**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 doubtlessly 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.

We built our 3D game suite in Unity, one of the most widely used, highly rated game engines available. 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. When playing Pong, statistics are gathered which detail the habits 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. The We used ovoids rather than rectangular prisms for the Pong paddles. 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. Also, we like the originality of the ovular paddles. Throughout our process we used GitHub for version control and project management. GitHub is absolutely amazing. GitHub encourages us to be inventive with our code, to try things which might break other parts of the program because it is alright – the version control and easy navigability of GitHub helps us to return to stable state if we do break our working code. Furthermore, GitHub was very helpful with managing issues – units of information focused on discrete tasks about our project. The superior issue management helped us to keep from skipping things, helped us to coordinate, helped us to decide where to go to next.

Now what about this new game business? Well, we could have created an entirely original game, but that is not an easy task. If we had done so, we surely would not have achieved the same level of progress. The very foundation of our project encircles the idea that great mechanics of old games can be brought back to life with innovation – original games are great, but renewing an older game is also grand, if not better – especially when given a short interval of time like we had.

So the implementation of the Kinect. The Kinect gesture control was not an easy thing to implement. There is little to no helpful tips or hints on how to successfully add it so that the game maker can implement a Kinect into their game. However, with some hard digging, we were able to resolve this issue and download all the necessary parts to have the Kinect work correctly. As the game maker, you have to install the SDK for the Kinect and also some extra files that, conveniently enough, are located on the Microsoft Downloads tab. With these tools in hand, we were able to start implementing the gesture control into our game. With these tools comes the Body Frame, KinectView, and GreenScreen assets which need to be imported into the Unity assets. There also has to be a file that detects the joints themselves. The DetectJoint script that we wrote has all of the code that detects the users body moments (or gestural movements) and relays that to our paddle objects allowing the user to move the paddles left and right if using the first person display or up and down using the third person display. It picks up on a personal skeleton of the user or the game maker for testing and allows them to see just how they look to the Kinect. The Kinect comes with a JointType enumeration which allows us to use all kinds of body parts for our movement such as our head, shoulders, wrists and elbows. For this, we used just our hands. After all of this is implemented using the code and testing, we are able to move the paddle in the game both ways according to how the camera is angled. Thus, giving us the ability to move the game paddles with just our hands in the game rather than a mouse and keyboard.nser

Following is the solution as viewed from a code perspective. <insert UML diagram here>

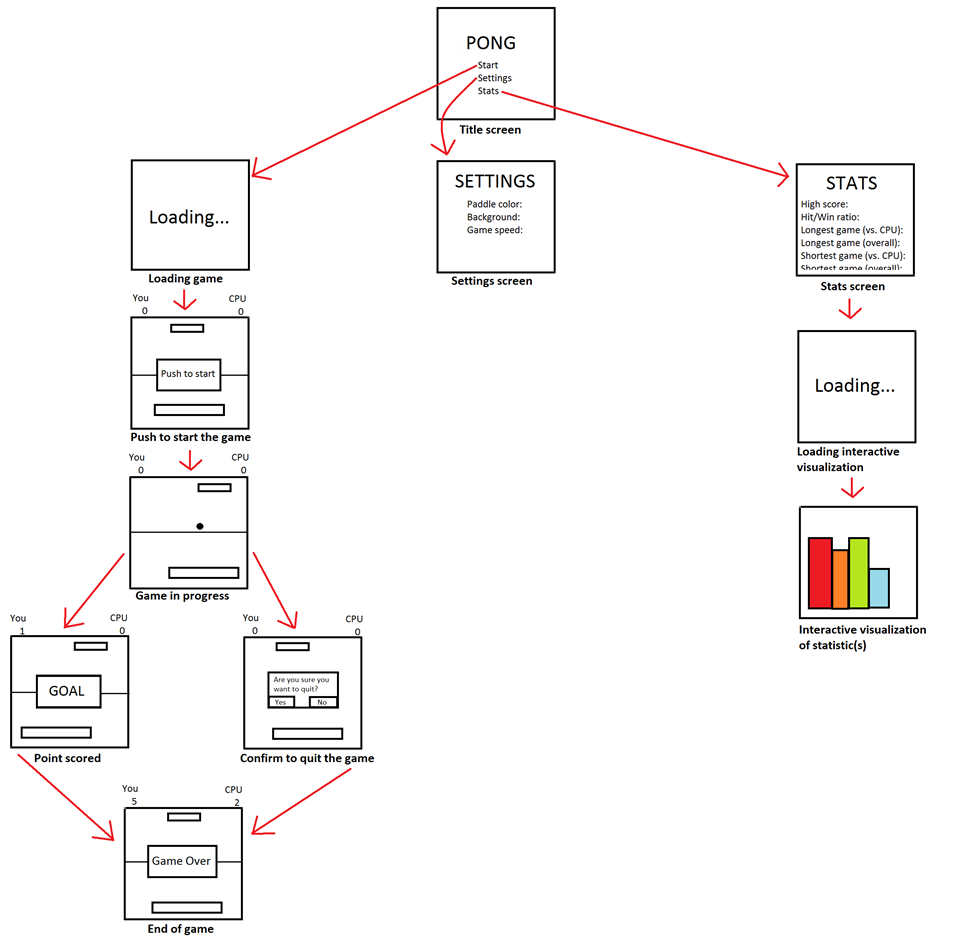
*Pong is not a complex game, but it does have standards. Smooth playthroughs are expected every time; the game should never lag. The gestural control should be intuitive and totally responsive. The 3D graphics, and the sound, if included, should be pleasant, though not necessarily highly detailed. The 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*

