a long time to obtain and were received in the second to last week of the project. In comparison, Photoshop CS3 only took 2 weeks to obtain due to the fact that eBay already owned the licenses and IT simply installed it on the computers once the request went through. Thankfully, free trials were available for Photoshop and Maya until both licenses were obtained, however the process to obtain all necessary elements was much longer than expected.

In addition to software, hardware procurement was also a lengthy process. Unfortunately, there were no computers to begin development for the first three days. The team used this time to extensively map out the game design of *Memory Dash*. Also, iPods for development and testing arrived three weeks after development began. iPhones arrived 6 weeks into the project after it was approved and shipped out. Because of the length of time it took to obtain the physical hardware, testing was pushed back to the very end of the project.

6.3.2 Memory Issues

Another main issue was uncovered once the iPhones were obtained. Previously, all testing was done on the iPod and it was not until testing was done on the iPhone that several problems emerged. It was discovered during testing that the iPhone uses 10% of its memory towards checking for calls and therefore has fewer resources available to allocate towards applications. This meant that *Memory Dash* ran noticeably slower on the iPhone when compared to the iPod. This was unexpected and required a complete analysis of the game assets and code to determine the specific things that were slowing down the system.

Therefore, several methods were attempted to decrease the amount of memory the game used. The use of texture sheets and compressed textures were used in several instances. In the main district select menu, the amount of textures used went from 30 to 11 with the use of texture sheets. This meant that several textures had to be scaled down and re-UV mapped within Maya, which was time-consuming, but necessary to save memory. In addition to texture compression, alternate methods of line rendering were researched. Originally, the game was programmed to create a series of boxes to simulate the cart movement path. However, since this increased the amount of vertices displayed on the screen and slowed the game, it was determined that Unity's built-in line renderer was a better choice.

6.3.3 Multiplayer

One other issue that arose was the problem with configuring the server, which runs the game. This server runs the Solaris operating system and was responsible for passing game logic from one player to another, hosting the website, and pulling data from the internet using the eBay API. eBay provided access to a server capable of handling all these functions. This server unfortunately required the installation of many pieces of software. Apache, PHP, and mySQL all needed to be compiled and installed on the Solaris server before it could be used for development. This proved to be a lengthy process, as many of the necessary build dependencies were missing and had to be retrieved before compiling each piece of software. In addition to being a lengthy process, the server was also hosting other Research Labs projects and many users connect to it daily to monitor their work. At one point, after modifying the apache server installation, many other developers realized their websites were not functioning properly and contacted the tech team about this issue. The team was able to quickly revert the changes and avoid any widespread problems caused by the server use.

In addition, during the testing of *Memory Dash*, a problem occurred when trying to access the website labs.ebay.com. The address is only accessible from outside of eBay's network, so only the internal hostname works from within network. A solution was devised that effectively switches between an internal server address and an external server address depending on which address is currently pingable.

6.4 Analysis

6.4.1 Hardware and software choices

6.4.1.1 Hardware

During most of the development, testing was done using iPod Touches rather than iPhones due primarily to the costs and difficulties in obtaining iPhones for testing purposes. Even though iPhones were eventually acquired, development and testing was not impeded because iPods were obtained early on. iPhones were only acquired and used in the last two weeks for final testing. When the game was being tested on the iPhone, it was discovered that there were performance issues that weren't apparent on the iPods. The 2nd generation iPod Touch's processer was increased from 412MHz to 532MHz over the 1st generation and the iPhone [25] accounting for the performance difference. This necessitated the testing and optimization of the code and art during the last few weeks of development. If the iPhones were received early on in the development cycle, last minute optimization could have been avoided.

6.4.1.2 Software

Many Software choices needed to be made as well as the hardware choices. The choice to use Unity to create the game rather then attempting to code everything in Objective C proved immensely valuable throughout the development process. Unity's editor allowed many parts of the game to be built and modified without everything having to be coded. Unity's built-in code for rendering, movement, networking, and integration with the iPhone took a lot of the load off of the programmers. Overall, the choice to use an existing game engine with built in iPhone support, instead of attempting to write a game from scratch, greatly reduced development time and allowed the creation of a nicely polished game at the end of the project. In addition to the usefulness of the engine itself, Unity's staff proved very helpful throughout the project, granting trial software while eBay was still attempting to purchase the engine, and even providing a beta of the new version to help with networking issues.

In addition to the software engine choice of Unity, the artistic software choices of Autodesk Maya and Adobe Photoshop proved to be sufficient as well. The artists were proficient in both Maya and Photoshop, which saved time in learning how to use alternate software. In addition, Maya is officially supported for importing models into Unity, which makes quick changes simple and efficient. Neither of the software choices proved to be difficult to use and allowed for quick, detailed artistic development.

