**3D, Gesture Controlled Arcade Games**

proposed by Evan Arroyo and Joshua Sims

Mario, Pac-Man, and Pokémon are pop culture giants. They rose to that status as 2D video games. Mario and Pac-Man became renowned in the early 1980s. Pokémon gained its fame in the mid-1990s. These games exploded in popularity because they were entertaining. After their inception, they developed another attractive quality: nostalgia. Eventually, they became cultural icons. This is true for many other 2D games. Despite the continual release of popular 2D games, 3D games are trending much more. Many 2D games have been adapted into 3D games. This is not necessarily because those 2D games were so lacking that they needed the change -- the 3D adaptations are often created out of curiosity. However, it is not unusual for the 3D adaptations to explode in popularity just like their classic counterparts. Aside from the graphical shift from 2D to 3D, there has been a popular shift from gamepad/keyboard input to gesture input. Our goal is to create a suite of 3D, gesture controlled adaptations of arcade games. We will start with the simple, well-known game Pong.

It is important to consider how a game will be played before considering how to create it. One can tailor a game to the platform(s) which can host the game’s features or one can tailor a platform to accommodate a game’s features. We have chosen to use an existing platform because Pong, and the other games in the suite, will not involve anything which may exceed the abilities of existing platforms. We could use WebGL as the platform for Pong and its partnered language Three.js as the programming language in which to write the code, however, the quality of gesture control on WebGL is determined by the camera used. Typically, that camera is whatever is built into/attached to the laptop/computer and such a camera is likely not intended for gesture control (just video streaming) and would be difficult to utilize for such. There do exist cameras intended for gesture control and they are typically sold bundled with an appropriate entertainment system. Those bundles feature the most practicality for our game. There are three primary contenders which are capable of hosting a 3D, gesture controlled Pong: Playstation Move, Wii U, and the Kinect for Xbox One. We have chosen the Kinect for Xbox One because it is readily available to us (no expenses) and it senses motion just as well as, if not better than, its primary competitors, the Wii U and the Playstation Move. We have investigated technologies for the creation of games hosted on the Kinect and there were two primary contenders (according to popular arguments).

Unity 5 and Unreal Engine 4 are the game engines which we have considered for our development environment. Both of these game engines are basically free. Unreal Engine 4 specifies a 5% royalty fee as its only expense and Unity 5 offers free use of most of its features. Unity 5 is more popular than Unreal Engine 4 and as a result has more resources for learning it. Many very successful games have been created with Unity 5 and likewise with Unreal Engine 4. Unity 5 features “industry-leading multiplatform support” -- including support for the Kinect (Unity3d.com, 2016). Unreal Engine 4 features far less multiplatform support; a middleware plugin is required to use Unreal Engine 4 with the Kinect. Both Unity 5 and Unreal Engine 4 are capable of rendering intense graphics in real time while maintaining smooth gameplay. Ultimately, we chose Unity 5 because of its greater popularity and versatility: more resources are available for it, more peers/employers are likely to be involved with it, and it features exceedingly good multiplatform support. After deciding on Unity 5, we had to choose which supported programming language to use.

