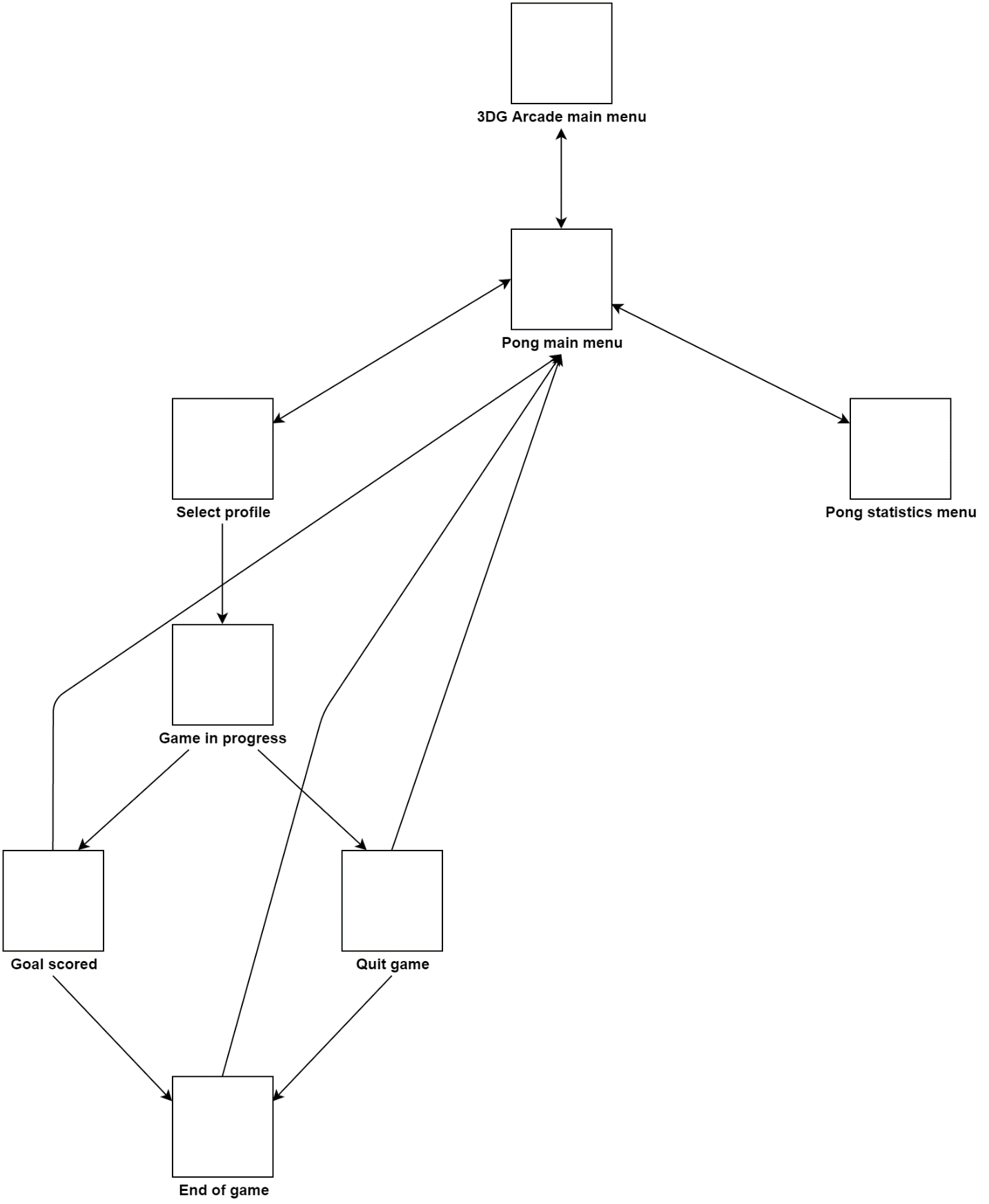
**3DG Arcade Fall of 2016 Report**

by Evan Arroyo and Joshua Sims

“3DG Arcade” is a suite of 3D, gesture controlled arcade games. We chose this project because we wanted to create games which are entertaining and intriguingly original. Great games are played many times over. Many fondly reminisce about games which they played often – the games which they return to years after the first encounter. These games are “the classics”. Building upon the concept of a classic game is a better foundation for a great game than the unproven blueprints of a fresh, conglomerate of ideas. Since the inception of this project, we have developed a 3D, gesture controlled adaptation of the classic arcade game called Pong. Our adaptation is entertaining, the gameplay and mechanics are replicative of the exceptionally successful classic Pong, and the 3D aesthetics and gestural control distinguishes it from its older, traditional counterpart – it provokes discovery as well as induces nostalgia. The same will be true of other games featured in 3DG Arcade.

