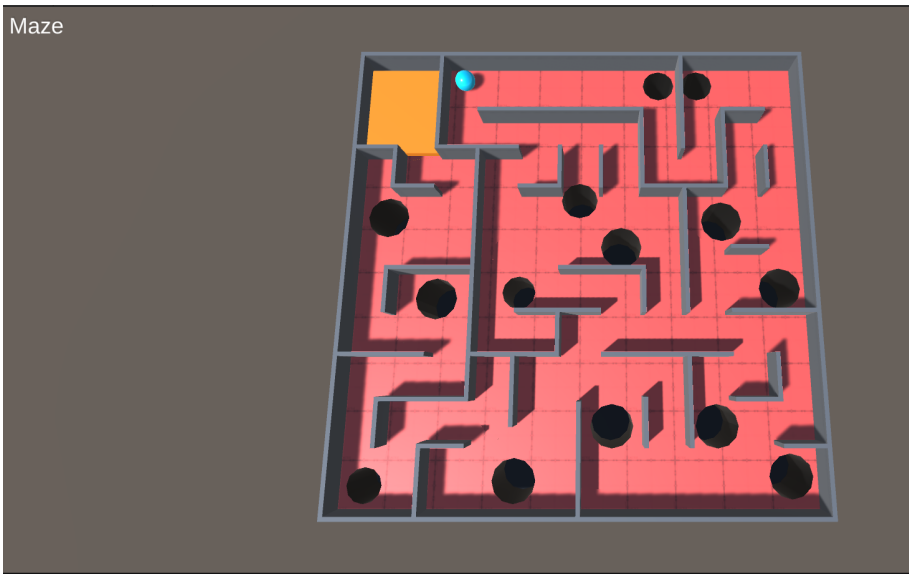
Labyrinth Simulator

Team #06

Steven Johnson, Jake Parker-Howe, Jacob Ross

The Labyrinth Simulator is a simple-to-launch, simple-to-play computer application designed to give users an alternate way to play the popular handheld labyrinth marble game. To successfully implement our desired vision, we needed to develop the environment, the board, and the physics to feel as realistic as possible. The program takes the form of an executable that can be launched locally for any individual to enjoy. This application is solely meant for entertainment purposes. While we were not able to develop a large selection of maze paths for the user to choose from, we did create a strong infrastructure for a simulator game in Unity that can be easily updated and upgraded.

As seen in the image below, the application is very simple and straight to the point. There is a marble and a maze. In order to win, the user must tilt the surface in such a way that the marble rolls into the finish line, as designated by the yellowish orange area. Via user controls, the maze will be tilted and the ball will roll accordingly. For example, if the user were to hold <D> or the right arrow key with the current orientation as shown in the image below, the maze would tilt to the right and the ball would roll toward the right as well. In the situation that the ball falls into a hole, the game would be reset back to the position seen, also known as the starting position. Upon reaching the finish line, there is a quick cut-in announcing that the level was complete. After a 2 second delay, the level is restarted.



To prepare for the development of our application, we first had to study and learn the basics of computer graphics. In our case, we were specifically dedicating time to learning the fundamentals of Unity. At the start of our development phase, we dedicated two weeks to our own individual learning efforts. We researched and found useful YouTube videos and webpages to assist us in this process and we shared them amongst each other. With this reserved time frame we were able to gain a sufficient amount of knowledge to apply.

After the preparation phase, we adopted a Unity Collaborate program to allow us to work on our application individually while also staying up to date with each other's efforts. Similar to a Github repository, we were able to make changes to our local files and publish them on the Collaborate server, making sure to take care of any file conflicts that arise. With this format, we

were able to delegate specific tasks to each of the members of the team and diligently resolve our own issues and develop our own portion of the project. Our initial goals were to create the maze itself, add a marble with working physics, and implement controls for the user. We were able to complete these desired goals within a reasonable timeframe (before our first demonstration). We were hoping to have more features implemented in our application before considering it finished, such as a main menu screen and multiple maze designs with varying difficulties, but we quickly fell behind on the extra tasks and worked to create a more barebones version of the product.

Over the course of our project, we have not needed to make any major or monumental changes to our project. We have noted some things that definitely could be adjusted, but we rolled with the punches and committed to our goals and strategies rather than spending more time changing features. For example, the first maze model was designed and created within Unity, using packages made by other developers. The bottom plane was generated by subtracting circles from specific areas of the flat surface. Although a working solution, it is far from optimal, as placing too many circles in a surface eventually will cause the program to crash. This means that large maze designs will not be possible with this format.

The maze object itself is made from an accumulation of basic and more advanced meshes. The walls of the labyrinth board are made from a simple, basic mesh in Unity that acts as a barrier or wall. The marble is made from the same mesh but, obviously, with a spherical shape. This allows the marble to roll into the walls and stop from falling outside of the maze. The last basic mesh is the invisible wall above the maze. The invisible wall acts as a safety net so that the marble does not fly out of the maze. We had issues in the past of the plane moving so quickly and the physics we implemented not interacting as hoped or intended. Although the invisible wall may not be necessary anymore, we are keeping it in case anything goes wrong. The most advanced mesh, as previously described, is the main plane of the labyrinth. To create this object, we utilized a Unity package known as ProBuilder. ProBuilder, along with additional packages, allowed us to take two pre existing objects and use them to subtract one from the other. To create our plane, we used a cylinder object to subtract from the plane at specific parts of the plane, which created holes.