this is how we meet those user reqs…

So, someone wants to play classic arcade games, but with an interesting, new twist… Welcome to 3DG Arcade. To run 3DG Arcade, the user must first download the executable named “3dg-arcade.exe” from the bin folder in our GitHub repository @ <https://github.com/J-o-s-h-S-i-m-s/3D-Gesture-Controlled-Arcade-Games> then the user must run the executable. The 3DG Arcade main menu loads. The user can then navigate to particular game by clicking on an appropriately titled button. Then the appropriate game main menu loads. From there the user clicks to play the game, and can click to view statistics or click to go back to the 3DG Arcade main menu. A settings button is present on the Pong main menu, but we have yet to implement user customizable settings – it’s a short way away though, we hav e may of the building blocks already in place – just some final arrangements to be made. Anyway, the player can click to play the game and is brought to a screen where they select, create, or delete a profile for themselves. Currently player two will not be able to utilize the Kinect – two player gestural control is still in the works – it is quite another beast, but we are heartened to undertake it. Anyway, after selecting profiles, the players may continue to the game or go back to the Pong main menu. Continuing onto the game, the players will encounter a delay and then the ball will shoot forward at player one. It would be possible to shoot it randomly or at a random player, but we feel that the shooting at player one does not much detract from the positive experience. After the ball shoots, the players dash side to side with their paddles, striking the ball at whatever angle is strategic/possible. Once a goal is scored, the appropriate player’s points are incremented. It goes on like this until a player wins – for now the winning threshold is set to 5 goals. Once a player wins, they are redirected to the player selection screen – we could allow them to continue, we just didn’t include that feature because there were other, more significant features to be developed.

So, aside from playing the game, the players can navigate to the statistics menu where they can view the statistics relevant to certain profiles. From the statistics screen the players can then return to the Pong main menu.

The following is an interaction flow diagram:



(the interaction flow diagram will be a bit different from this).

So our project works quite well. <insert screenshots while talking about this>. The main menu is not given a great deal of work to do, but it does what it is supposed to do well and it looks quite attractive. The same goes for the other menus. The choose player menu and the statistics menu did require quite a bit of thinking. The choose profile menu uses dropdowns and input fields which lay atop eachother, but are only revealed at appropriate times. The chosen profiles influence the statistics that will be recorded. The statistics menu uses dropdown menu to filter the statistics database and then the filtered data is displayed on the screen. The calculation of dependent values is done at the time of statistics rendering – not when the goals are scored because it doesn’t HAVE to be done when the goasl are scored – not data is lost if the dependent values are only calculated and recorded at the time that the players request to see them.

The game works very well – via Unity, we can set different speeds for the paddles and the ball, however, until we implement the settings menu, the executable using user will be unable to modify such things and will not benefit from all that this game has to offer. Despite that, the game is still very entertaining. The collection of statistics gives extra incentive for players to play – it gives them more reason to do unusual things – to focus. There are so many variables that could be included to make the game more interesting – power-ups, obstacles, random encounters – but, in terms of the classic Pong, we have done quite a good job. However, there does exist one quirk in which the ball picks up tons of speed – it’s bettern than a slowdown though – in fact its kind of exciting. There still may exist other bugs too – especially since we have not done proper testing.