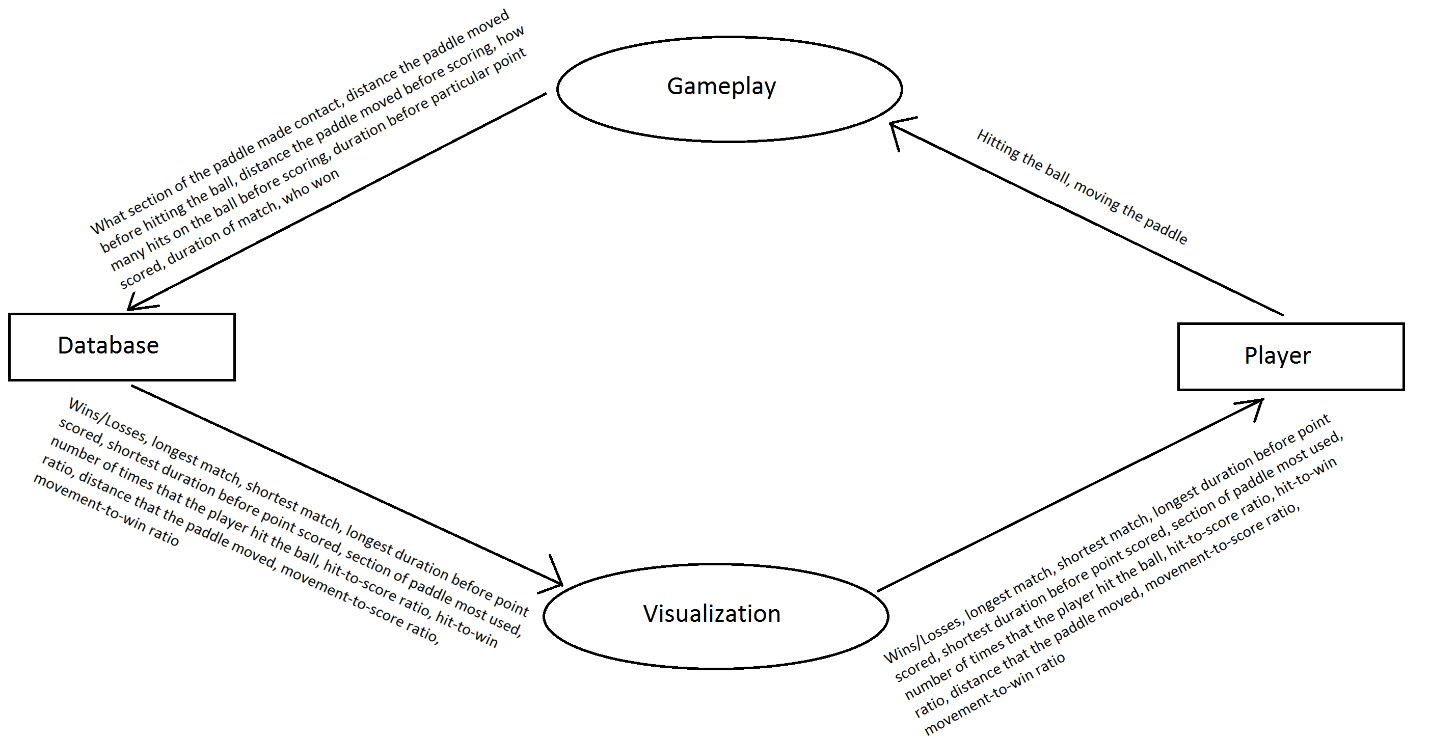
Unity 5 supports three programming languages: Boo, UnityScript (basically JavaScript), and C#. However, we decided to reject Boo after reading the following report from Unity: “as so few people use Boo, and the resources required to support it in the docs are not negligible, we’ve decided to drop support for Boo documentation for the Unity 5.0 release and use our resources in a more constructive way” (Unity3d.com, 2016). Researching the difference between C# and UnityScript in relation to Unity 5 did not reveal any significant advantages or disadvantages for either language. Our decision was influenced by our lack of experience with C#: it seems that none of the classes at Western Carolina University require the study of C# whereas there are several which require the study of JavaScript. Thus, we chose to use C# for the sake of increasing our knowledge and versatility. Having decided how to write our code, we then investigated how we should store and update it.

