Socket Golf – Building A Google Cardboard Game In Unity

Written by Caleb Carlson

With Ruben Gamboa

Project created in partnership with Brandon Neff, Nathan Spaulding, and Cameron Leach

Senior Honors Thesis

University of Wyoming

Abstract:

Virtual reality is a new and emerging technology in the field of computer science designed to immerse the consumer into the product. To study and learn more about this technology, a four-person team of graduating seniors set out to build a mobile game for Google Cardboard. The game that was created uses the Unity game engine along with Unity multiplayer servers for the development tools. The application was designed to be run using both a Google Cardboard headset and an android controller to allow the user to control the game without removing themselves from the immersive experience.

The idea for the game came from an existing game called Rocket League, where the premise is to play soccer with a car. Throughout the design process the ideas for the game changed, but the final product, rightly named Socket Golf, is a mixture of soccer and golf. Each player spawns a bean-like character and plays a game of golf with a soccer ball. The player is provided with three different kick angles and kick strengths they can cycle through to assist them in getting the ball into the hole.

The final product of the project consists of three main components for the game. They include a tutorial, where the user can learn the controls; single player, where the user wants to make a hole in as little strokes as possible; and multiplayer mode, where two players can fight each other to be the first to get the ball in the hole.

Acknowledgements:

This project was created as a part of my Senior Design class for Undergraduate Research Day in a group consisting of four team members including myself. Without the help of Brandon Neff, Nathan Spaulding, and Cameron Leach, this project would not be possible. While this paper is my own work, their contributions to the project as a whole should not go unmentioned as several of the components of this project were created and designed by them. Gratitude and appreciation goes out to them for not only helping with the completion of this project, but for also being my best friends throughout my college experience.

Table of Contents

List of Figures

Background on VR

Virtual reality is a new and emerging technology that was created with the idea of taking real world experiences and placing them into a format for gaming and media to make entertainment more immersive. This is not to be confused with augmented reality where a device will make it appear that items have entered the real world. Virtual reality (VR) is experienced entirely in a virtual space. The technology to run virtual reality varies largely in price depending on how realistic of an experience the user wants to have, ranging from about fifteen to eight-hundred dollars. For the time being, there are two mains types of virtual reality that are available to consumers. The first type of VR is computer based virtual reality and the second type is mobile virtual reality. For this project, only mobile virtual reality was examined.

How Does Basic Virtual Reality Work?

Virtual reality (VR) is achieved through the use of headsets. How these headsets are constructed can vary, but the major components for all are the same. The main parts of the headset that allow virtual reality to work are the lenses, which are always located very close to the user's eyes. These lenses bend light and force the person's eyes to bend slightly outward, as if they are focusing on a distant object and not a screen, tricking user's brains into seeing 3D. This happens through a process call *vergence* where the brain judges distances based on how far each eye is turned in towards the nose. In the real world, as depicted in Figure 1**,** a person's eyes would then focus to see far off images, but in a VR headset this is also the job of the lenses. The lenses both help to adjust the angle of the user's eyes and focus the image properly to make an image on the screen appear as if it is far off. Now, not all people see the 3D effects as well in virtual reality based on how far in their eyes are turned. This causes virtual reality to not be as immersive for everyone. People who do not have a good sense of vergence can still see the image, but they will not get the sense of space, just a flat image.

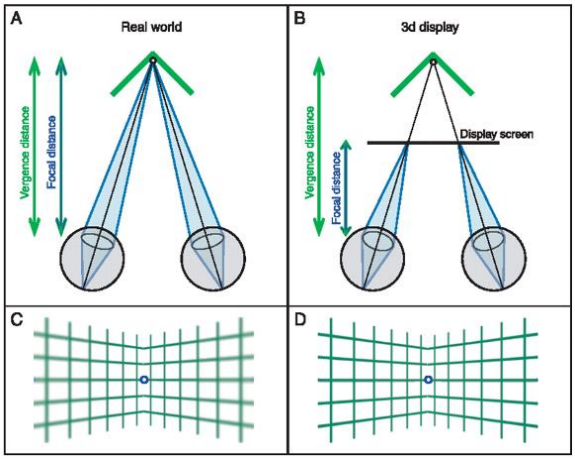

Figure 1 Eye Position and Focus Diagram

Source: Oculus Developers, https://developer3.oculus.com/blog/vr-sickness-the-rift-and-how-game-developers-can-help/

Unity – What is It, What is It Good For, and Why Choose It?

