3D, Gesture Controlled Pong

~~proposed by~~ Evan Arroyo and Joshua Sims

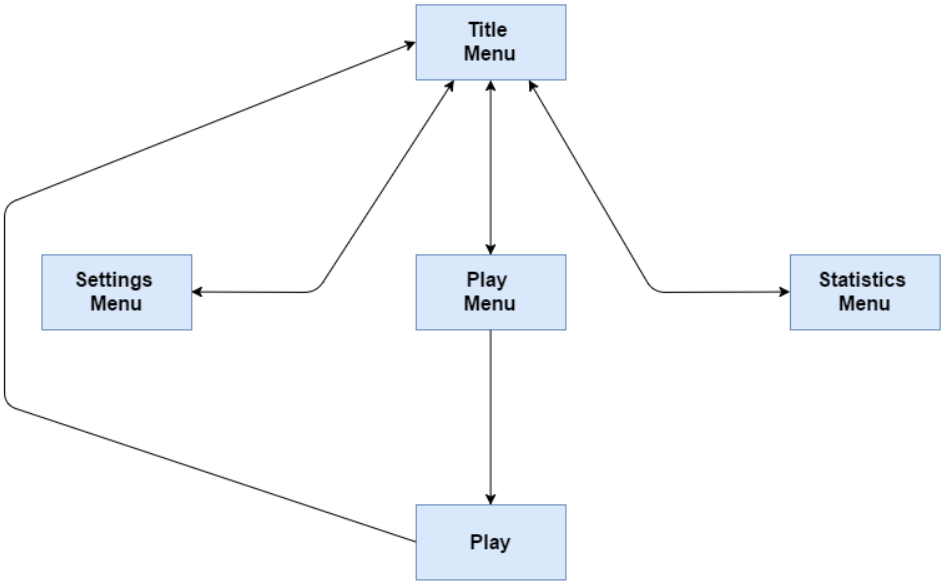
Missing introductory material. What is the problem domain? Why is it important? Why is difficult?

During the Fall Semester of 2016, we created a Pong derivative that features 3D graphics and gestural control of the player's paddle. We intended for this game to be part of a suite of 3D, gesture controlled games. However, during the last sprint of that semester, we realized that extending the game would be much more worthwhile than developing other games. We satisfied the requirements of classic Pong and we progressed as software engineers, but we did not achieve an impressive game. It is a satisfactory game. Rather than finish a collection of two or three satisfactory games, we want to finish a game that entices and intrigues the player. ~~We want to create a desirable product. Outside of monopolies, lucrative business seldom stems from mediocre products. This project will teach us how to create successful software, not just working software, if we reach for attractive and exciting qualities.~~

Our efforts from last semester produced a stable, satisfactory Pong experience. 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 accelerates suddenly after striking a paddle. The game proceeds suitably between each goal and ends as it should. The statistics are accurately recorded and are presented conveniently to the player. Generally, the UI is easily navigable and the gameplay is simple, but sufficiently challenging.

However, we have a plethora of improvements planned. We plan to implement a settings menu from which the player can adjust qualities that affect gameplay. The gameplay is the

primary focus of the change to come. Below is an interaction flow diagram which illustrates a simple overview of the user's experience.



Considering the large amount 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 title 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; randomly appearing (and disappearing) obstacles; and elements 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, and multiply the number of balls in the arena. Two player capability, background music, and sound effects may be implemented as well, but ~~yield to the development~~is a lower priority than

~~of~~ the aforementioned features ~~and the like~~. 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 • ~~Absence of lag~~ • ~~Program does not crash~~ • ~~Game ends after a player scores x number of goals~~ • ~~Free of noticeable bugs~~ • 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 • ~~Free of all bugs~~ • Artificially intelligent opponent that challenges, but does not overwhelm, the player • Two player gameplay (neither player is artificially intelligent) • ~~Background music~~ • ~~"Push to start" button~~ • Profile deletion	• Sound effects • Choice of multiple camera angles • Official support on systems other than PC coupled with Kinect • Background music •	• 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) • ~~Various camera angles~~ • ~~More or less than two paddles~~ • ~~Official support on systems other than PC coupled with Kinect~~ • Data encryption • ~~Official~~Standardized method of importing/exporting settings or statistics • ~~SQLite~~ • Persistent data technology other than the BinaryFormatter class in C# • ~~Games other than Pong~~ • Data visualization (other than descriptive text)