Rather than move our files around our local file directories and back them up to separate, local drives, we have decided to use version control technology to secure our files. “Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later” and it saves developers a large amount of time by doing so (git-scm.com, 2016). A developer could choose to never utilize version control and instead overwrite his files constantly. But, if he were to encounter the need to use an earlier version of some file, he would have to rely on his memory to replicate that earlier version. Unless that developer had eidetic memory, he would then spend a large amount of time reinventing the file that he needs. This sort of scenario usually comes about when a developer is trying to save time by neglecting version control. However, current version control technology is so intuitive and quick to use that the amount of time saved by not using version control is so proportionally miniscule to the amount of time consumed by the consequences that it is not worthwhile to consider *not* using version control. We therefore decided to use version control software -- particularly, GitHub. We chose GitHub because it is free, widely praised, and unfamiliar to us. It is important for us to learn to use GitHub not only because it is very effective, but also because so many projects now reside in it. After deciding to use GitHub, C#, Unity 5, and the Kinect, we reexamined our decision to start our 3D, gesture controlled game suite with Pong as opposed to some other 3D, gesture controlled game.

To create a game with an original design would be interesting, perhaps more rewarding, and likely more impressive. However, the ­process of designing a game is time intensive, especially when the goal is to maximize originality. The later implementation of a design can reveal significant faults which demands revision of the appropriate parts or of the design as a whole. The variability that follows originality is the crux which turned us toward the creation of an adapted game. The 2D games which appealed to us most were popular, yet fairly simplistic. Mario, Tetris, Centipede, Space Invaders, Pong, and Breakout were all considered. We considered the difficulties of creating a 3D, gesture controlled adaptation for each. Mario and Space Invaders were likely to require more graphical work than the others because the models in those games are much more detailed than the models in the other games. Classic Centipede and Tetris seem to lack any features which would involve a challenging implementation of physics. Breakout does involve a challenging implementation of physics: the movement of the ball. However, Pong involves the same implementation of physics as Breakout and it also features simpler gameplay. We ultimately chose Pong because it was the simplest game that was still challenging enough to justify a semester of work dedicated to creating it. After we complete the minimum viable product for the Pong game, we will focus on gathering statistics relevant to the gameplay.

We will calculate statistics produced by the player’s (or players’) activity and we will allow the player(s) to view that data in-game. The statistics will be relative to particular pairs of competitors (John vs. Jane or John vs. CPU). Furthermore, where applicable, each statistic will have an overall version as well as match-specific versions and point-specific versions. The featured statistics will include a hit to win ratio (how many times the ball was struck by the paddle before the relevant player scored), total wins, total losses, longest game, shortest game, longest time for a point scored, shortest time for a point scored, the section of the paddle most often used, and how much that player moved his paddle (expressed as distance). Other statistics such as favorite paddle color, favorite background, and favorite sounds will be calculated if such customizations are included after the minimum viable product is created.

A data-flow diagram for our game is included below.