This project uses the game engine Unity for its implementation. However, without having prior knowledge on the subject, it could be difficult to understand how exactly it works into this overall project. Unity is a game engine and a game engine is the basic software for creating a game. Game engines are responsible for rendering graphics of the game, collision detection between objects, memory management, and several other options within a game. While the engine itself still requires a programmer to code exactly how objects interact between each other, such as what happens when a player touches a ball in game, a game engine like Unity assists in creating the 3D ball and player objects.

There are two main third-party game engines freely available to developers: Unity and Unreal Engine 5. Unity was used in this project due to the fact that it has a much simpler user interface and easily allows development of a game on multiple platforms. Unreal Engine 5 offered better graphics for the game, but as this project was created for mobile phones, there was no need for excessive graphic quality because phones simply do not have the hardware to support this. Unity was also chosen due to the fact that the tools they provide for developers include web addresses for multiplayer matchmaking. This saved the team a lot of time in terms of implementing multiplayer and allowed them to focus more on other aspects such as data communication between clients instead of writing the backend server to allow this communication in the first place.

Designing the Project and Google Cardboard

When the team first sat down to come up with an idea for the senior project, there were several ideas that came up. One of our common interests that we all shared was mobile development. Having all taken the mobile programming courses offered at the University, we set out to use the knowledge we had gained. Our intention was to also expand and branch out into new territories of mobile development. Another common interest in VR led us to the subject of Google Cardboard, which was something that we had not examined heavily in our classes and something we were interested in exploring further.

Google Cardboard is the name Google has given to its virtual reality platform for mobile phones. The name comes from the fact that all a user needs to view virtual reality through Googles platform is a viewer made from cardboard. All a person needs is one of the many cardboard headsets available and a mobile form to be able to experience virtual reality. This provides a cheap alternative to many other options of VR which can cost anywhere from one-hundred to eight-hundred dollars. The tradeoff is these experiences are not as expansive or immersive because phones have limited hardware capabilities and do not have controllers that can be used to track arm movement. This is In comparison to something like the Oculus which runs on a computer and has controllers to track users arm movement.

Figure 2 Google Cardboard Headset

Source: Google, https://store.google.com/product/google_cardboard

Once we had our basic subject area figured out, we moved on to determining what kind of VR game we wanted to make. One of the issues with VR is that it can make the user physically sick and render the game unplayable due to that sickness if not done correctly. We came up with a basic idea drawing inspiration from the game Rocket League ™, but wanted to make a game of our own. It was from this that we drew the idea of playing golf with cars. As eventually revealed by this paper, our final product did not turn out to contain cars at all, but instead we used bean people. That was simply done as a result of different ideas being bounced around during development.

In further discussing our project, the group came to the conclusion that for a big mobile project, we should aim to incorporate multiplayer. Many modern games these days are moving to a more social form of gaming. While multiplayer and networking did not sound like the

easiest path for our group to take, we decided that for our project to be truly considered a modern game we needed to include the ability for users to come and play together. To do this we would also need to work with android controllers which would allow the player to move around a virtual space. Head tilting simply would not provide enough controls for the fairly complicated game we had envisioned.

Controls

Controls for the game were created with the understanding that the player of the game would not be able to see the controller during play due to the fact that they would have on a Google Cardboard headset. Using this knowledge, the team designed controls to be minimalistic and intuitive so that a user would be able to play the game without giving much thought to what they were doing. Movement was mapped to the joystick and jumping was mapped to the "A" button because both controls are pretty standard for most modern games. An ability to change the kick power and angle was added to enable the player to have different options while playing, and we mapped these to the bumpers and triggers so they would be able to change these elements in the game while still moving the player. After playing the game, we discovered that due to the third person camera we had created, the ball could disappear from sight in front of the player. As a result, we came up with the idea to move the camera over the player so individuals playing the game can view from the top-down. This was mapped to the top lettered button as it would connect in the player's mind that top-view would map to the top-lettered button. We then added the ability to view the player from both the right and the left and mapped those to the right and left lettered buttons respectively, again to cement the controls intuitively in the player's mind. Controls were created and maintained via Unity's built-in manager and also with references in the code to "joystick buttons" which is how Unity manages buttons for android controls.