Thankfully our project works well even though we did not utilize TDD, unit tests, or integration tests as we so sincerely wanted to. Not only are those tools helpful for productivity – they are great for maintainability. But we did not use TDD, unit tests (or integration tests) because we wanted to focus on developing features. It is the definitely the better practice to use TDD, but we chose to exclude TDD, unit tests, and integration tests from our time budget. Joshua, one of the developers of this project, has been working with a software company for a little more than 6 months – from his experience, completing assignments at the non-school level is much easier since you are not expected to balance so many different assignments and are encouraged to do one thing and do it well.

So, the testing that we did was playthroughs, click testing, and placing certain statements in certain strategic place to see if something predicted would happen. Not robust, but thankfully it did the job. Thankfully, the absence of unit tests and integration tests was not related to any bugs.

So what problems did we encounter?

At the very beginning of the project, the ball would randomly jettison out of the arena. We weren’t sure why and we knew that were were total novices to Unity so we assumed that we had built the arena incorrectly or set the ball incorrectly. (Wen had JUST arranged everthing). After we rebuilt everything (which was actually quite an easy task), the ball stayed within the arena.

Well, the ball was getting caught along the barrier and we tried to figure it out for awhile by messing with the geometry of the paddles, the material properties of the paddles and of the ball – in the end it was so simple: the ball was a bit too low and was htting the paddles at an angle such that it was being pushed downward (although movement was restricted in the vertical directions, so that was unexpected).

Another slight problem is the incessant ricocheting of the ball between the barriers – however, it is not preferable to completely eliminate a great degree of ricocheting because it makes for an interesting and actually quite effective strategy.

For awhile, toward the beginning of the project, the paddles were rectangular prisms. This paddles were changed to ovoids because when the rectangular prisms struck the ball, the ball did not bounce off at erratically which is fun and adheres to the classic Pong mechanics.

We had trouble keeping the paddles confined to within the barriers so we changed their composing materials (the rigidbody) but then the ball would slow down – it jus twouldn’t hit correctly. Eventually we removed the rigidbody components from the paddles so that they would just deflect the ball like a wall, though then the paddles could pass through the barriers so we just wrote code around the paddle movement code to prevent the paddles from moving beyond the locations of the barriers.

Implementing the Kinect was generally very difficult.

The courses that helped us make informed decisions and fix problems were CS 253 for make us aware of useful design patterns and design principles. CS 370 for making us aware of race conditions (which we avoided entirely from programming near because there was no need for such kinds of things). CS 350 made us more aware of the advantages of using memory in certain ways and arranging code in certain ways (for the sake of optimizations). CS 150 and 151 were foundational for Java and thus C# -- though many other CS classes have had us use Java as well. Generally, the challenging programs that we have completed as part of CS courses really developed our abilities.

brief on statistics…

We collected data regarding the player’s gameplay.

People usually use that data to showcase their skill, set records, use as a guide for improvement, matchmaking, or awards.

non-network difficulty…

tracking the statistics to award the player whenever a certain statistic reaches a certain value, record records, advise the player based on statistics, generate an overall skill score.

network difficulty…

It would not be difficult to screenshot the 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, there would be a shortcoming in its plausible validity. If the statistics were shared in-game such that the game is a distributed application then the statistics would have a far greater credibility, but the networking aspect of distributed applications is difficult to manage because networking introduces another variable of difficulty.

detailed statistics…

we gathered statistics on…



The project followed the timeline generally well. What didn’t follow so well was not the order of the intervals or that they spilled into one another – it was within the intervals. Work tended to be done within the last 4 or 3 days of the sprint because so many other things were happening – as I’ve said school is a balance whereas working at the career level is a focus. Particularly, the last orchestration and gluing of the statistics and the navigation menus and the choosing of players was the most intense thing – another sprint would have been nice, but it was doable – just very intense with everything else going on.

So the timeline was pretty good, ut like I said we would have done things differently in regard to the within the sprints work. Furthermore, if we did manage our time better during the sprints (but it is very hard considering this balance that must be maintained) we may be able to incorporate unit tests and MAYBE TDD. But admittedly, it is unlikely that we will be implementing TDD – TDD takes time – it is very rewarding because it is an investment which ultimately saves time by preventing bug. But if you can take the risk and get out safely, not using TDD is quicker and I predict that we will be taking the risk considering the balance that we must maintain. However, we are much more adept with Unity, C#, the Kinect, GitHub, and the process of making a game now so I think because of that things will move quicker and maybe time will became loose enough to squeeze in enough unit tests and continuous integration.

