3D, Gesture Controlled Pong

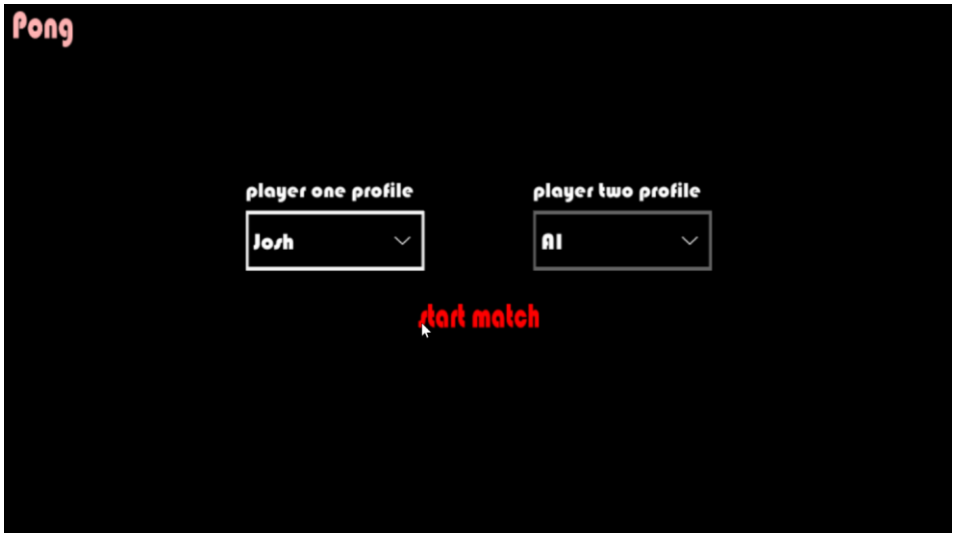
Evan Arroyo and Joshua Sims

~~We want to progress our skills as software engineers. Impressive, purposeful software is a benchmark of good software engineering – it is the benchmark we are working toward. More specifically, we are working toward making a computer game because we are both interested in developing games throughout our careers. The two primary attributes which form the focus of our project's appeal are 3D graphics and gestural control.~~ Immersive, 3D games are ~~trending more than 2D games~~becoming increasingly popular. Part of providing an immersive experience is ~~and~~ gestural control~~,~~ and, given its intuitive and realistic nature, ~~is~~will likely ~~to~~ be the prominent user interface as virtual reality and augmented reality advance in popularity (Garber). ~~Upon the inception of o~~Our project involves~~, we decided to~~ developing a suite of 3D, gesture controlled games. During the Fall Semester of 2016, we created a Pong derivative that features 3D graphics and gestural control of the player's paddle. Toward the end of the last sprint (a two-week interval of development) of that semester, we realized that extending the game we had made would be much more worthwhile than developing other games~~—r~~. Rather than finish a collection of two or three satisfactory games, we ~~want~~aim to produce a finished, polished ~~a~~ game that entices and intrigues the player. We satisfied the requirements of classic Pong but we did not achieve an impressive game. To do that, we will reshape our game. It will feature 3D movement of the paddles and ball and we will introduce randomly (do all of them need to be random?) occurring additive effects such as obstacles, objects that trigger ball or paddle modifications when struck, goal zone segments that grant extra points if struck, disabled goal zone segments, and wind. These additional features (and others which we have yet to conceive) are our focus for the Spring Semester of 2017.