Our solution to this project's problems and its composing parts were influenced by our chosen game engine – Unity. (What were some of your other choices?) ~~W~~We chose Unity as opposed to other game engines for several reasons. The first reason is its ~~because of its unmatched~~ multiplatform support. Unity can deploy a project to nearly every modern, common platform via a single click (Unity3d.com). We want to be proficient with Unity because Unity is the game engine that enables developers to distribute their software as broadly as possible, thereby maximizing their profits and potential feedback. Second, ~~W~~we also chose Unity for Windows because of our choice of motion capture technology, the Kinect, is not supported on Max OS X which is the only option other than Windows for the Unity installation. We chose the Kinect as opposed to other motion capture technologies (such as …) because the Kinect is one of the most well established, consumer, motion capture technologies (James Ashley). Although not officially supported by Unity, it is (sometimes sufficiently compatible ??), thanks to Zigfu's ZDK. Furthermore, there exists an active and helpful online community which discusses problems and answers questions relating to ZDK and Unity. ~~Additionally, our instructor, Dr. Barlowe, has experience with the Kinect and is available many times throughout the week to answer questions that we may have.~~ Other technologies that we could have chosen in lieu of the Kinect include motion capturing suits, some of which also provide haptic feedback. However, none of those technologies matched the affordability of the Kinect and none of them seemed better at tracking the user's movement. Haptic feedback, although interesting, would not be appropriate for Pong because Pong would, at most, arouse a level of haptic feedback equivalent to playing a game of table tennis. There exists a connection between the Kinect and C#, our choice of scripting language, that also compelled us to choose the Kinect: both the Kinect and C# are Microsoft products and the Kinect SDK encourages the use of C#. We chose C# as opposed

to UnityScript, the only other well-supported scripting language for Unity, because we have not been exposed to C# in previous courses at WCU (we *have* been taught JavaScript which is essentially UnityScript). Not only did we choose C# to broaden our education, we chose C# because it is commonly used throughout the software development industry whereas UnityScript is confined to projects pertaining to ~~u~~Unity. C# also has a great IDE – Visual Studio. The sleekness and convenience that Visual Studio offers affirmed our convictions to use C#. Unity uses Visual Studio as the default editor whenever a C# script is opened. It~~'s~~ is also the IDE that is used in Unity debugging tutorials. ~~The compatibility with Unity is great, but it is not the only thing which convinced us to use Visual Studio.~~ Unit testing and integration testing in Visual Studio is extraordinarily simple ~~and ,~~ well-supported, and ~~. That convenience is valuable – because of it we do not have~~ allowed us freedom ~~to tangle with~~ from unofficial extensions for the purpose of testing.

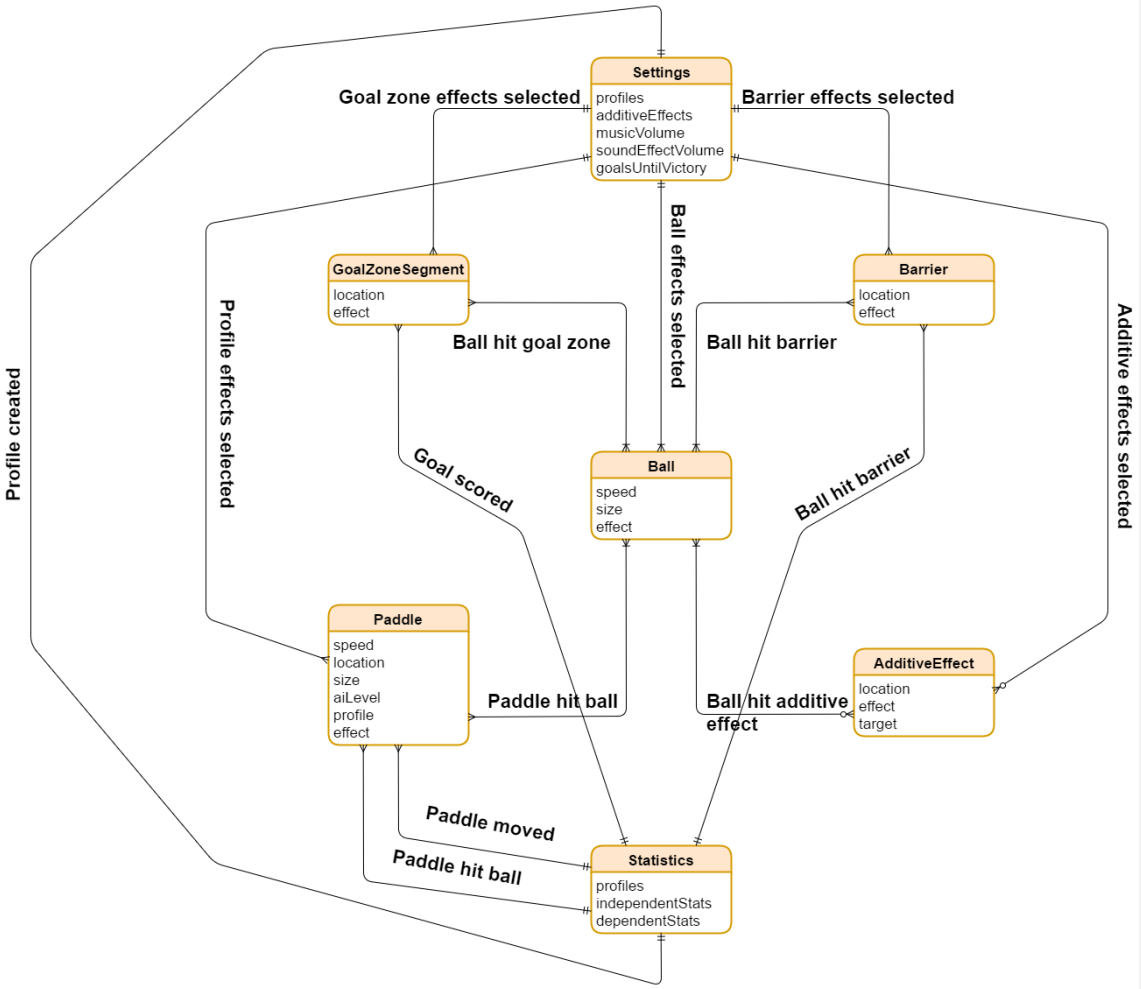
Why do you need data persistence? The problem of data persistence and Unity surprisingly has no officially supported solutions regarding a DBMS (database management system). Regardless, we searched the Unity forums in search of a community familiar with the pairing of SQL and Unity – particularly SQLite because our game is not a distributed application. We did find not official documentation pairing SQLite and Unity ~~together~~. However, we did find a non-DBMS solution in an official Unity tutorial (citation) regarding data persistence. 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 human-readable (and thus human-modifiable – read access does not imply write access) text~~.~~ (Grant Allen and Mike Owens – this is not listed in your works cited). To secure our project's source code, we used Git. Git is superior to its competitors for several