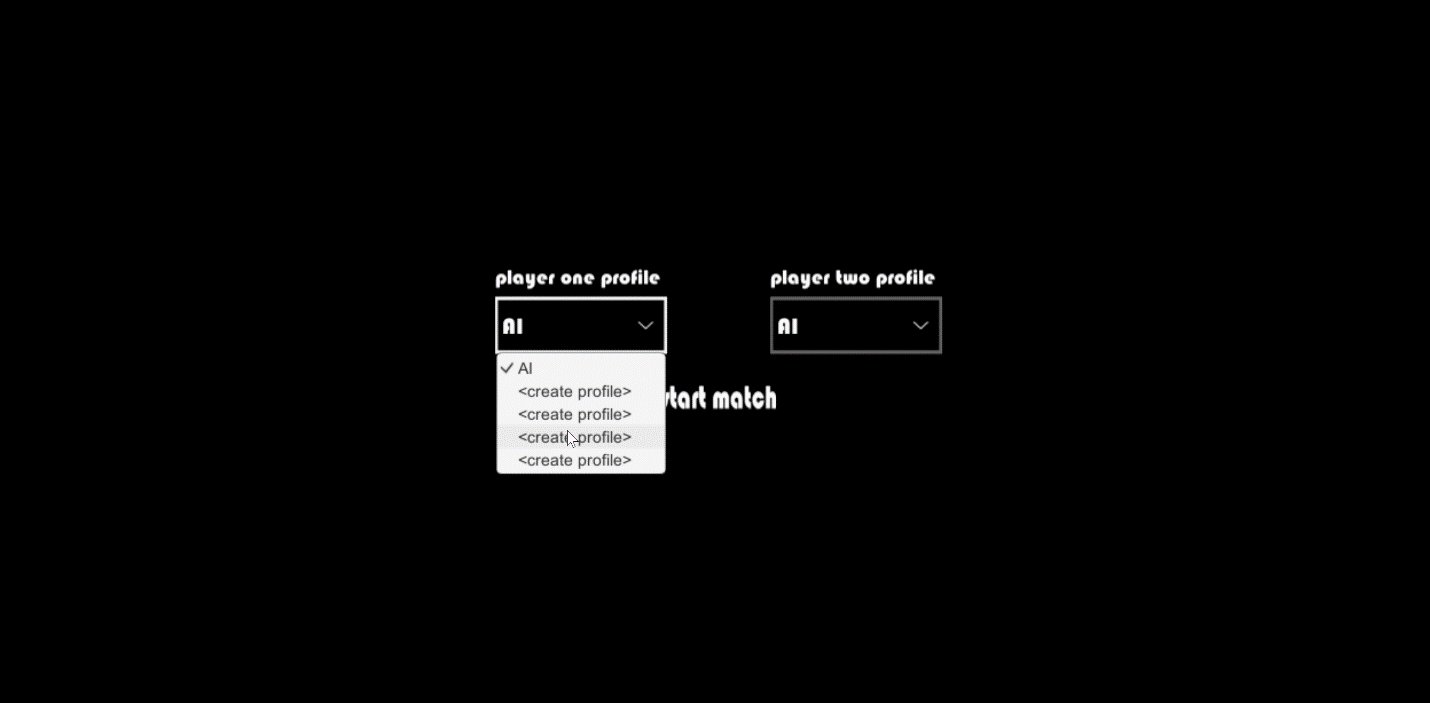
Our soonest upcoming objectives are to further develop Pong – we will be careful not to get too involved, however because we intend to create at least one more game before the end of the Spring semester. For Pong we will at least include a basic settings menu that allows the player to choose the ball speed, paddle speed, and number of goals to reach a win. We MAY include split screen if it does not seem too time-consiuming, we may implement a PUSH to START button, we may implement a ARE YOU SURE you want to quit button. However, we probably won’t retrospectively refactor the code or implement unit tests because we want to develop the next game more before doing those maintainability/optimization things for Pong.