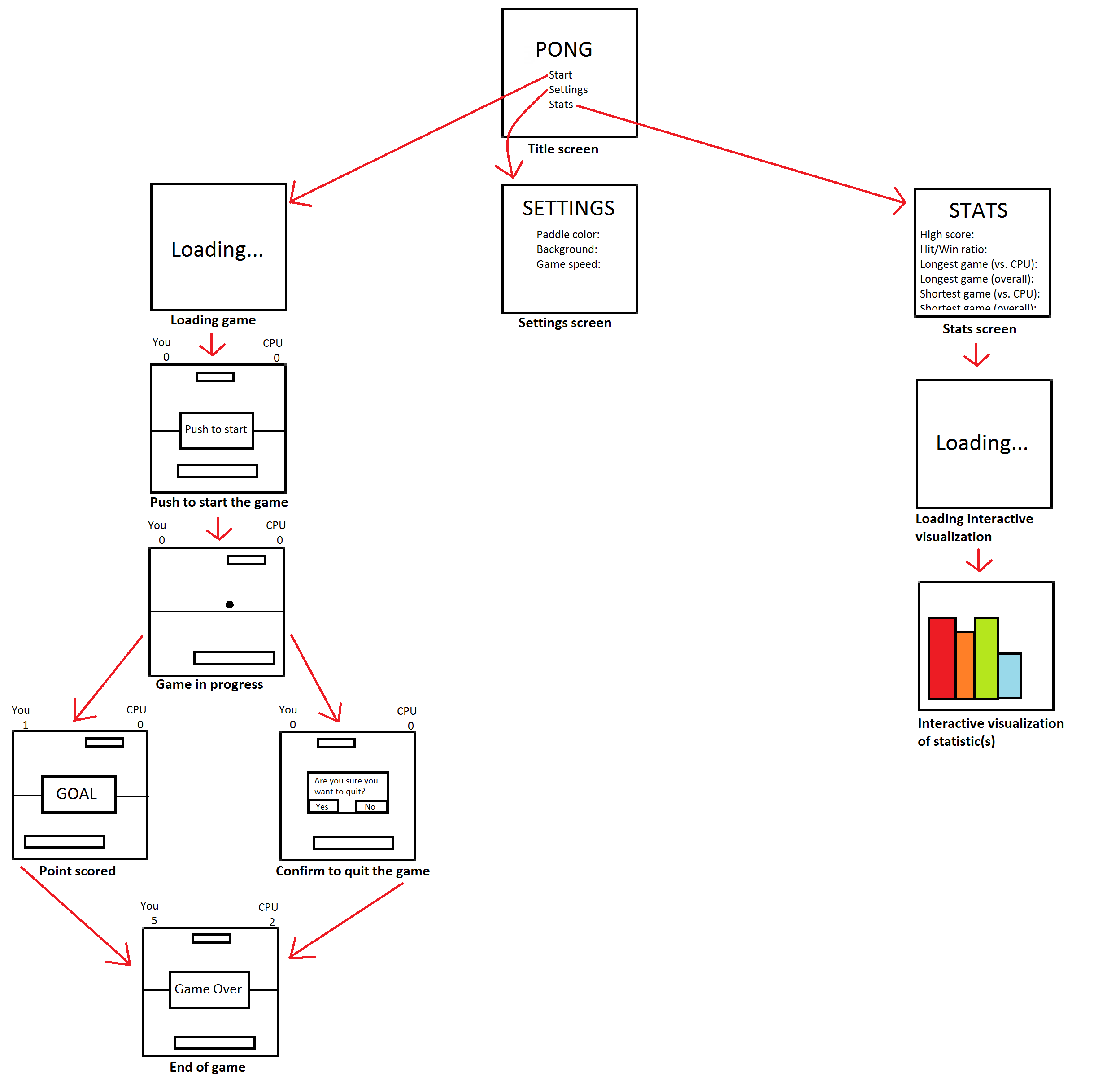
As shown in the above data-flow diagram, the source of our game’s data is the player. The player, by interacting with the game, sends data to our database. The player sees some statistics describing his performance whenever he scores a goal and whenever he navigates to the stats screen.

Game developers incorporate the collection of game data for the sake of furthering the players’ motivations for playing the game, increasing profits, and debugging the game. Either by curiosity or competitiveness, players are commonly motivated to view their statistics, to beat a high score, or to unlock achievements and/or abilities related to their stats. Games which utilize networking can benefit even more from statistics generation. Statistics generation is useful for match-making players of relatively equal skill and for determining trends. Trend data can be a boon for marketing. Some common trend data includes how long the average user spends playing the game, the time at which most users are online, and what kind of user is likely to do x, y, or z. Perhaps the greatest use of game data is that of debugging. Some problems are better studied from a long term perspective -- statistics allow developers to do that. Developers may never notice a problem with the game if it is so subtle that it is only apparent when viewing data gathered from long term usage. Networking is necessary to maximize data collection, but networking can be very problematic.

Network issues can make it impossible or very difficult to collect accurate data and networking demands much more effort. Here are some common networking difficulties relative to data collection: ensuring that data is coming from and going to the appropriate place; acquiring data without data loss caused by disconnections; ensuring that data is not being generated by robots which are manipulating the game; and investing in a place to store the possibly massive amounts of data collected from hundreds, thousands, or millions of users. Our game, however, will not use networking -- it will be offline. The users for our game will not likely be disappointed by the lack of online capability: they will be expecting a 3D, gesture controlled adaptation of the classic, offline Pong experience.