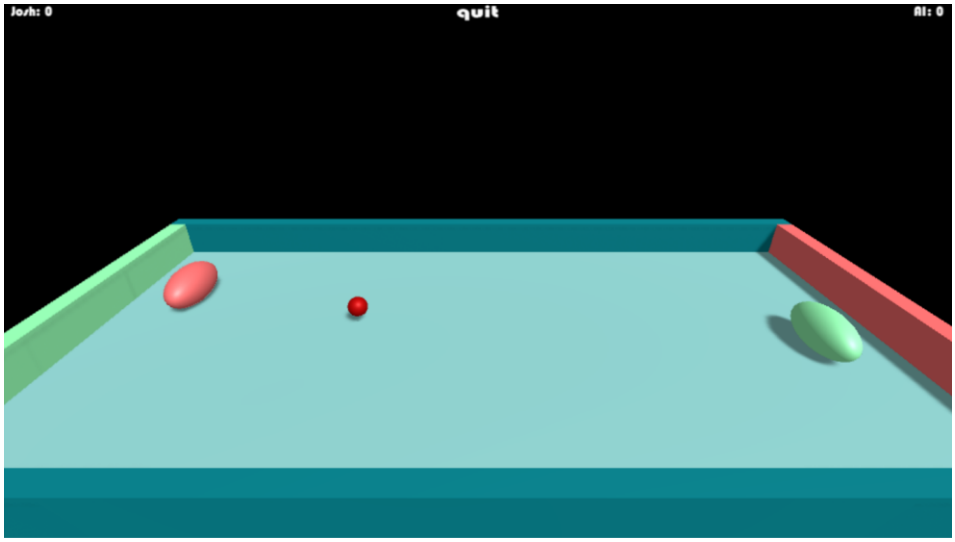
Our efforts from last semester produced a stable, satisfactory Pong experience with few bugs. ~~A few quirks remain, but the game neither crashes nor lags.~~ The paddles and ball behave as expected except for an infrequent bug in which the ball suddenly accelerates after striking a paddle. The game proceeds suitably between each goal and ends as it should. Statistics are accurately recorded and are presented conveniently to the player; the UI is easily navigable; and the gameplay is simple, but sufficiently challenging. The series of screenshots presented below illustrate the product of last semester's work.

This is the root menu for the entire suite. We named the suite "3DG Arcade." This root menu as well as any mention of "3DG Arcade" will be removed from the project because our focus has shifted away from developing a suite – we will only be extending Pong, not making additional games.

This is the root menu for Pong. The “3DG Arcade” button in the top left corner of the screen will be removed and the “settings” button will be enabled once we implement the settings menu. From here, the user can click the “play” button to play the game or the user can click the “statistics” button to view statistics pertaining to a specific player.



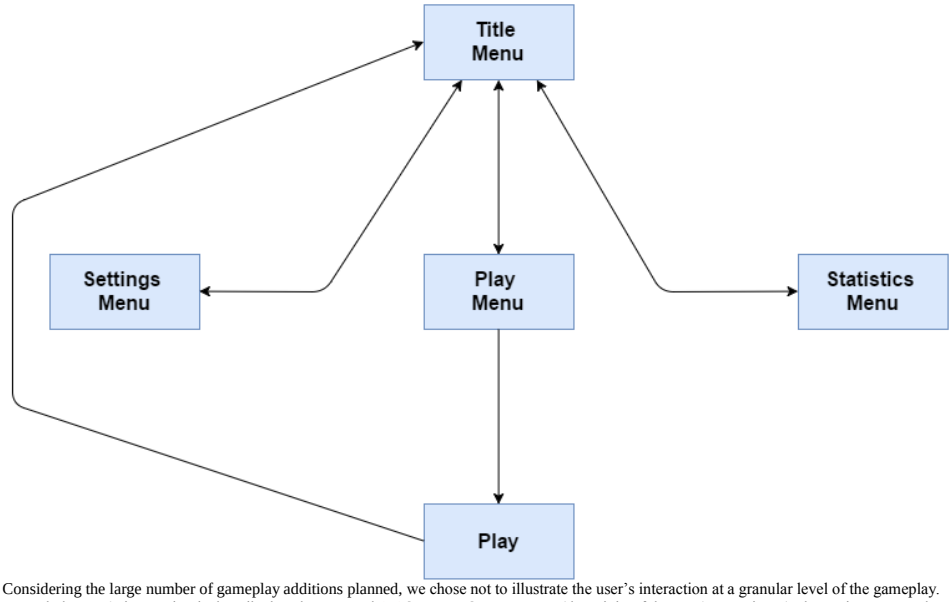
This is the play menu – the screen that the user encounters after clicking the "play" button at the Pong root menu. From here, the user selects, or creates, a profile which allows statistics to be gathered for that player during gameplay. After choosing a profile, the user may click the "start match" button to play the game or the player may click the "Pong" button in the upper left corner to return to the Pong root menu.