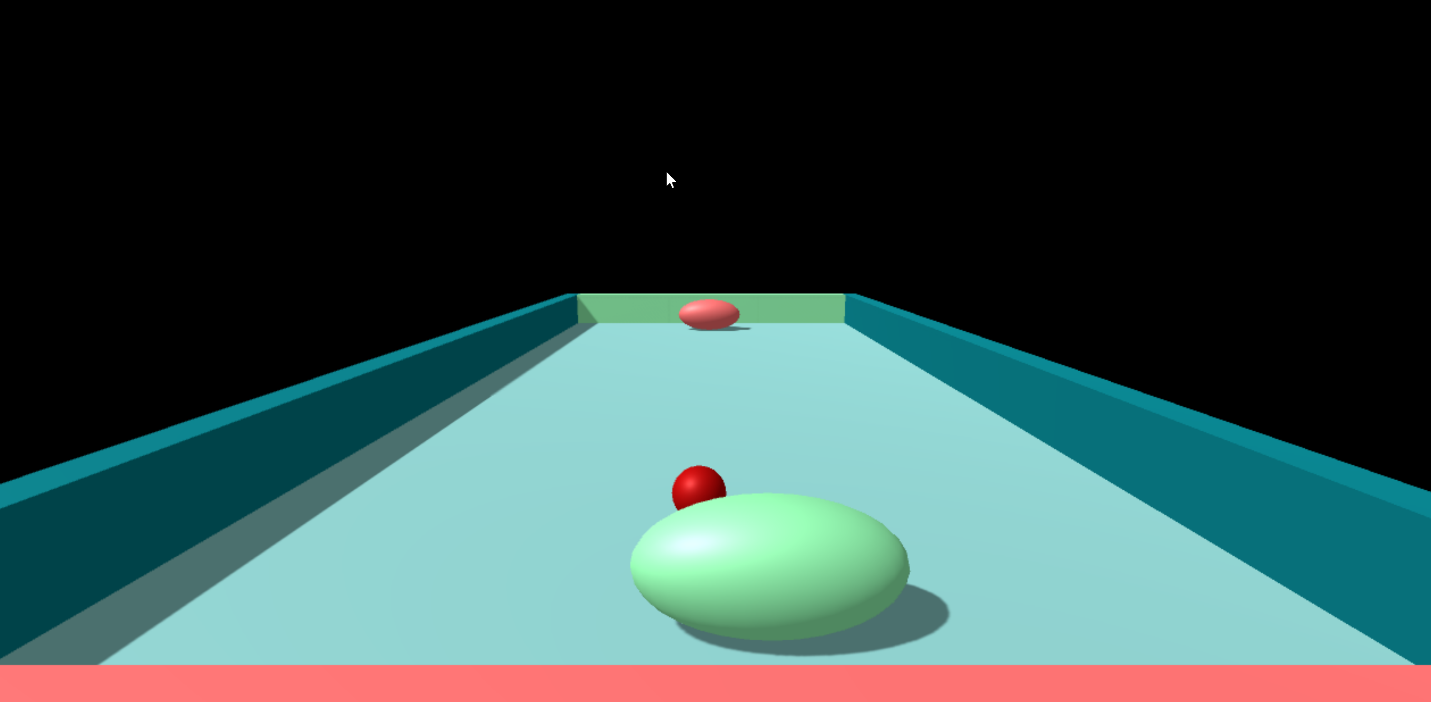
The games which we may develop next: galaga, dig dug?