reasons. Git is much faster than Subversion because Git does not apply differences between files in order to produce the queried file ~~(Subversion does)~~. Instead, Git saves a new, separate file and produces it the instant that the file is queried. CVS, another competitor, does not feature support for atomic operations and therefore the integrity of the stored data is not guaranteed. Git *does* support atomic operations. 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.

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 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. CS 370 informed us of atomic instructions, issues related to concurrency (race conditions), and the consequences of issuing system calls – 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. CS 150 and 151 gave us a strong foundation in Java, and thereby a strong foundation in C#. (Place the preceding sentence first in your list of courses)~~, and many other CS courses at WCU expanded our knowledge of Java.~~ Our calculus courses (as well as physics courses) taught us how to efficiently solve problems relating to physics – this is useful when writing code pertaining to the physics of the gameplay.

There is still much more about Unity that we have yet to understand or 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 it. And lastly, perhaps the most imposing challenge will be researching the Kinect in order to achieve the functionality that we want. The Kinect is well-supported, but we do not really have official documentation to follow so researching the Kinect is a struggle. We have yet to find a fully capable middleware for the Kinect and Unity, however, we recently discovered OpenNI which has been proven to work well and so OpenNI (why? – Discussion of OpenNI should be placed earlier when you talk about alternate technologies.) will be our next avenue of Kinect-to-Unity research (Ju-Ling, Shih, and Hsu Yu-Jen). 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.