6.4.2 Requirements

There were several expectations about what the purpose of the game would be. Although it was difficult to predict all of what eBay wanted from the project, the several meetings with the project sponsor at the beginning were helpful in determining a general idea of what purpose the game would serve. The project sponsor, Neel Sundaresan, stressed that he wanted the game to have a dual purpose: to be fun to play, and to obtain data useful to eBay Research Labs. *Memory Dash* meets both of these requirements in that it is a fun, catchy game to play, but it also has a section that mines user opinions and pairs them with the other players' opinions. Some examples of data being mined is users opinions on coolness of items, cost, and durability. Although this game compiles potentially numerous amounts of data, it is up to the Research Labs to determine how they will use this data.

6.5 Future

Memory Dash will be submitted to Apple's app store on Thursday, March 5, 2009. Once an app is submitted, Apple must approve it before it is distributed through the app store. This can take anywhere from one day to a month or more. Once *Memory Dash* is approved, it will be available to download for free from the app store. Any future changes or releases will be done by eBay Research Labs, including management of data gathered from the game. Although *Memory Dash* compiles and stores data received from players, it is up to the Research Labs to determine how they will interpret and utilize the data gathered.

7 Conclusion

Memory Dash was completed and will be submitted to Apple's app store on Thursday, March 5, 2009. The final result was a feature-complete and fun game that served its purpose of obtaining meaningful opinions from players about eBay items. Several elements helped the development team complete the project on time including using the game engine Unity Pro instead of coding in objective C from scratch and working with art tools that the art team already knew. In addition, because the game was thoroughly designed from the beginning and schedules were drawn up, there were clear milestones and deliverables expected every week.

Although the game was completed on schedule, there were many issues that occurred that hindered progress including memory issues on the iPhone that resulted in necessary optimization of art assets and code, difficulties in obtaining necessary software and hardware in a timely manner, and several multiplayer networking issues. The project team learned how to work around issues that came up and adapted the game development because of certain memory problems.

Although the development process included several issues that needed addressing, overall the game was very enjoyable and challenging to create in the specified time limit, eight weeks. It is expected that it will be accepted into Apple's app store and will be played by numerous users, creating copious amounts of meaningful user opinions for eBay. A website developed for the game is available at http://labs.ebay.com/memorydash/.

References

[1]. **Gwap.** ESP Game. *gwap.com.* [Online] [Cited: Febuary 19, 2009.] http://www.gwap.com/gwap/gamesPreview/espgame/.

[2]. **Hottrix.** iBeer 2.0. *hottrixdownload.com.* [Online] 2008. [Cited: Febuary 19, 2009.] http://www.hottrixdownload.com/secure/iBeer/iBeer.html.

[3]. **KinVibe.** Razor. *KinVibe.* [Online] 2009. [Cited: Febuary 19, 2009.] http://www.kinvibe.com/Razor.html.

[4]. **Apple, Inc.** About Xcode and iPhone SDK. *Apple.* [Online] 2008. [Cited: Jan 21, 2009.] http://developer.apple.com/iphone/download.action?path=/iphone/iphone_sdk_for_iphone_os_2.2__9m2621__final/iphone_sdk_for_iphone_os_2.2_readme_final.pdf.

[5]. —. iPhone OS overview. *Apple.* [Online] 2008. [Cited: Jan 21, 2009.] http://developer.apple.com/iphone/gettingstarted/docs/iphoneosoverview.action.

[6]. **Autodesk.** Autodesk Maya. *Autodesk.* [Online] 2009. [Cited: Jan 21, 2009.] http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=7635018.

[7]. **Adobe.** Adobe Photoshop CS4. *Adobe.* [Online] 2009. [Cited: January 22, 2009.] http://www.adobe.com/products/photoshop/photoshop/.

[8]. **Apple, Inc.** iPhone developer program. *Apple.* [Online] 2008. [Cited: Jan 21, 2009.] http://developer.apple.com/iPhone/program/develop.html.

[9]. —. Learning Objective-C: A Primer. *Apple.* [Online] 2008. [Cited: Jan 22, 2009.] http://developer.apple.com/iphone/gettingstarted/docs/objectivecprimer.action.

[10]. **Stonetrip.** iPhone Publishing. *Stonetrip.* [Online] 2008. [Cited: Jan 22, 2009.] http://www.stonetrip.com/shiva/publish-3d-game-on-iphone.html.

[11]. **Unity Technologies.** iPhone Publishing. *Unity.* [Online] 2009. [Cited: Jan 22, 2009.] http://unity3d.com/unity/features/iphone-publishing.html.

[12]. **Apple, Inc.** *iPhone Dev Center*. [Online] 2008. [Cited: Jan 21, 2009.] http://developer.apple.com/iphone/.