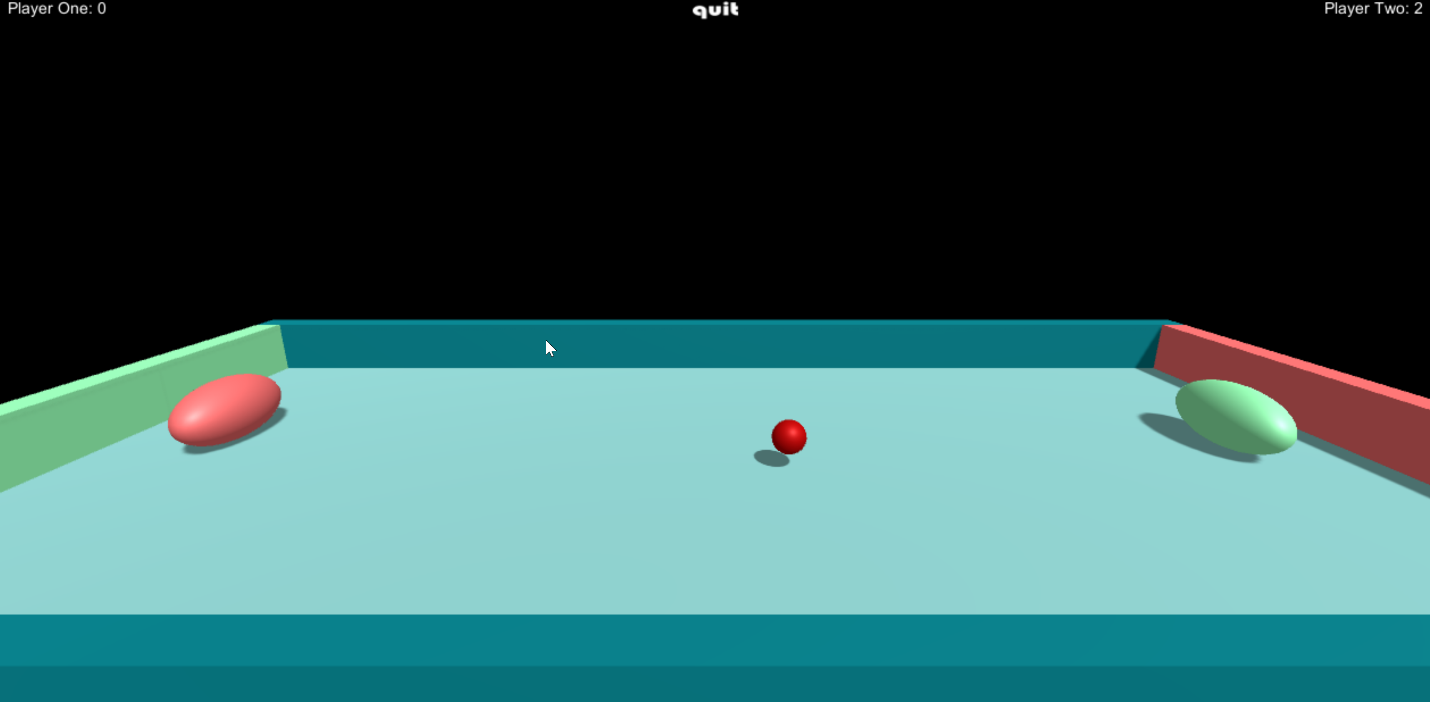
screenshots which will be scattered around at appropriate places