This is a snapshot of the gameplay. Victory is awarded to the player who reaches 5 goals first. After a player wins, a "game over" sign is displayed and the user is returned to the Pong root menu.

This is the statistics menu. The user can specify a profile by selecting one from the dropdown menu which is to the left of the statistics listing. Upon selection of a profile, the statistics relevant to the selected profile are loaded and displayed. The user may scroll up, down, and side to side to browse the statistics listing.

We have a plethora of improvements planned. We plan to implement a settings menu from which the player can adjust qualities that affect gameplay. Those gameplay affecting qualities, which we will generally refer to as “additive effects”, will be our primary focus. Aside from the implementation of a settings menu and the many additions to the gameplay, the flow of user interaction will remain the same as before. To illustrate this, we have included an interaction flow diagram below.



Considering the large number of gameplay additions planned, we chose not to illustrate the user's interaction at a granular level of the gameplay. Instead, the user's interaction is described as the traversal (??) between the root menu (the origin of the user's experience), the settings menu, the play menu, the gameplay, and the statistics menu.

The changes to gameplay will include, but are not limited to, 3D movement of the paddles and ball and randomly occurring additive effects including obstacles which impede the ball and incorporeal (but still visible) objects that, when hit by the ball, increase or decrease the ball's speed, increase or decrease a paddle's speed, deactivate or activate a player's goal segment, increase the number of points awarded for scoring on a particular goal segment, trigger wind which influences the ball and the paddles' movement, or multiply the number of balls in the arena (sentence a little too long/awkward). Two player capability, background music, sound effects, and profile deletion may be implemented as well – those features, however, are lower priority than the aforementioned gameplay factors. We have included a MoSCoW analysis below which describes the changes to be made and the respective priority of those changes.

Must have	Should have	Could have	Won't have
• 3D arena • 3D movement of paddle and ball • Profiles (and profile creation) • Statistics menu (featuring accurate statistics) • Navigable UI • Settings menu • Gameplay derivative of classic Pong • Artificially intelligent opponent • Gestural control over UI and paddle via Kinect • Obstacles (random and static) for the ball to collide with • Elements which, when touched by the ball, modify the ball, paddle, or arena (including goal zones and barriers)	• Intuitively navigable UI (and clear visibility of UI elements) • Intuitive physics • Artificially intelligent opponent that challenges, but does not overwhelm, the player • Two player gameplay (neither player is artificially intelligent) • Profile deletion	• Sound effects • Background music • Choice of multiple camera angles • Official support on systems other than PC coupled with Kinect	• Networking • User insertion of music, sound effects, or graphics (the user cannot upload a sound to be played each time the ball collides with something nor can the user upload a picture and use it as the skin for their paddle) • Data encryption • Standardized method of importing/exporting settings or statistics • Persistent data technology other than the BinaryFormatter class in C# • Data visualization (other than descriptive text)

To assist us in completing our project, we decided to use a game engine because the level editor therein would allow us to quickly assemble the significant, but simple parts like the arena and thereby enable us to give more time and focus to the much more complex parts like the behaviour of the UI and gameplay elements and the implementation of the gestural control (awkward). We could have developed the 3D graphics and behaviour without a game engine, by using, for example, WebGL or Three.js (a simplifying wrapper for WebGL), but we would sacrifice a great amount of productivity by doing so. WebGL and Three.js do provide greater control over the optimization of code, but our project should not be computationally demanding