The physics of the ball happened to be one of the hardest things to implement. In previous iterations, the ball would fly outside of the labyrinth at an insane speed, hence the addition of the invisible ceiling. We found a new and current solution, which was to tweak the gravity of the game by a great amount. We tested this hoping to fix the problem of it flying out of the maze, which it did, but it also seemingly adjusted the rolling physics to play much more smoothly and with greater control over the ball.

Our application is developed in Unity with C#. When built, the application becomes a simple .exe file that has very low computer requirements, so nearly anyone can run the application on their device. As long as the user has a keyboard or joystick plugged into their computer, they should be able to play and enjoy Labyrinth Simulator.

By using the Unity Collaborate feature, each team member was able to connect to the same project and publish their changes as they worked on it. Within our Unity project, we have

various folders for each part of the application, including input, materials, ProBuilder, and scripts. The input folder is Unity's built-in control, but we ultimately use the scripts to manage the movement of the plane.

We have C# scripts for camera control, plane control, player control, rotator, and the pause menu. The rotator and plane control scripts work together to manage the tilting of the labyrinth. As the user presses WASD, arrow keys, or uses a joystick, the scripts adjust the speed and rotation of the rigid body objects, including the walls, floor, and ceiling of the maze. Each of these objects maintain the same orientation in relation to each other the entire time. The camera control script was used during the first weeks of development to understand how the scripts affect the application, but is now unused. The player control script is used to place the ball at the initial position at the start of the program, when the ball falls into a hole, and when the level is restarted via level completion or the "Restart Level" button. Finally the pause menu script freezes the in-game time and brings up an interactable pause menu when the proper key/button is pressed.

The materials and ProBuilder folders house the assets for the marble and wall textures. At this first release, the textures are very basic and lackluster. The walls are a simple gray, the ball is a basic blue, and the floor is the default red texture that ProBuilder makes. In the future, these textures will be changed and the appearance of the game will be much more pleasant. As explained briefly before, we did not want to place great effort into areas that would be majorly overhauled in the future anyway. Retexturing the labyrinth would make it look much nicer, but when we overhaul the models, it would need to be textured again.

The current model for the labyrinth works, but in the future, to make the addition of larger mazes possible, we would like to branch outside of Unity and create our models with a CAD software. This way we will not risk the chance of our Unity crashing while placing holes in a ProBuilder object, and we can more precisely place the holes on the map, which will help with differentiating the difficulty of maps. This would also allow us to texture our models very easily since we found 3D modeling outside of Unity to be much nicer overall.

We had a number of planned features that did not make it into the final release of the game. We originally planned to have the game load the main menu at launch which would allow the user to adjust settings and manually select different levels. This idea was scrapped because most of the buttons and scripts were not functioning as intended, and with the deadline closing in, we decided to put it to rest at the last minute. The remnants of this scrapped idea are still seen in the pause menu where there is a "Quit to Menu" button that is not implemented. Another feature we scrapped was checkpoints. Initially we planned to have multiple levels with ranging sizes and difficulty scales, which would make checkpoints a necessity. However since we only have one level which is quite easy and takes hardly any time to complete, we decided that checkpoints were not worth the effort. The next big idea we scrapped was additional hazards to the player such as a piston-like wall that would bump the ball out of the labyrinth, thus resetting back to the start or most recent checkpoint, or even parts of the floor that disappear on a timer. In the future, we could host the game on a website that would open up co-op/versus play and allow

individual players to track their stats and progress, and see how they stack up with others in an online leaderboard.

Ultimately, our goal was to create a functional, easy to update, and fun game that simulates the labyrinth marble handheld game. Despite needing to spend time learning computer graphics, running into various speed bumps during the development of the project, and juggling the creation of the project with other college courses and projects, we successfully released a version of the game that we are content with and proud of. The game physics are comparable to that of real life, so it feels very natural to play and the controls are simple and intuitive. There could definitely be some aesthetic improvements to our game, including a pretty background or hands that hold the labyrinth and move when the user tilts the board, but the product that we are releasing now is functional and entertaining.

The first team member to note is Steven Johnson. Steven played a crucial role in this team as he initially discovered Unity Collaborate and locked down an efficient way for all of us to cooperate together on this project. The pause menu was a feature that was implemented solely by Steven, and the menu does its job very well. His availability throughout the development period of the project has been very clutch, as he has helped greatly with last minute changes and ensuring we meet deadlines. Next, Jake Parker-Howe was the first member to put the pedal to the metal and begin development. With the help of various Unity tutorials and YouTube videos, he managed to create a minigame with a rolling ball and collectibles on a basic plane with user input. Most of this gained knowledge was refurbished and used within the current game. He is responsible for the scripts that make the gameplay feel as smooth as it does, including the controls and physics of the game. Last, Jacob Ross has acted as the lead documenter for the team, as well as the one to create the maze plane via ProBuilder. After creating the plane, he tested the gameplay and wrote the player control script to reset the game when the user falls into a hole. Team members were generally tasked with their own subprojects, but ultimately we communicated constantly to ensure we were meeting the project's desired outcome.