Pong, like many arcade games designed 30+ years ago, is not a complex game, but it does have standards. We have successfully developed the qualities expected of any Pong clone as well as qualities beyond the scope of original Pong.

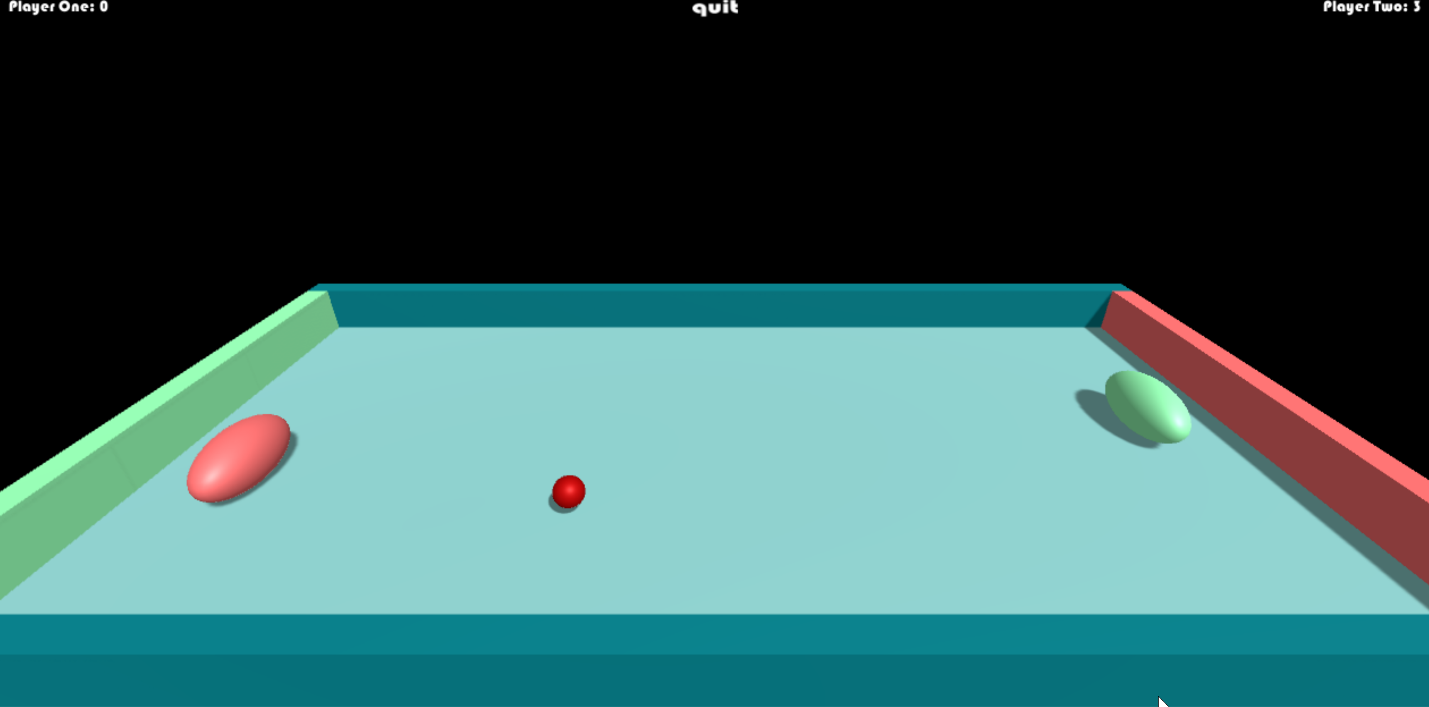
(As a visual aid for the following description of user requirements and their corresponding solutions, we have provided the following diagram which illustrates the flow of user interaction.)