Figure 3 Controller Mapping

Taken From Our Powerpoint Presentation

Implementation of the Project

The Beginning

After examining both Unreal Engine 5 and Unity, a decision was made to create the game with the Unity engine as it was a little more user friendly and well documented. It also gave the team the advantage of being able to use Unity servers for multiplayer instead of having to create one for the game. This provided a huge head start when creating multiplayer features. The project began by doing research on mobile VR and understanding the components of Google Cardboard SDK for Unity. At first, there were issues implementing Google's Cardboard code, but after some research and quite a bit of troubleshooting, the basics

of the SDK were figured out. From the discovery of how Google Cardboard code worked a small demo of a car with a golf ball hole and ball was created; all viewable with a cardboard lens. It was at this point that the team split off to work on different elements and ended up with three different "demos" for our one application.

The Three Demos

The first demo, as previously described, contained a small car with the golf hole. As shown in Figure 4, this was really a proof of concept as everything was contained within a small box. From this the general ideas were derived on how to keep track of the number of strokes a player took and how to determine when the player scored.

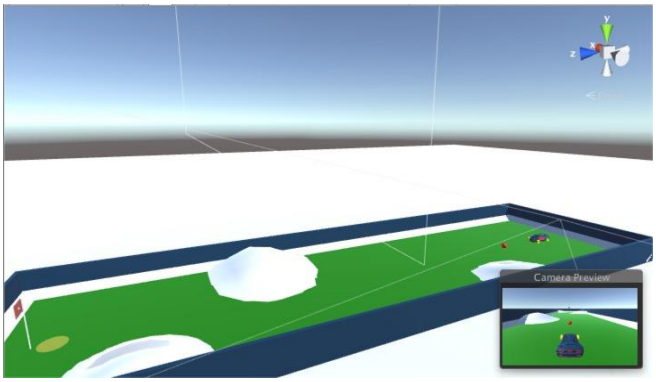

Figure 4 First Demo Of The Project

A secondary demo showcasing the physics and interaction between the player and different objects was then created. This demo, while not quite as extensive as the first, held the basics for how a player would walk up to an object and kick it. Through a relatively short script, the player, (now a bean guy) was able to kick anything into the air like a soccer ball as long as the object in question was tagged as "kickable" in the project.

Concurrently, while the first and second demos were being created, a third demo containing the basics of networking was made for the project. This was a huge step in creating a multiplayer game as no one in the group had experience writing networking code. This short demo only contained a scene in which two bean guy models could connect and be able to move around the space together. The concept was simple, but the importance of this idea was crucial in the grand scheme of the game.

Reanalysis

It was at this point that we sat down and really reanalyzed our project. We ended up swapping out our car model for our "new" bean guy model. This had been showcased in two of the demos, not only for simplicity sake; but as a way to slow down our player and prevent excessive motion sickness. With this new model, it was necessary for us to create scripts to move the player wherever the user desired which was what the previous car model, provided by Unity's asset store, already had. This provided the opportunity for us to learn how to create a script to move the player instead of using prebuilt scripts created by Unity software engineers. This also provided an excellent learning opportunity on what it takes to create a full game on our own. It ended up providing some ease in terms of networking code for the players to connect to each other. It was also in this reevaluation period of the project that the group decided to focus on three specific areas for the final product: a tutorial to show players the basics, a single player, and a multiplayer.