[13]. —. iPhone Developer Program Portal. *Apple*. [Online] 2008. [Cited: Jan 21, 2009.] http://developer.apple.com/iphone/download.action?path=/iphone/iphone_developer_program_user_guide/iphone_developer_program_user_guide__standard_program_v2.4.pdf.

[14]. **eBay, Inc.** *Color Palette*. 2009. eBay Brand Portal.

[15]. —. *Typeface*. 2009. eBay Brand Portal.

[16]. —. *Logo Guidelines*. March 2006. eBay Brand Portal.

[17]. **Unity Technologies.** *Unity iPhone Basics*. January 29, 2009. Reference Manual: Developing for the iPhone.

[18]. **Firewyre.** Poly Counts - How Other Games Count 'Em. *MMORPGMaker*. [Online] [Cited: January 29, 2009.] http://mmorpgmaker.vault.ign.com/phpBB2/viewtopic.php?t=597&sid=03bfd3ee60a3bc52717dec33537fee16.

[19]. **Apple, Inc.** Technical Specifications. *iPhone 3G*. [Online] 2009. [Cited: January 29, 2009.] http://www.apple.com/iphone/specs.html.

[20]. **Unity Technologies.** Sound. *Unity Manual User Guide*. [Online] [Cited: January 29, 2009.] http://unity3d.com/support/documentation/Manual/Sound.html.

[21]. —. iPhone Input. *Unity Script Reference*. [Online] [Cited: February 5, 2009.]

[22]. **eBay, Inc.** About the eBay Developers Program. *eBay Developers Program*. [Online] [Cited: Feb 18, 2009.] http://developer.ebay.com/businessbenefits/aboutus/.

[23]. **Lay Networks.** Comparative analysis - TCP - UDP. *Lay Networks*. [Online] [Cited: Feb 19, 2009.] http://www.laynetworks.com/Comparative%20analysis_TCP%20Vs%20UDP.htm.

[24]. **Kegel, Dan.** NAT and Peer-to-peer networking. [Online] July 17, 1999. [Cited: Feb 19, 2009.] http://alumnus.caltech.edu/~dank/peer-nat.html.

[25]. **Arn.** 2nd Generation iPod Touch Faster than iPhone. *Touch Arcade*. [Online] November 23, 2008. [Cited: Febuary 19, 2009.] http://toucharcade.com/2008/11/23/2nd-generation-ipod-touch-faster-than-iphone/.

Appendices

Appendix A – Art Asset List

Below is a list of all the different art assets that were necessary for the game. This list was created in the first week of production.

GUI

- Intro Screen
    - Logo (name of game)
    - Background art
- HUD (Game)
- High Scores
    - Background art

CITY MAP

- Tech district
    - Building 1
    - Laptop
    - Building 2
    - iPod
    - Building 3
    - Cell Phone
- Sports center
    - Building 1
    - Soccer Ball

  - Building 2
  - Trophy
  - Building 3
  - Baseball Cards
- Auto shop
  - Building 1
  - Tire
  - Building 2
  - Generic Car
  - Tube man

GAME PART A (card matching)

- Level Maps
  - 2 Tech levels
    - Apple/computer type store
    - Generic tech store (Best Buy)
  - 2 Sports levels
    - Flea market in basketball court
    - Sporting goods store (sports authority)
  - 2 Auto levels
    - Auto parts store
    - Car dealership (outdoor area)
- In-game
  - Card and card spinning animation
  - Shopping cart

  - Path effects

GAME PART B (card sorting)

- Background art
- (Use cards from game part A)

AUDIO

- Getting a match
- You win
- You lose
- Flipping a card
- Choosing/clicking sounds (sound feedback from choices)

IF WE HAVE TIME:

- GUI
  - Achievement icons for conquering areas
- Map
  - Other district areas (department store, clothing store, jewelry)
- Game part A
  - Variations on the shopping cart
  - Path variations (oil on the floor slows you down, etc… different paths have hazards)
  - Boosts/bonuses on paths (add speed, etc)
  - Other district areas mean 2 extra area levels per district.
- Audio
  - Time's up sound
  - Received an achievement

    - Menu/Game map music
    - Game A music
    - Game B music

Appendix B – Art Schedule

Below is the art schedule that was created for all 8 weeks of the project according to the art assets that needed completion (see Appendix A for the art asset list).