After running the 3dg-arcade.exe file, the user encounters the “3DG Arcade main menu” and from there can proceed to the “Pong main menu” by clicking the “Pong” button. At that screen, the user can go to the “Select profile” screen by clicking “play” or the “Pong statistics menu” by clicking “statistics”. Alternatively, the user could return to the “3DG Arcade main menu” by clicking the “3DG Arcade” button Currently, the “settings” button is disabled because a settings menu for Pong has not yet been created. From the “Select profile” menu, the user can select a profile which will individualize the user’s gameplay statistics. After selecting a profile, the user can click the “start match” button to begin a game of Pong. Then the user will remain in the game until the winning number of goals is scored or until the player clicks the “quit” button. Upon those conditions, the user will be redirected to the “Pong main menu”.

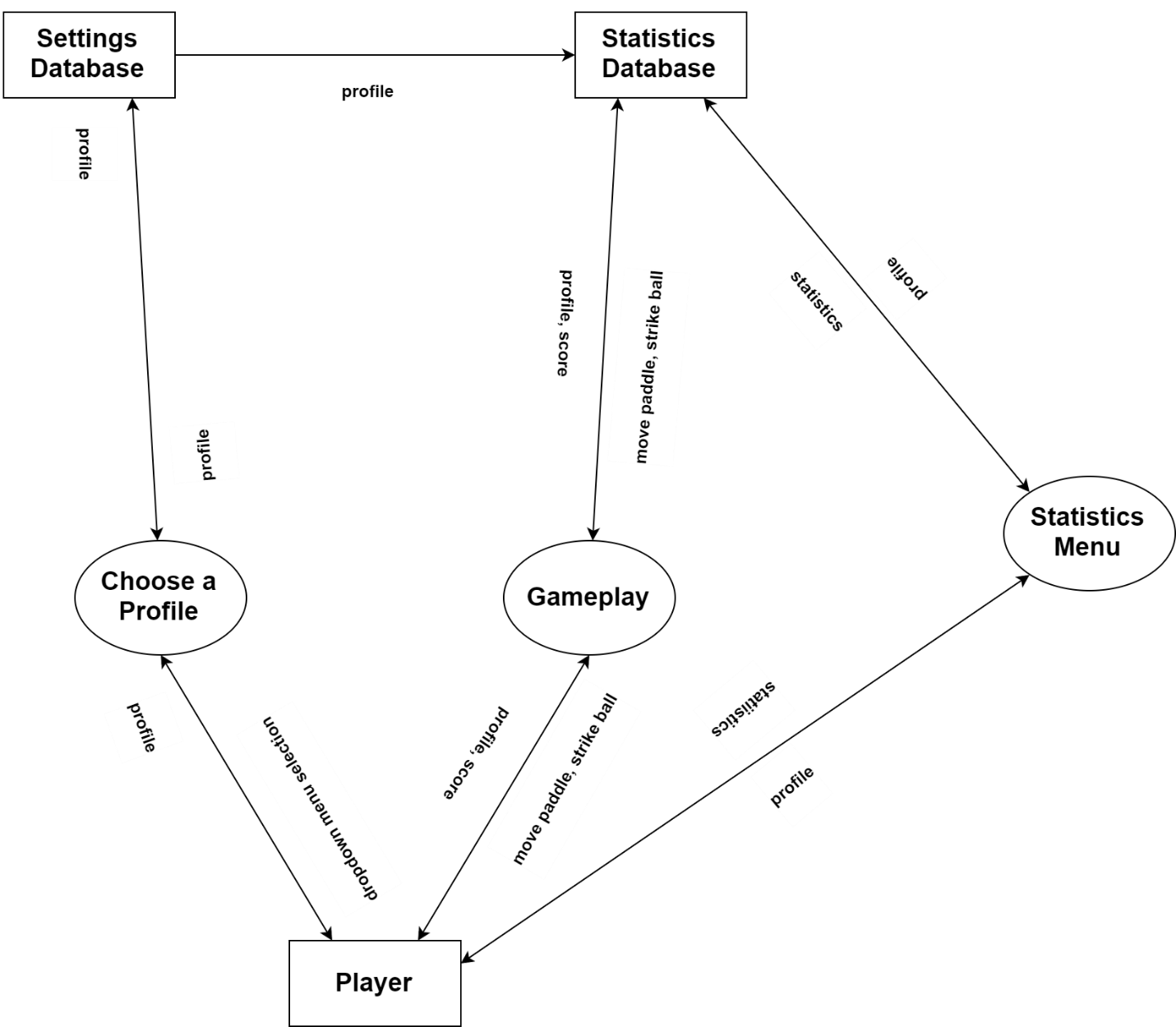
In original Pong, smooth playthroughs are expected every time; the game should never lag. By choosing to save statistics (a process intensive enough to interrupt play) after each goal, precisely after the ball is reset, but before the ball has begun to move again, we have ensured that the game does not freeze momentarily when the user scores. We could have programmed the save to occur after each match. That would have certainly guaranteed that the user does not experience any slowdowns during play. But, then, if the program were to be shut down voluntarily or involuntarily before the end of each match, the statistics gathered during the match would not persist. The simplicity of the navigation menus also contributes to a smooth experience. Rather than high definition images or animated graphics, we load only what is needed for the user to progress through the suite.