enough to merit such optimization. Furthermore, game engines have been proven to be capable of building (the engine doesn't build the software) very complex software without impairing its ability to run smoothly (Unity3d.com). There are many affordable and effective game engines available and after investigating the most prominent of them, we chose Unity. Other game engines, including Unreal Engine – Unity's closest competitor (in terms of popularity) – would likely meet our project's requirements, but we want to familiarize ourselves with Unity because it seems to be the *most* capable of handling more demanding projects~~— projects which we will encounter in our future as software engineers~~. Unity features unmatched popularity and multiplatform support and it can render incredibly complex and realistic graphics, evidenced by the short film *Adam* which was rendered in Unity in real time (Unity3d.com). We run Unity atop Windows as opposed to another operating system because the Kinect, our choice of motion capture technology, is not officially compatible with any other operating system (Developer.microsoft.com). We chose the Kinect ~~as opposed to~~instead of other motion capture technologies because the Kinect is one of the most well~~-~~established motion capture technologies available. It is also among the most affordable. Before choosing the Kinect, we studied several other motion capture technologies. Some, such as the "Teslasuit" by Tesla Studios or the "HaptX" technology by AxonVR, provide haptic feedback to the user. These technologies are particularly interesting because haptic feedback is likely to be involved in the growing virtual reality industry (place citation here)~~— an industry which we would like to be a part of~~ (Adams and Hannaford). However, implementing haptic feedback would be unnecessary work for our project ~~–~~ because Pong would, at most, invoke a~~n~~ ~~insubstantial~~minimal level of haptic feedback equivalent to playing a game of table tennis. Other motion capture technologies which rely on handheld devices, such as the Wii and PlayStation Move, are less appealing because they only

capture hand and, to some extent, arm movement. Those technologies will likely be replaced by cameras and other motion capturing devices which track the user's total~~ity of the user's~~ body. Therefore, although the handheld device technologies such as the Wii and PlayStation Move are well suited to our project – which only involves gestures of the hand – they are not likely to be useful for immersive, virtual reality projects which require tracking of the entire body. ~~— projects which are likely to define the coming generation of entertainment.~~ We looked for motion capture technologies which track the entire body and we discovered the Kinect, Perception Neuron, OptiTrack, and Xsens. The aforementioned technologies were the only full body tracking solutions which were fully developed and available for use. ~~The Kinect was the only one which cost less than $1,000 – more specifically, it cost us nothing. Furthermore, the Kinect was easily within our reach: we were able to obtain the device within the first week of the project – the devices associated with the other technologies would have required shipping over a variable period of time.~~ Ultimately, we chose the Kinect because it was the most affordable, the most readily available, and the most well-documented of the motion capture technologies we researched. Microsoft is the company which developed the Kinect. Microsoft has provided full documentation of the Kinect API (place citation here), although very few examples exist ~~of that API in action~~ (Developer.microsoft.com). The programming languages supported by the API are C#, JavaScript, and C++. Unity, however, does not support C++ and it technically does not support JavaScript – it supports UnityScript which is significantly, but not completely, different from JavaScript. UnityScript is a programming language which is relevant only to Unity. Skill in UnityScript would translate well to JavaScript, but our team ~~has~~ already ~~been well exposed to~~has experience with JavaScript through several of our Computer Science courses. ~~— w~~ We would prefer to develop our proficiency in C# since neither of us had been exposed to C# prior to this

project. To be sure that UnityScript was not markedly better than C# for our project, we researched the differences between the two as they pertain to performance and ease of use. We discovered that the only significant difference between them is that far more people use C# with Unity than UnityScript (Unity3d.com). This discovery solidified our decision to use C#. C# also has a great, officially supported IDE (Integrated Development Environment) – Visual Studio. Visual Studio offers code completion, customizability, and debugging tools. ~~Furthermore, Unity uses Visual Studio as the default editor whenever a C# script is opened. It is also the IDE that is used in Unity debugging tutorials (Unity3d.com).~~ Unit testing and integration testing in Visual Studio is extraordinarily simple and allowed us freedom from unofficial extensions or middleware for testing. ~~Aside from testing the gameplay and the user interface,~~

~~we test t~~The accuracy of the statistics gathered during gameplay is also tested. The gathering of statistics necessitated data persistence. ~~The problem of data persistence in relation to~~ Unity surprisingly has no officially supported solutions regarding a DBMS (database management system). Particularly, we were seeking to pair SQLite with Unity because SQLite is well~~-~~suited for non-distributed applications such as our own (Junyan, Shiguo, and Yijie). However, we did find a non-DBMS solution in an official Unity tutorial ~~regarding data persistence~~ (Unity3d.com). The tutorial recommended using BinaryFormatter, a C# class, to serialize objects into binary data instead of JSON or XML because binary is more secure than JSON or XML which both store data as plain text.