Pong is not a complex game, but it does have standards. Smooth playthroughs are expected every time; the game should never lag. The gestural control should be intuitive and totally responsive. The 3D graphics, and the sound, if included, should be pleasant, though not necessarily highly detailed. The average user wants to be challenged, but not dominated by the AI. If split screen functionality is included, it should function well and the screen should be split in a way that is comfortable for both players. If statistics are promised, then accurate statistics should be collected and delivered. If various settings are mutable, a wide range of choices should be offered. Loading times should be minimal. Navigation across menus should be intuitive. Overall, the user should be satisfied that he has played a fun game of 3D, gesture controlled Pong.

To illustrate this vision to some extent, a user interaction flow diagram is shown below.



As shown in the above user interaction flow diagram, the user will navigate to the game, to the settings, or to game-related statistics via the title menu. Other transitions are defined by the arrows connecting the various screens.

For further illustration of our plan, we have also included a MoSCoW analysis, represented below as a table.

|  |  |  |  |
| --- | --- | --- | --- |
| Must have | Should have | Could have | Would like but will not have |
| The game is playable, the 3D graphics and gestural control are implemented, the CPU is responsive, the statistics are accurate, and interactive visualizations of the statistics are available and accessible to the user. Furthermore, we must have a database that captures game statistics per player (statistics such as the hit-to-win ratio, the section of paddle most used, and the distance that the paddle moved). | Error handling; navigation menus; attractive sounds effects (no music); attractive 3D graphics; intuitive and totally responsive gestures; very capable AI; split screen capability; more statistics; customizable background, paddle color, and sound effects. | Highly detailed 3D graphics, more precise gestures, enhancement features which do not appear in classic Pong (power-ups, random events, etc.). | Networking (local or online multiplayer). |

To ensure that our game is in a stable condition which does not hinder the enjoyment of its features, we will develop this game via test driven development and, before each additional feature, we will perform unit tests. These practices will ensure that each additional feature is an enhancement to an already stable product. Testing for the gestural control will mostly be done via playthroughs of the game. Much of our knowledge which will contribute to the quality of our game comes from our experience at Western Carolina University.

CS 253 covers the use of unit tests and test driven development -- this will be highly valuable since unit testing and test driven development will be at the forefront of our product’s stability. CS 253 also covers the use of GitHub, helping us to use that resource for version control. CS 150 and CS 151 have strengthened our skill with Java and that skill should translate into the similar language C#. We have studied JavaScript in several CS courses, including CS 253, CS 263, and CS 352. That familiarity with JavaScript will be useful if C# is not sufficient in meeting our needs. CS 363 will help us to create UML diagrams. Although neither of us will be taking CS 361 thissemester, we will both be taking it during the coming semester. We can use knowledge gleaned from that class to further enhance the 3D graphics in our game. CS 370 has furthered our understanding of how processes/threads interact and what the operating system is/how it functions – that knowledge should be useful for any program. CS 350 has made us more aware of optimality in regard to the low-level instructions beneath the programming language. Such awareness should aid our judgment when optimizing this game. However, before optimization, we will focus on developing a minimum viable product.

The timeline for this project, with a focus on developing a minimum viable product, is

shown in the table below.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sept. 8** | **Sept. 15** | **Sept. 29** | **Oct. 13** | **Oct. 27** | **Nov. 10** | **Dec. 1** |
| **Discuss project proposal.** | **Basic models and background finished. Customizable camera angle implemented.** | **Ball and paddle physics completed.** | **Working AI implemented.** | **Gesture control implemented.** | **Basic statistics collected and checked for accuracy.** | **Navigation menus implemented and statistics viewable in-game. Other features pursued if surplus time exists.** |

Works Cited

*Git-scm.com*. GitHub, 2016. Web. 11 Sept. 2016. <https://git-scm.com/>.

*Unity3d.com*. Unity Technologies, 2016. Web. 11 Sept. 2016. <https://unity3d.com/>.