However, the bare minimum buttons and text are still aesthetically pleasant; the arcades which housed the 100+ lb Pong machines were vibrant and Pong, though graphically simple, should be pleasant to look at. The paddles in our adaptation are ovoids rather than the traditional rectangular prisms. Though we think the non-traditional shape is attractive, the decision to use ovoids was not founded on aesthetics – the purpose of the ovular form is to guarantee proper Pong mechanics. The ball must bounce off the paddles at various angles as though the face of each paddle was rounded. Either ovoids or some invisible ovular shape would have achieved the necessary angular deflection of the ball from the paddles but we chose the ovular shape because it was more direct and quicker to accomplish.

Classic Pong features rectangular paddles – they must have been wrapped with code which simulates a rounded body. We too used rectangular paddles during the first weeks of development, struggling to program that rounded body, until we stepped out of that mindset and realized that Unity features ovular models. Proper game mechanics is the most important aspect of any game – it defines user interaction, the quality which separates games from other forms of entertainment. We struggled with keeping the ball and paddles within the walls of the arena, preventing the ball from sliding along those walls, and programming competitively satisfying artificial intelligence. Until we reconstructed the entire arena, the ball randomly shot through the barriers. The problem must have been some slight fault in the 3D coordinates of the arena’s composing objects, causing an odd rebound. Sometime after that, we had trouble keeping the *paddles* confined to the arena. We tried changed the properties of the paddle’s physical components and some changes did prevent the paddles from passing through the barriers. However, as a side effect of those changes, the ball would slow down inappropriately when hit or would pass through the paddle. Eventually, we decided to just program the paddle not to pass through the two coordinates which define the walls of the arena. The problem in which the ball slid along the walls of the arena was resolved after we raised the ball upward such that its center was level with the center of the paddles. This solution was strange because the problem must have arisen from the downward force on the ball imposed by the striking paddles, regardless of the fact that the ball’s movement was locked in the upward and downward directions. Programming the artificial intelligence necessary for a single player match demanded a balanced solution: the average user wants to be challenged, but not dominated by the AI. The movement of the AI-controlled paddle has to be slow enough that the paddle cannot always reach the ball even though it always knows where the ball is going. There are many was of implementing such behavior, but the least convoluted was the route we took. We programmed the AI paddle to update its coordinates such that they would mirror the coordinates of the side-to-side coordinates of the ball. However, the AI paddle is also programmed to wait a set amount of time before performing that update. The amount of time can be increased or decreased, but only via a modification to the source code. Other than the level of AI reactivity, the programmer can readily change several game characteristics such as the speed of the paddles, the speed of the ball, and the camera angle. However, these settings are also not mutable via a user interface. We chose to favor the implementation of mechanics over the implementation of settings because varying settings were not a primary feature of original Pong.

But why then did we focus on the collection and accessibility of statistics – statistics were not a primary feature of original Pong. We chose to implement statistics because it adds a genuinely *different* dimension of entertainment to the user’s experience. Settings are an *extension* of existing dimensions; settings increase the playability of a game, but not to the extent that statistics do. During gameplay, statistics are gathered which quantify and track the tendencies and skill of the players. Those statistics are persistent: they can be viewed after quitting the Pong match or closing the application. To achieve that persistence, we used a C# class called BinaryFormatter which serializes objects, translating them into binary written to a hidden file. The deserialization of that file – the loading of the data – is also performed by BinaryFormatter. Using BinaryFormatter ensures better security. Other popular, Unity-compatible, serialization techniques, the use of JSON or Unity’s PlayerPrefs class, write plain text to files. Plain text is not ideal; it is human readable and can thus be easily modified, invalidating the data that it represents (Geig, “Persistence - Saving and Loading Data”). To ensure that only one copy of the statistics database, a class housing an array of values, exists in our application, we implemented the singleton design pattern. All components of the Pong game can submit statistics to the database as soon as they are generated. The independent statistics are saved after each goal as opposed to after each match; goals are a substantial milestone and are well suited as events upon which to save data. Furthermore, goals occur more often than wins, strengthening the integrity of the database by ensuring that it suffers minimal loss. Dependent statistics are calculated and recorded when the statistics are pulled to the display by the user. Calculating and recording dependent statistics at a time other than that of access is a waste of processing – like heating a meal hours before it is eaten.