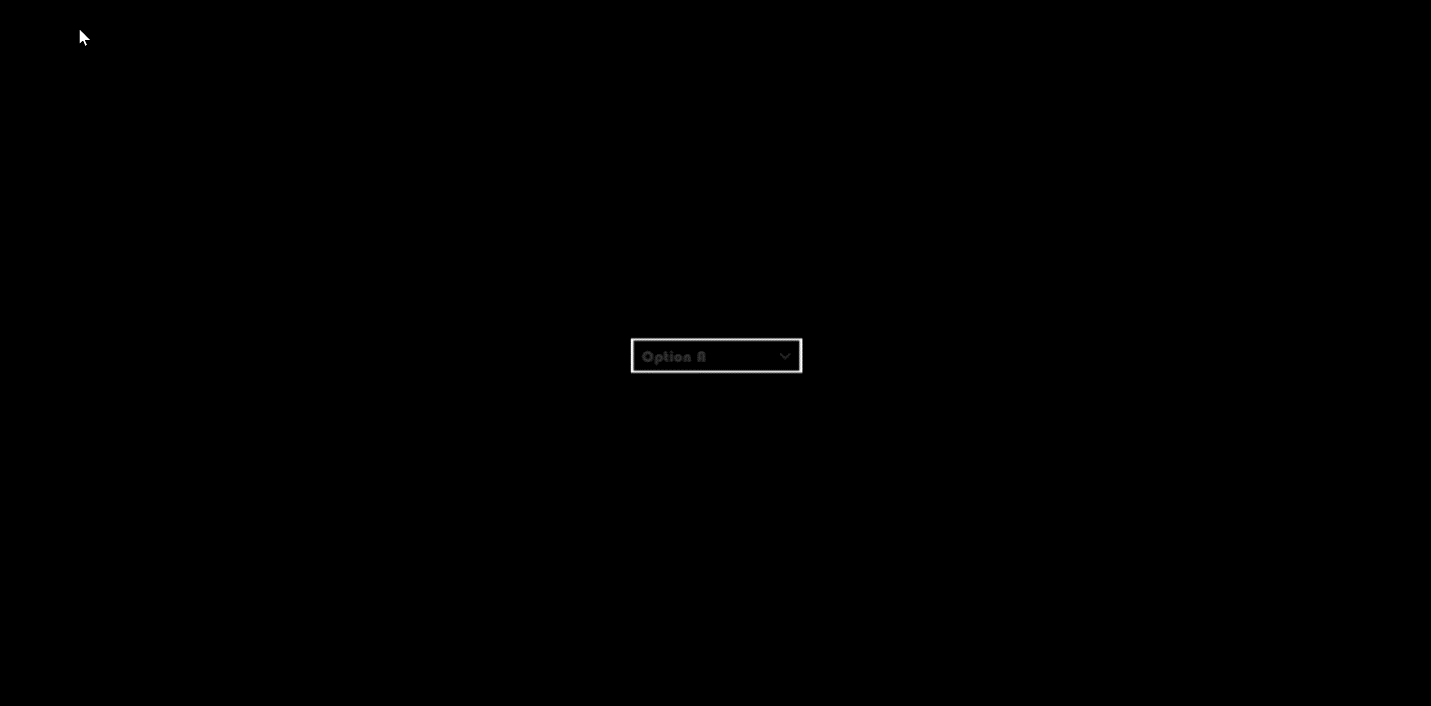








This one is the statistics menu (it’s not done yet).



and this MOSCoW analysis will be placed somewhere that is appropriate

|  |  |  |  |
| --- | --- | --- | --- |
| 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). |

(the above MOSCoW analysis is going to be a bit different from this).

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>.

---------------------------------------------------------------------------------------------------------------------

multimedia (scatter it around as needed)

data flow diagram

interaction flow diagram

UML diagram

screenshots of the current system

MOSCoW analysis

Our suite of games is built with Unity and C# and is designed to be played on a PC with the Kinect for Xbox One. We could have used Unreal Engine 4 or JavaScript or Boo. We could have

We used BinaryFormatter for serialization (persistence) instead of JSON…

We used ovoids rather than rectangular prisms…

We save on each goal

Throughout our process, we used GitHub for version control and project management.

We could have created entirely original games…

Kinect implementation…

use case…

Someone wants to play classic arcade games, but with an interesting, new twist…

To run 3DG Arcade, the user must first download the executable named “3dg-arcade.exe” from our GitHub repository @ <https://github.com/J-o-s-h-S-i-m-s/3D-Gesture-Controlled-Arcade-Games> then the user must run the executable. The 3DG Arcade main menu loads. The user can then navigate to particular game by clicking on an appropriately titled button. Then the appropriate game main menu loads. From there the user chooses players, clicks to play the game, and can click to view statistic or click to go back to the 3DG Arcade main menu. From the game, the player can choose to quit and return to the game main menu or just play.

The project works well

Our navigation menus work well

Pong works – just missing settings and..?

We did not use TDD, unit tests (or integration tests) because we wanted to focus on developing features. It is the definitely the better practice to use TDD, but we chose to exclude TDD, unit tests, and integration tests from our time budget.

We encountered problems…

Thankfully, the absence of unit tests and integration tests was not related to any bugs.

The ball getting caught along the barrier

The ball incessantly ricocheting

The ball not bouncing off the paddles at angles

Paddles composition - not contained within barrier, but also must hit the ball and let the ball bounce

Ball would shoot out of the barriers so rebuilt the arena

Implementing the Kinect

The courses that helped us make informed decisions and fix problems were…

brief on statistics…

We collected data regarding the player’s gameplay.

People usually use that data to showcase their skill, set records, use as a guide for improvement, matchmaking, or awards.

non-network difficulty…

tracking the statistics to award the player whenever a certain statistic reaches a certain value, record records, advise the player based on statistics, generate an overall skill score.

network difficulty…

It would not be difficult to screenshot the 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, there would be a shortcoming in its plausible validity. If the statistics were shared in-game such that the game is a distributed application then the statistics would have a far greater credibility, but the networking aspect of distributed applications is difficult to manage because networking introduces another variable of difficulty.

detailed statistics…

we gathered statistics on…

the source is the player’s interaction with the game.

list user reqs…

the project followed the timeline well…not well…

We would have done differently…

If this was a work atmosphere instead of a school atmosphere, we would have incorporated the unit tests.

We plan to…

Further develop Pong – refactor, implement unit tests, include split screen, include settings, push to start the match, are you sure you want to quit

develop another game – galaga, dig dug…