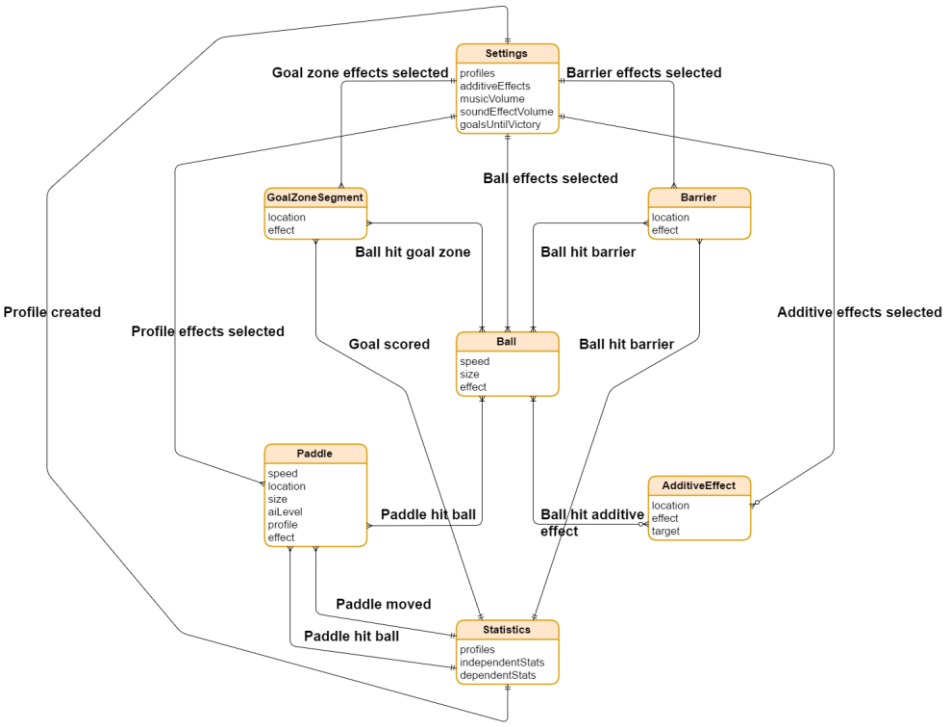
To back up and manage our project's source code, we used Git. Git is superior to its competing version control technologies. Git is much faster than Subversion because Git does not load and apply the differences between the versions of a file in order to produce the queried file. Instead, Git produces the queried version of the file as it was originally submitted. This requires

the storage of each version of a file in its entirety (rather than just the differences between it and the previous version). Fortunately, such a reliance on storage space is not a serious issue since data storage is very affordable. CVS, another competitor, does not feature support for atomic operations and therefore the integrity of the stored data is not guaranteed. Git, however, *does* support atomic operations. Mercurial is another option. We chose Git instead of Mercurial because Mercurial is not as well established – we are more likely to use Git in the workforce and therefore should become as familiar with it as possible (Rawson) – not sure what is being cited here.

Last semester's work on this project helped prepare us because we became familiar with Unity, C#, Visual Studio, Git, GitHub, Kinect, our habits as a team, how to write a project proposal, and how to create diagrams. CS 150 and 151 gave us a strong foundation in Java, and thereby a strong foundation in C#. CS 453 will help us realize when and how a DBMS is used – this is valuable even if we do not use a DBMS in *this* project because we can visualize how a DBMS would be used in a similar project. CS 253 taught us Git commands and design patterns which we have used and plan to use frequently. CS 263 helped us develop habits that promote healthy project management. CS 350 helped us appreciate low level instructions and, coupled with CS 351, helped us ensure that we keep things at least sufficiently ~~optimal~~efficient. CS 370 informed us of atomic instructions, issues related to concurrency (race conditions), and the consequences of issuing system calls. ~~–~~ ~~a~~All of these ~~matters~~ are important to consider when writing code. CS 361 will increase our familiarity with Unity and expose us to valuable insights regarding graphics in general. Our calculus courses (as well as physics courses) taught us how to efficiently solve problems relating to gameplay physics~~ – this is useful when writing code pertaining to the physics of the gameplay~~.