Week 1:

- Get software set up
- Unity Tutorials
- Whiteboard planning of game design
- Start working on Tech district and sports center map

Week 2:

- Complete Tech district and sports district for main map
- Compile districts into one map that's under 7000 vertices
- Create a sample level to be used in testing (Tech level - computer store)
- Create card and spin animation
- Compile all sample assets to create test game
- Complete levels:
    - Computer store (tech district)
    - Flea market on field (sports district)

Week 3:

- Create shopping cart model

- Complete auto district for main map
- Complete levels:
    - Car dealership (auto district)
- Obtain required audio files
- Logo/game name
- HUD design for game

Week 4:

- Complete level:
    - Generic tech store (tech district)
    - Sporting goods store (sports district)
    - Car dealership (auto district)
- Background art
    - Intro screen
    - Card sorting (Game part B)
- Finalize HUD

Week 5:

- Background art
    - High scores
- Path effects
- Do the items marked "if we have time"

Week 6:

- Playtest and polish
- Documentation
- Maybe do items marked "if we have time"

Week 7:

- Playtest and polish
- Documentation
- Maybe do items marked "if we have time"

Week 8:

- Playtest and polish
- Documentation
- Maybe do items marked "if we have time"

Appendix C – Tech Schedule

Below is the tech schedule that was created for all 8 weeks of the project according to the technical features that needed implementation.

Week 1:

- Get computers
- Investigate server
- Get Unity license
- Learn how to use Unity

Week 2:

- Create basic app
- Deploy basic application to iPhone

Week 3:

- Use iPhone input - clicking
- Use iPhone input - drawing/dragging

Week 4:

- City Selection Scene
- Make Cards pop up and stuff
- Place Cards on map
- Make input only start when near cart
- Show path block will take
- Block following input
- Path finding thing from input
- General matching game
- Moving avatar

Week 5:

- Card image server
- Make game multiplayer - Matchmaking
- Make game multiplayer - game synced
- Make game multiplayer - cards synced
- Add GUI
- Add Sounds
- Make game multiplayer - carts synced
- Use iPhone input - network interface
- Talk over network - Send numbers
- Talk over network - Receive numbers
- Talk to master server
- Fetch eBay data
- Sorting game

Week 6:

- High Scores List pull rankings
- Part B revised
- Results server
- Gameplay server
- Local Multiplayer
- Make standard controls
- Submit Scores

Week 7:

- Bugfixing

Week 8

- Bugfixing

Appendix D – Server setup details

Here are some helpful scripts required for the setup of php with gd on apache on Solaris.

The following libraries need to be compiled/installed:

- curl-7.19.3
- freetype-2.1.4
- httpd-2.2.11
- jpeg-6b
- libxml2-2.6.30
- mysql-5.1.31
- openssl-0.9.8j
- php-5.2.8
- readline-5.2

- zlib-1.2.3

For problems with ar not found:

PATH=/usr/local/bin:/usr/bin:/usr/sfw/bin:/opt/sfw/bin:/usr/xpg4/bin

Configuring mySQL:

```
CC=gcc CFLAGS="-O6" CXX=gcc CXXFLAGS="-O6 -felide-constructors -fno-exceptions -fno-rtti"
./configure --prefix=/home/kbickar/mysql --with-low-memory  --with-zlib-dir=/usr/local
```

Configuring and Compiling PHP:

```
./configure --prefix=/home/kbickar/php --with-gd --without-iconv  --with-
apxs2=/home/kbickar/apache2/bin/apxs --with-config-file-path=/home/kbickar/php  --with-
curl=/home/kbickar/curl --with-libxml-dir=/home/kbickar/libxml --with-jpeg-dir=/usr/lib --with-
mysql=/home/kbickar/mysql --with-zlib-dir=/usr/local/ --with-freetype-dir=/home/kbickar/freetype
```

Note that PHP will not compile if there are multiple mySQL installations in the path (specifically libmysqlclient.so). All the libraries listed will proble

Setup for mySQL:

Root Account: root:Zc7uFD9aDy

Account used by PHP (can only DELETE, INSERT, SELECT, and UPDATE) : erl:ea4dxKKk

Database Name: memory_dash

Players Table:

```
CREATE TABLE players (id VARCHAR(40), name VARCHAR(32), color ENUM('Blue', 'Red', 'Green', 'Yellow'),
games INT(4), points INT(5), PRIMARY KEY(id));
```

Game Table:

```
 CREATE TABLE games (game_id INT(6) AUTO_INCREMENT, state INT(2), time VARCHAR(10), district INT(2),
player1 VARCHAR(40), player2 VARCHAR(40), map INT(2), sort INT(4), sorted_1 VARCHAR(8), sorted_2
VARCHAR(8), points_1 INT(3), points_2 INT(3), item0 VARCHAR(20), item1 VARCHAR(20), item2
VARCHAR(20), item3 VARCHAR(20), item4 VARCHAR(20), item5 VARCHAR(20), item6 VARCHAR(20), item7
VARCHAR(20), PRIMARY KEY(game_id));
```