UNET and the Lobby

The first area we focused on in the new implementation plan of our application was the multiplayer component. We realized it would take the most work in comparison to the other areas. A new multiplayer lobby was added along with the inclusion of Unity's UNET multiplayer

to make a more user friendly experience. UNET is Unity's built in multiplayer support, and code that is added via the services tab of the project. UNET works by creating a peer-to-peer network over the internet, meaning that players essentially connect to each others devices as opposed to a dedicated server. Unity does this by setting aside a specific website or web address for communications between player's devices to go over. With this service, any player that wants to host a game will just have to create a lobby to host players, and Unity's service will allow other players, called clients at this point, to connect to the host's device. The downside of this implementation was that it can be very hard to sync data across players as it is hard to tell from a coding standpoint who is a host and who is a client. There is also some debate out there as to whether or not peer-to-peer connections are not the best for gaming due to the fact that it can give advantage to the host. Connection drops are much more likely because it is a person's device hosting a game, not a dedicated server whose only purpose is to host a game. For example, if a host quits the application the game will automatically close and return clients to the lobby because there is no host site to connect to anymore.

Originally, code for the lobby came from the Unity asset store, but we made several modifications to tailor the code to our style of multiplayer, and the group really made the code their own. The downside to using code that was not completely our own, was the fact that we were never able to figure out a great way to create a lobby in the virtual space. As a result, we instead implemented a transition scene that prompts the user to remove the phone they are using from the cardboard headset to set up the multiplayer game. With this new approach to connecting players, the next step was to build the course the players would use.

The Map and Its Components

The idea to create a randomly generated course was proposed during a group meeting and it was during this meeting that the element would be included in our project. With this design of a random course, any player would never have the exact same experience every time they played. This idea, though complicated, would also eliminate the need to create several courses for a player to use and, in theory; keep the game fresh and exciting over time. This was then implemented in the project, but not without its difficulties. Not only was the generation of a random map each time a problem in its own right, but the group soon found that when players connected to the Unity servers, they were each generating their own individual maps. This issue was fixed in time by the use of variables that were synced across each player, but provided much anxiety throughout much of the project's lifespan.

During the construction of the randomly generated map, it was also decided that to make the golf course really feel like a course, other objects, such as trees, would need to be added to the map to create obstacles for the player beyond the simple variation in height. This would, however; involve more complicated shapes than Unity would easily generate. To accomplish this, the group turned to Blender, which is a third party model building software, to create these objects. Blender was specifically chosen because it was easy for users who did not know much about model building to pick up. None of the group had much experience with model building, and it allowed for easy importation of models into Unity. After quite a few online tutorials, a tree model was generated and added to the randomly generating maps in random spots of their own.

The trees brought a new unique problem to the generation of the field. While we could get them to generate randomly in the map, they would not spawn on the ground of our map.

This was due to the fact that to generate a random map, Unity would essentially carve out a shape from the origin coordinates located at (0,0,0) and push downward to make a bowl for the player to play in. This worked perfectly for getting cool and interesting maps, but as a result; trees would be created in our map around the origin of (0,0,0) and thus be suspended in the air. To alleviate this problem we made use of Unity's raycasting system.

Raycasting

Raycasting is a tool Unity uses to determine how far objects are away from each other. This is accomplished by sending a ray, an invisible beam much like sunlight rays, from one object in a specific direction. Once the ray has been sent out it, will keep traveling in a specific direction until it collides with another object within the game world. Once this happens it is fairly easy to calculate the distance in between the objects by registering the current location of the object sending the ray with the object the ray collided into. Figure 5 gives a visualization of what happens when an object raycasts. In this picture, the object located at the camera marker is sending a raycast at an angle downward to the surface below. As stated, a raycast cannot be seen when it is sent, but in this case the blue line is a visualization of the line being generated by the object to raycast. In the case of the tree objects created in Blender, they would be generated in the world and as soon as that happened they would raycast straight downward to find the map below. Once the raycast hit the map, the distance would be calculated and the tree would then be placed on the map itself. It is important to know that this all happened while the game was being set up by the host device so the players of the game would never see the generation of the trees.