(To illustrate the flow of the Pong statistics, we have included the following diagram.)

As of this report, a total of 44 statistics are collected and accessible. Some of these statistics include hits per win, displacement per win and winning goals scored on the right per winning goal and losing goals scored on the right per losing goal. As the above diagram illustrates, the user instigates the collection of statistics via the gameplay. Before playing, the user chooses a profile which governs where (to whom) the statistics are allocated. That profile information is sent to the settings database which then stores and forwards that information to the statistics database. The user can view statistics relevant to any existent profile via the statistics menu.

We have simplified the problem of statistics collection by omitting some features commonly associated with it. For example, in many other games, statistics are used to award the player whenever a certain statistic reaches a certain value or make automated judgements about a player’s. We only collect the statistics and show them to the player on demand. Distributed applications, especially, face a much more complex problem with statistics. For example, (disregarding the difficult of sending something reliably over a network) it would not be difficult to screenshot statistics and share them across the network as an image file or as a text file, but unless some kind of certificate was incorporated into that data, it plausible validity is damaged. Furthermore, if the statistics were transmitted while in-game, the programmer has to prepare for floods of data. We chose to develop this project as a non-distributed application because we did not want to deal with networking problems before we had a working application. Perhaps we would eventually make this a distributed application – after development of the non-distributed version has settled into a solid state. It would certainly make the statistics component of our project much more attractive if those statistics could be shared or used for matchmaking. Statistics are interesting, but the statistics component is not the only new dimension which distinguishes our Pong adaptation from the original. Perhaps more important to our game than statistics is the inclusion of 3D graphics and gestural control. The 3D graphics and the gestural control do not influence the game mechanics, the most critical part of the game, instead, they serve to intrigue the user who is likely already familiar with 2D Pong controlled by a joystick or keyboard. The statistics are worthwhile, but the 3D graphics and gestural control are more responsible for attracting attention.

To meet the aforementioned user requirements, we utilized several tools. We chose Unity as our game engine. Unity is one of the most widely used, highly rated game engines available. Despite its popularity, we were totally unfamiliar with it. Fortunately, researching an aspect of Unity is simple – we find the information we want from <https://docs.unity3d.com/Manual/>, <https://docs.unity3d.com/ScriptReference/>, and <https://unity3d.com/learn>. From those rich sources, we can discover how a unity feature relates to other features and how to write the code necessary to use a feature. Other than being conveniently extensible and understandable, Unity is built for high speed productivity. Manipulating objects in Unity is usually much quicker than manipulating them in C#, the scripting language in which we wrote the behavior that orchestrates our project’s components. The entire Unity UI functions as a tool-laden workshop surrogate for source code. Despite Unity’s advantages of convenience and speed, we used C# frequently and it was an excellent choice of a language. Researching C# is simple (thanks to Microsoft’s abundant and easily navigable documentation), the API is robust and sensibly designed, and Unity features total support of the C# language – it extends the C# API. We work with C# and Unity on the PC. 3DG Arcade is intended to be run on the PC and it supports the use of the Xbox One Kinect. The Kinect is responsible for receiving and interpreting the gestural input which controls navigation across UI menus and the movement of the player’s paddle in Pong. The Kinect was not easy to integrate with our program. There are little to no helpful tips or hints on how to successfully do it. After two sprints (although not dedicated to) researching the Kinect, we were able to resolve this issue and download all the necessary parts to integrate the Kinect. We had to install the SDK for the Kinect and several extra files from Microsoft. After the installation we had to import the “BodyFrame”, “KinectView”, and “GreenScreen” assets to Unity. The “DetectJoint” script which we wrote houses the code which tracks the user’s gestures and relays that information to the paddle object, allowing the user to move the paddle with their hand. The “JointType” enumeration allows us to differentiate between body parts such as the head, shoulders, wrists, elbows, and, particularly, the hands. The struggle to learn the Kinect API persists, but it has lessened. The connection between the Kinect and C# as Microsoft products helps to alleviate that struggle. Furthermore, we now have a better idea of how much time must be allocated to Kinect integration. The time which we are able to commit to this project is not enough to accommodate everything which we want to accomplish. The busy nature of college compels us to scatter our time among various courses and other obligations that we have rather than focus it on completing a single project at a careful pace. We did not utilize test driven development (TDD), unit tests, or integration tests although we sincerely wanted to. Not only are those tools helpful for productivity – they are great for maintainability. However, we did not implement and use them because we wanted to focus on developing features. So, the little testing that we did conduct consisted of playthroughs, click testing, and placing certain statements strategically in the source code to see if something predicted would happen. It was not a robust approach, but thankfully it did the job – the absence of unit tests and integration tests were not related to any bugs. To maximize the time committed to development, we used Git for version control and GitHub for project management. Git encourages us to be inventive with our code, to try things which might break other parts of the program – the version control and easy navigability helps us to return to a stable state if we do break our working code. GitHub helps us manage issues – units of information which reflect discrete, project-related tasks. As a software engineer working for a nearby company, Joshua, a member of our team, uses Atlassian products including BitBucket, JIRA, and Confluence for version control and project management. According to his experience, the Atlassian products as a bundled whole are more effective than GitHub in regard to large teams. BitBucket (powered by Git) functions much like GitHub, offering cloud repositories and code review, JIRA, when coupled with BitBucket, allows users to track codebase updates in relation to project issues, and Confluence offers a documentation repository integrated with multimedia editing tools. Entities (documents, issues, codebase updates) within each of the aforementioned products can link to one another, regardless of the product. Confluence is the major divisor between GitHub and the Atlassian products. GitHub does not feature document management. Despite those comparisons, we are satisfied with GitHub and do not feel inadequately equipped without any of the features included in the Atlassian products – this is not surprising since the Atlassian products benefit large teams far more than teams of two. We will continue to use GitHub to help us coordinate our actions and decide on the most efficient routes to take through this project.

