Evan Arroyo and Joshua Sims

21 April 2017

Final Report

The explanation of the purpose of this report appears here. The explanation of the purpose of this report is best written after the rest of the report is completed and therefore it will be written after the rest of the report is completed.

The aim of our project was to create a three-dimensional, gesture controlled adaptation of Pong. We selected this project because it involves subjects which interest us as well as many others in the computer science industry – potential employers included. Among these subjects are three-dimensional graphics, gestural control, data persistence, and, particularly, ubiquitous applicability, and ubiquitous accessibility. The most popular software applications are often both ubiquitously applicable and accessible. Google Search, Facebook, YouTube, Snapchat, Instagram, Ballz, and Spotify – these are examples of ubiquitously applicable and accessible software. Each is available on varying platforms, including mobile devices which are almost always within the user’s reach. Furthermore, these applications require little prerequisite knowledge. They are thus inherently accessible to most individuals and the bases for using them are common. They are predominantly used for entertainment, education, and/or socialization. The audience for these applications is thereby universal. Little knowledge and skill is required to play Pong and thus Pong is inherently accessible to most individuals. Furthermore, the basis for playing Pong is common amongst most individuals: it is the desire to be entertained. Our adaptation of Pong extends upon its classic counterpart with three-dimensional graphics, three-dimensional physics, additive gameplay effects, gestural control, and statistics. These extensions, however, have not hindered the ubiquitous applicability of the game nor its accessibility. We have gained not only proficiency in creating software which shares qualities common to tremendously popular software, we have also gained proficiency in the technologies which compose that software via the application of our chosen toolset.

Through our efforts, we have improved our ability to estimate the requirements of a project as well as the time to be dedicated to each requirement. The features that we pursued and the timeline which ordered them appear in the following table.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 |

We did not finish the development of every one of the features listed, but we did finish those which were of highest priority – specifically, the additive effects, three-dimensional graphics, three-dimensional physics, and gestural control. First, we implemented the gestural control. Thereafter, we built the arena, introduced the ball, gave it movement and tested the physics provided by Unity. Then we programmed the ball to reposition itself at the middle of the arena after reaching a goal zone. At this point, all the elements of basic Pong were complete, except for the opponent paddle. We then implemented the artificial intelligence. After programming the opponent paddle to follow the ball, we focused on developing the additive effects. The additive effects required the most attention – the multiple balls effect especially. To incorporate multiple balls, we had to refactor the artificial intelligence to consider the different scenarios which it would encounter when pursuing multiple balls. Once the additive effects were completed, we implemented the settings menu and thereby enabled the user to toggle the additive effects.

The software that we have built could easily be multi-platform given that we built it upon Unity which features the ability to deploy applications to nearly any platform. The caveat is that our technology for interpreting the user’s gestures, the Xbox Kinect, is not compatible with many platforms. It would be simple, though, to receive user input via some other means because our implementation loosely couples user interactions with the user input technology. The user interactions were programmed using the C# scripting language. We used C# to program everything included in our project, however, given more time, we would reprogram the collection and persistence of the statistics using SQL – we researched, but did not incorporate a SQL database solution for our project because the implementation of such was too demanding given that it was of a lower priority than the substantially more demanding implementations of the gameplay and graphical menus.

We completed the gameplay, however, it could be extended further. The artificial intelligence could also be extended. Currently, the artificial intelligence pursues the ball which is at the top of the ball queue. The balls are enqueued per their position in the arena. If a ball passes the middle of the arena, toward the AI paddle, it is enqueued. If a ball is hit by the AI paddle, regardless of its position in the queue, it is marked to be dequeued. If a ball is at the top of the queue and is marked to be dequeued, it is dequeued. We did not complete the statistics. The UI is not interactive via the Kinect. The game does not end after a set amount of goals.

There are no bugs in the code from what we can tell, but we may have discovered one had we tested the application more. We tested the application by interacting with it. We did not utilize unit tests as we originally intended to. However, the interaction testing was successful – it yielded good quality. Through interaction testing, we found bugs in the AI – the AI would not track balls if they bounced back, if the main ball went immediately into the AI’s goal, if another ball was hit before the ball being tracked. We fixed those by providing extra flags for the ball which determine the state of the ball as it should be tracked by the AI. We encountered bugs where the balls would slow down suddenly or speed up suddenly for some odd reason. We just set the ball to go at its normal speed if it did otherwise. We encountered bugs where the balls would technically hit two goal zones and score points for each goal zone although we thought that the OnCollisonEnter method on the ball was called only once for the first object it collided with, not twice. But apparently, it can collide with two objects “at the same time” so it was called twice for each goal zone that it collided with. For that bug, we just made it into a feature because it would have been much more difficult to eliminate that quirk rather than ride with it. So we introduced crosshair on the goal zones which demarcated the extra points areas that you could hit. If you hit the center of the cross, you’d technically hit every goal zone and get points for each. If you hit a line, but not the center, you’d be hitting two goal zones and get points for each. We encountered trivial runtime bugs where we forgot to initialize some object before it was used.

In the future, we would set the game to end after a set amount of goals. We would make the UI interactive via the Kinect. We would introduce sound effects and background music. We would then provide more settings, aside from effects settings, for sound effect mute, background music mute. We would implement local multi player – 2 or even more players depending on what the Kinect could handle – maybe even multiple Kinects involved. We would make the app distributed – we would use a server as a meeting point for the clients and the server would handle the data with a database solution. We would then complete the statistics. We would then do matchmaking and high score tables. We would then provide more settings, aside from effects settings such as arena shape. We would then implement hittable power ups which trigger effects. We would then provide more effects.

Aside from extending the game, we would write unit tests which test for 100% code coverage (every line of code is executed) and which test for values, sequences of events, and concurrency. We would improve the implementation of the application to make it more maintainable. We would make the code as loosely coupled as possible so that different modules could be independently changed.

The summarizing conclusion appears here. The conclusion is best written after the rest of the report is completed and therefore it will be written after the rest of the report is completed.