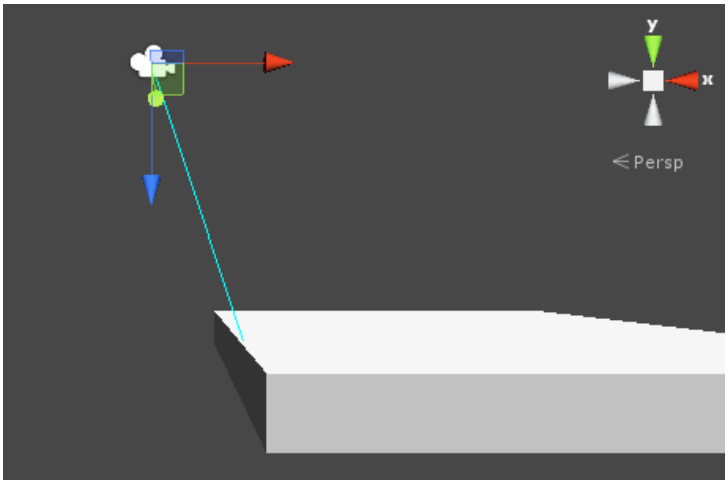

Figure 5 Visualization of Raycasting

Source: Unity, http://answers.unity3d.com/questions/475631/make-raycast-shoot-vertical.html

It was at this point that the group thought they had a solution to the issue of the two randomly generated maps, however, we would later find out this was not the case. Instead of the map passing values to all connected players, it was giving such low variation values that almost the exact same map was being generated for each user. This was resolved, however, and while one of the group members was working on this solution another was using Blender to create the final object for the board: the golf ball hole.

The Last Map Component

The golf ball hole generated its own challenges because the hole could spawn anywhere on the map at any height. To accomplish this, the golf ball hole was created in Blender. The reason it was not just created in the game was due to the fact that Unity does not have a way to create complex shapes easily. In order for us to create an actual "hole", we would need to raise the area around the hole to create a crevice in which the ball could rest. In Blender, the hole was created with a larger, extra wide base so that no matter the spawn point on the map,

the hole would spawn on the map without the bottom being seen. Unlike the trees, which were also made in Blender, when the golf ball hole was imported into Unity, the y and z axes were flipped. This caused the golf ball hole to spawn at a 90 degree angle. This was easily remedied, but raycasting had to be written to accommodate for this rotation, otherwise the hole would raycast to the nearest mountain and use that as the height.

Data Syncing and Player Values

Once all of the pieces were in place, the challenge became the same as before; which was syncing all of the map data across the multiplayer network. This was the point when we realized that the maps were not in sync. Through the use of Unity syncvars, which are essentially values that can be passed back and forth across the network, we were able to create lists of random numbers that the host player would select and pass on to all of the clients within the network. This marked the end of map creation and allowed us to move on to game elements.

The next step in the project involved implementing some of the player control elements including a varying kick power and kick angle for the player. This was relatively easy and involved consulting Unity's documentation on implementing 3D text, and the use of arrays containing the various powers and angles to cycle through. These were tied to both the triggers and the buttons on the android controller so the players could cycle through them fairly easily while playing. We decided to place these all on the right-hand side of the player in the multiplayer version so they could be visible without distracting the player from the game.

The final step in our multiplayer game was to sync up all of the scores of the players and to end the game. This turned out to be an issue of its own, very similar to the map data syncing across all players. Through study of the UNET documentation, which was few and far between, we were able to get the scores all syncing through a Dictionary variable. A Dictionary is a C# object which allows a person to map strings, which are essentially words, to values. With the implementation of this object we were able to map player names across the network with their scores. For the end of the game we used these synced variables and created a script that would show a panel in front of the player with everyone's scores as soon as they scored. Once everyone had scored, a text would appear on the panel letting the players know they would return to the lobby after ten seconds. With this last component, the multiplayer section of our code was labeled as complete.

Single Player

Throughout the completion of the multiplayer aspect of our game, two other modes were also being created: single player and a tutorial. The single player was originally planned to be built with the multiplayer code by just removing the networking code, however; this failed. Single player was then built from the ground up using the same game aspects of a random map and using the same player controls. The single player game has a par of fifteen permanently set on it while playing the game and will tell the player their score once they make a goal.