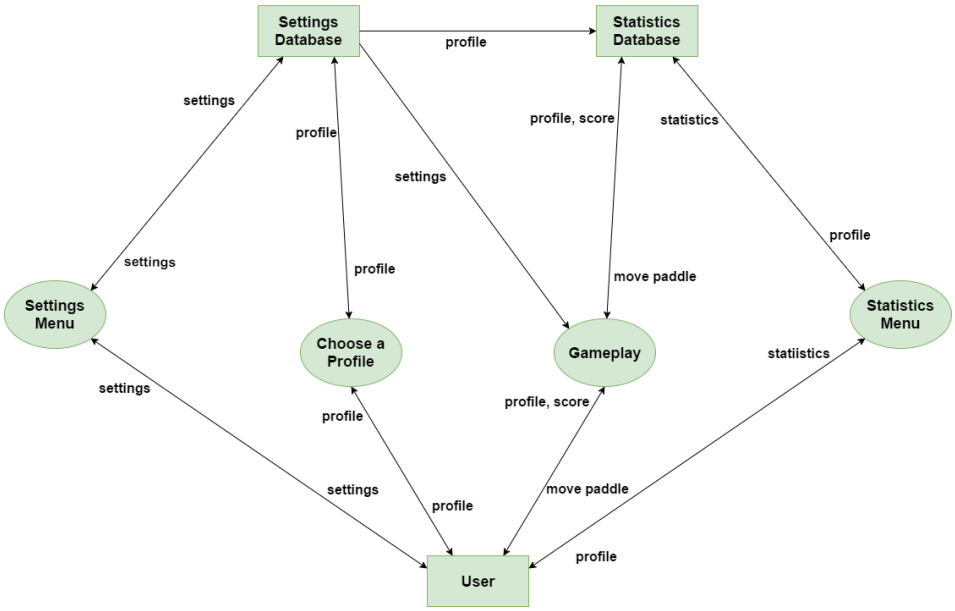
Our formal education has assisted us greatly, but we have much more to learn independently. There is much about Unity that we have yet to understand ~~or~~ and discover. ~~We do not feel lost in Unity, though; learning Unity has been a challenge, but not a burden.~~ Maximizing our project management efficiency requires a more thorough understanding of Git, GitHub, and ZenHub (the project management extension for GitHub). If we perform continuous integration tests within Visual Studio, we want to learn how to automate that process. And lastly, perhaps the most imposing challenge will be researching the Kinect to achieve the functionality that we want. Microsoft provides an API for the Kinect, but provides very few examples of how different elements described in the documentation interact. Therefore, programming the behaviour of objects controlled by the Kinect is almost entirely a trial and error effort. Thankfully, the integration of the Kinect is somewhat independent of the operation of the rest of the game so if we fall behind on that part of the design, it is not likely to shut down any other part of the design.

The design of the gameplay is illustrated by the following ER diagram.



This gameplay-centric diagram uses Crow's Foot notation. The term "additive effects" refers to the many elements which influence the gameplay.

The statistics that we collect and display to the user are not required for satisfactory gameplay, but they do fuel the user's motivation and curiosity. We intend to collect data which describes a player's skill, habits, strengths, and weaknesses. Currently, we record displacement (of what?), hits, wins, losses, goal segment scored upon (this is a little awkward), and combinations of the aforementioned. We may, as a part of the expansion, include other statistics, such as the amount of time played during a match. ~~We have included a~~A data flow diagram is shown below below to illustrate the flow of data throughout Pong.



The specific details regarding the statistics and settings data are not enumerated in this diagram. Instead, the transmission of data has been abstracted to exclude granular information about the data. Illustrated in this diagram is the generation and transmission of settings data (which influences gameplay) and the generation and transmission of statistics data. The user generates settings data by selecting certain options at the settings menu. That data is persisted in the settings database and is loaded before gameplay begins so that it may influence the gameplay appropriately. The recorded settings data can be viewed in the settings menu. Statistics data is generated during gameplay by the user who moves the paddle. It is then saved in the statistics database after a goal is scored by either player. The recorded statistics data can be viewed in the statistics menu.