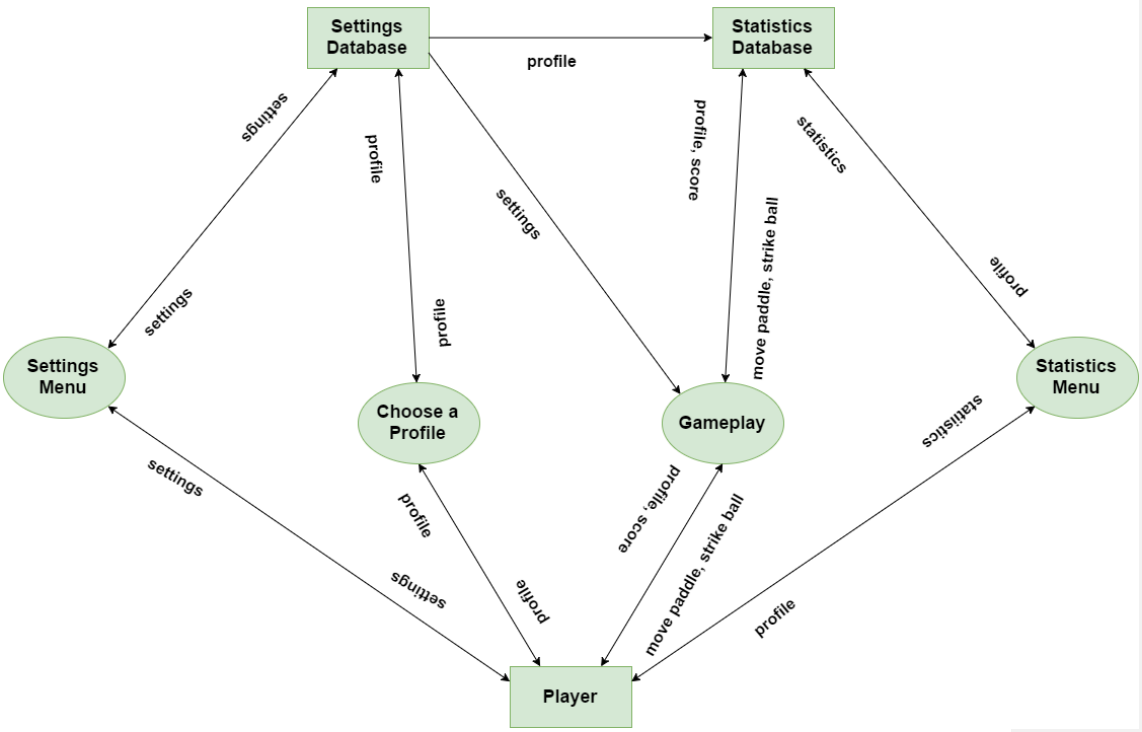
The basic structure of our design is illustrated below by an ER diagram.



This diagram is gameplay-centric. The term "additive effects" refers to the many elements which influence the gameplay. What does your diagram show?

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 regardless (regardless of what?). We intend to collect data which describes a player's skill, habits, strengths, and weaknesses.

Currently, we record displacement, hits, wins, losses, goal segment scored upon, 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 data flow diagram below to illustrate the flow of data throughout Pong.



The direction of the data is illustrated by the direction that the words are facing. TEXT SHOULD BE HORIZONTAL IN ALL CASES.

The data I've described is typically used for record keeping (in order to "break" records), matchmaking, prize winning, competitive play, self-improvement, or just for curiosity. Such data is most potent in the context of distributed applications. However, it is difficult to network data.

Our application 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. Securing data, not just to transmit it, but also to prevent access to it, requires encryption. ~~However, the offline, localized data related to our game is not of high importance and so binary is sufficient – data encryption is unnecessary.~~ We would like to learn more about data encryption, but it is not necessary for the scope of this project.

We will definitely test the system via playthroughs (user acceptance testing) and we will apply TDD (state what this abbreviation stands for), unit testing, and continuous integration testing, but we are unsure of how rigorously we can manage that alongside the many other goals which we have set for ourselves.

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 • "Push to start" button • Game over after 5 goals • AI (for what aspects?) • GUI interaction via Kinect	• Statistics menu • Settings menu • Two players possible • Player two controlled via Kinect	• Additive effects (such as… Be specific)	• Additive effects (such as … Be specific)	• Background music • Sound effects • Code refactorization • Code freeze	• Final report • Presentation

Works Cited

Grant Allen and Mike Owens. 2010. *The Definitive Guide to Sqlite* (2nd ed.). Apress, Berkely, CA, USA.

James Ashley. 2015. “Beginning Kinect Programming: With the Kinect for Windows V2 SDK.” Apress, Berkely, CA, USA.

Ju-Ling, Shih, and Hsu Yu-Jen. "Advancing Adventure Education Using Digital Motion-Sensing Games." Journal Of Educational Technology & Society 19.4 (2016): 178-189. Academic Search Complete. Web. 21 Jan. 2017.

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

Grading:

The first paragraph(s) should discuss the background of your project – especially why your project is important and why your project are worthwhile.

Formatted: Space After: 0 pt, Line spacing: single

You explained some of your design choices, but did not mention some of the alternative solutions. Are there other game engines? Are there other camera systems? You mention OpenNI but that should have been included in context of why you chose the Kinect.

Why do you need data persistence?

You listed some of the additions you plan, but didn’t discuss them in the paper. What specific additions are you including? Why did you choose them?

The last two diagrams need to be organized better (text rotated to be horizontal, lines straightened, minimize line crossings.)

At least one of your references is not listed in the works cited.

Three of your references should be from either a journal or a conference. It looks as though only one is. You can find additional references for either the motivation (Are immersive

Formatted: Space After: 0 pt, Line spacing: single

environments more enjoyable?) or the platforms (What other input/output possibilities are there? If this has been tried before, what were the results?)

Grade: 76