Some of the college courses that we have taken have helped us make informed decisions and solve problems related to this project. CS 253 has made us more aware of useful design patterns and design principles. CS 370 taught us about race conditions and how to deal with them or avoid them. We are more knowledgeable about memory and hardware relationships after taking CS 350. CS 150 and 151 gave us a foundation in Java and thus (due to its similarities) C#. Many other CS classes have bolstered our knowledge of Java via continued use of the language. Generally, the CS courses which we have taken thus far have prepared us with skills, if not knowledge, for the tasks accompanied by this project

The future of this project could entail the development of a new game, but it seems to be (according to conversations held within our team) angled toward the continued development of Pong.

(A MoSCoW analysis is included below, illustrating the objectives relevant to this project.)

|  |  |  |  |
| --- | --- | --- | --- |
| Must have | Should have | Could have | Would like but will not have |
| The game is playable (adequate responsiveness and expected mechanics); 3D graphics; the gestural control is implemented and usable; the user can create and select a profile; the statistics are collected, accessible, and accurate; and the AI poses a satisfactory challenge to the user. | Error handling; navigation menus; sound effects; *attractive* 3D graphics; intuitive gestures. | Music or some ambient noise; Highly detailed 3D graphics 3D movement of the paddles and ball; power-ups; obstacles for the ball; random events during gameplay; select a paddle color; select a paddle speed; select level of artificial intelligence; select a camera angle; split-screen. | Networking (local or online multiplayer). |

Aside from developing an interactive settings menu, including sound effects, and refining gestural control, we would like to enhance the game mechanics. We could implement 3D movement and other features which such as power-ups (rewards gained by hitting them with the ball), random events, and obstacles (for the ball). We feel completely capable of implementing the aforementioned features, especially since we are now more aware of our weaknesses and our strengths. Our team learns quickly and can therefore develop features rather quickly. We may even be able to invest some time into unit tests if we can appropriate enough time for them. Furthermore, if we can adopt a habit of completing work earlier in a sprint rather than later, we should be more likely to develop unit tests. Neglecting this project until the last three or four days of the sprint has, at times, hindered our progression. Aside from some mismanagement of time within each sprint, we have followed the timeline proposed at the beginning of this semester fairly well and we predict that we will complete next semester’s tasks with greater efficiency.

Works Cited

Geig, Mike. "Persistence - Saving and Loading Data." *Unity*. Unity Technologies, 2016. Web. 30 Nov. 2016. <https://unity3d.com/learn/tutorials/topics/scripting/persistence-saving-and-loading-data>.