(This paragraph should be part of the previous paragraph). The data described ~~is typically~~could be used for record keeping ~~(for marking personal achievements as well as for matching or surpassing the achievements of others)~~, matchmaking, prize winning, competitive play, self-improvement, or just for curiosity. Such data is most ~~potent~~ useful in the context of distributed applications (applications which transmit data across a network). However, networking and related topics, such as security, are beyond the scope of our project.~~it is difficult to network data. Our project will not deal with networking: transmitting data and certifying data across a network would introduce unnecessary complications. Data security is another issue that we will not delve into. Effectively ensuring that data is not tampered with or accessed by~~

~~unauthorized users requires data encryption. We would like to learn more about data encryption, but it is not necessary for the scope of this project.~~

We will test our system via playthroughs (user acceptance testing) and we will apply TDD (test-driven development), unit testing, and continuous integration testing, but we doubt that testing will be as rigorous as we would like it to be considering that there are so many other challenging goals which we have set for ourselves. The timeline for accomplishing those goals is shown below.

~~The timeline for the Spring Semester of 2017 is shown below.~~

Jan. 24th	**Feb. 7th**	**Feb. 21st**	**Mar. 14th**	**Mar. 28th**	**Apr. 11th**	**Apr.25th**
• 3D arena built • 3D movement of ball (and paddles) • 3D gestural control of paddle via Kinect	• Ball resets after goal • Game over after a number of goals specified by the user • Artificially intelligent opponent • GUI interaction via Kinect • Statistics menu • Settings menu	• Additive effects (such as randomly occurring obstacles, wind, and ball-modifying elements)	• Additive effects	• Additive effects	• Additive effects • Code freeze *Time permitting* • Two player capability • Player two paddle controlled via Kinect • Background music • Sound effects • Code refactorization	• Final report • Presentation

Works Cited

Developer.microsoft.com. Microsoft, n.d. Web. 26 Jan. 2017.

L. Garber, "Gestural Technology: Moving Interfaces in a New Direction [Technology News]," in *Computer*, vol. 46, no. 10, pp. 22-25, October 2013. doi: 10.1109/MC.2013.352

L. Junyan, X. Shiguo and L. Yijie, "Application Research of Embedded Database SQLite," *2009 International Forum on Information Technology and Applications*, Chengdu, 2009, pp. 539-543. doi: 10.1109/IFITA.2009.408

Rawson, Rob. "2016 Version Control Software Comparison: SVN, Git, Mercurial." Biz30.timedoctor.com. Biz 3.0 and Time Doctor, 21 Dec. 2016. Web. 18 Jan. 2017. <https://biz30.timedoctor.com/git-mecurial-and-cvs-comparison-of-svn-software/>.

R. J. Adams and B. Hannaford, "Stable haptic interaction with virtual environments," in *IEEE Transactions on Robotics and Automation*, vol. 15, no. 3, pp. 465-474, Jun 1999. doi: 10.1109/70.768179

Unity3d.com. Unity Technologies, n.d. Web. 18 Jan. 2017.

Overall, well-written and you addressed all required topics. However, there seemed to be three areas that could use improvement:

1. Organization: The ordering could have been better. For example, a better ordering would have been discussing last semester, the platforms you chose, why you chose them, and the MoSCoW analysis for this semester. Currently, you have the platforms you chose (last semester) after what you have planned (this semester). Where did your MoSCoW analysis come from?

   Also, the section about your formal education would have been better at the end.

   The paper ended abruptly.

2. References: This was version was a substantial improvement. There were minor mistakes in the placement of the citation in the sentence.

3. Sentence structure: Some of the sentences were awkward. Additionally, there were sentences that were not needed and took away from the main point of your paper.

Grade: 87

Formatted: Indent: Left: 0.5"

Formatted: Indent: First line: 0.5"

Formatted: Indent: Left: 0.5"