Tutorial

The tutorial was created after we had created the multiplayer game. At this point we realized just how hard it would be for a person to just pick up the game and play it without any instruction. While the controls are not too complicated, there would still need to be some sort of direction given in order for someone to play the game. An area where the player would

interact through several objectives was set up to allow them to understand the basics. Once the player finished these objectives, they would return to the main menu to start playing the game.

Analysis of the Project

We set out on our project with three goals in mind in this order of importance:

1. A fully working game running in Google Cardboard using some type of controller as our input device
2. Multiplayer support so that two people with android phones could play against each other in real time or maybe turn based
3. Leaderboards, achievements, etc. to make the app feel even more polished

Analyzing these three goals, we successfully completed our project to our satisfaction. We were able to complete our app in Google Cardboard with both a controller used as input and have multiplayer support for not only two people, but actually four. On top of this we were able to not only able to implement this game with android phones, as specified above, but thanks to Unity's multiplatform support, we were able to create the application for iOS as well. The only goal that really was not met was the addition of leaderboards and achievements in our project. We looked into implementing Google Player services in our project, but decided against this for two reasons. The first being that it would cost us around twenty-five dollars to add this support for our application, and we were looking to create this app as cheaply as possible. The second reason we decided not to do this was because it would limit our application to only be supported on android operating system phones. Instead, we were able to implement Facebook's API in a way that we could bring a player's username into the project and save their best scores to Facebook's database. Since this was our final and least important goal, we decided that this would be a good enough score keeping system for us to feel our project was a success.

Looking to the future of this project, we have a couple goals to consider as we continue to work on this project beyond the scope of this specific research project. The first addition we would like to implement is the ability to invite players to a match. With the inclusion of Facebook's API, we not only would like to save their scores, but we would like to one day be able to invite other Facebook friends to play anytime a person is online and looking for a game. This would improve on our multiplayer service and draw even more attention to the application in general. Along with this, we would love to design a network lobby from the ground up that would exist in the virtual reality space. This would prevent the user from having to ever remove their device from the cardboard headset once the game begins. This would be an improvement over the current situation because removing a player from the virtual space ruins the immersive experience of VR.

Another idea we would like to pursue in the future is the inclusion of power-ups in our game. This would allow players to do things such as move faster, jump higher, or maybe even line up their kick for them no matter the angle. This would also add another fun element to our game that would bring people back to our application. We had talked about including this element from the very beginning of the project, but time did not allow for us to include this in our existing implementation.

Conclusion

Virtual reality is a new and emerging technology that can be used to create new entertaining experiences for multiple platforms. Mobile games are a market that allows for quick profit and success, however; there is a lot of work that goes into creating these experiences. It took our team about eight months to design, create, and implement this fun little mobile application and while it was a complete experience, it is a little shy of what we really planned. It is apparent to us why it takes years or months, with a much larger team than what we had, to create these immersive experiences. This experience has taught our entire group what level of dedication it takes to create a complete product from the origination of an idea to its final project. Not only that, but the team has learned how to develop in C#, use Unity's game engine, write working networking code, design a game, build the models, and work with android controllers all within the past several months. This has given us great experience in taking initiative in learning new computer science skills and what it takes to be a successful team out in the real world. We have all deemed this project a success in terms of completing our college careers and taking that next step into the professional business computing world.

References

1) Blender Foundation. 2017. Tutorials. (2017). Retrieved March 7, 2017 from https://www.blender.org/support/tutorials/

2) Jonathan Williamson. 2017. Blender Basics. (2017). Retrieved March 7, 2017 from https://cgcookie.com/course/blender-basics/

3) Tom Forsynth. 2013. VR Sickness, The Rift, and How Game Developers Can Help. (July 2013). Retrieved April 15, 2017 from https://developer3.oculus.com/blog/vr-sickness-the-rift-and-how-game-developers-can-help/

4) Unity Technologies. 2017. Unity User Manual (5.6). (2017). Retrieved April 15, 2017 from https://docs.unity3d.com/Manual/index.html