Hide/Seek

Steven Dill, Josh Kiesa, Austin White

**Introduction:**

The core concept of our game was to make a multiplayer rendition of the classic game Hide and Seek. In our version, one Hider is pitted against a team of Seekers who are trying to tag him. The player is in a digital world trying to escape the eyes of his pursuers.

**User Controls & Game Info:**

Players move around with WASD, and their camera is controlled by moving the mouse. Space makes the player jump, hold left shift to run. Clicking the left mouse button while playing as a Seeker attempts to tag the Hider.

The goal is for the Seekers (red) to catch the Hider (blue) in the maze. Seekers spawn at the base of a ramp they must climb. After climbing the ramp, Seekers may choose how they enter the maze. They may go in through the near sides, or go above the maze to enter from the far sides.

Powerups (large green orbs) are scattered throughout the maze that give the Hider a speed boost. Seekers may also grab the powerups to deny them from the Hider, however Seekers do not get the speed boost.

**Programming Structure:**

The game was initially based off the MU 0.5 and lobbyDemo code received in class. Very little of this code still exists, as much of it has been repurposed to suit our needs.

**Steve:**

The game uses Unity’s Lobby Manager to handle player spawning. Very little is editted as we did not have time to create a custom lobby. Player Spawning is handled primarily by PlayerNetStartup. In its start method all of the player’s proper scripts are started. Using a Synced bool, the script determines whether the current spawning object should be a Hider or Seeker. The first in becomes a Hider, and then sets the bool to true. Afterwards the characters are assigned the proper color.

To handle the syncing of Player position data, we used the syncing scripts provided in MU. However, we were forced to readjust the lerp rates and some of the logic statements to reduce lag in our game.

Tagging in the game is handled by the Seeker script, which is added to every Seeker player object. When a Seeker clicks, it sends a command to the server to check the distance between the Seeker and Hider. If the Hider is within reach, the script deactivates the Hider’s controller, freezing it.

**Austin**:

The map generation code is done through the application of a SyncList. I create a SyncList that is populated only on the server. SyncLists can only be populated with basic variables such as an int. In this case I used an int to represent a game object. I created an array of prefabs, whose index is used in the SyncList. I populated the SyncList with a random number between 0 and the size of the GameObject array, inclusive of 0.

IE.)

Terrain1 = GameObjects[0];  
 Terrain2 = GameObject[1];  
 for (int i=0;i<NumberToGenerate;i++)  
 {  
 SyncList[i] = random(0,GameObjects.Size() ;  
 }

After populating the SyncList, I created a method to implement the map I generated. This required a position of where to generate the referenced GameObject. I decided to instantiate the terrain in sets of 5x5 squares. I did this by nesting 3 for loops. The first of them was a for loop from 0 to the number of 5x5’s I wanted to instantiate. Inside of that was Two for loops from 0 to 5. Using this I could instantiate the game objects in a 5x5 square based at a position stored in an array of vector3’s.

**Josh**:

There are three types of powerups: a speed boost, a speed reduce, and a freeze. The speed boost does exactly that, gives the player who picks it up a burst of speed for a period of time. This can be grabbed by any of the players. The speed reduce and freeze however can only be collected by the Hider. When the hider picks them up the ability is applied to all the Seekers, either a reduced speed or being completely frozen and cannot move at all.

The powerup system is organized in the PowerupManager script along with a Powerup script and a script for each specific powerup. Each powerup specific script updates the abilityID for its referenced power (1=Speed Boost, 2=Speed Reduce, 3=Freeze). The Powerup script handles the synchronization across the clients. It contains code to update the position of the prefab whenever it changes.

The Powerup Manager is where the main gameplay code occurs. This handles the spawning of each prefab as well as checking for collisions between the players and the prefabs. All the prefabs are contained in a List and every twenty seconds a random index is retrieved and spawned over the server until each powerup has been spawned. This powerup is then moved to the spawnedPowerups List and taken out of the first one.

**Testing & Feedback:**

Due to a major persistent issue (detailed below) we were unable to conduct usability testing in class and throughout the semester. The only testing done were small playtests among 3-4 people to test the viability of the game.

**Development Issues:**

A major issue occurred in our project shortly after beginning. The issue seemed to be an improper connection from the client to the server, even when running on the same PC. Many attempts were made at resolving the issue, including converting to Unity’s lobby system. This partially solved the problem, however players could not spawn properly as separate teams still. According to the logic given in Unity’s documentation, several solutions should have worked. This includes applying scripts in the LobbyManager (like OnLobbyServerSceneLoaded) or using Unity’s LobbyHook system. Even using ClientRPC calls did nothing, and actually resulted in the least successful build.

Eventually the current solution was worked out, by doing the changes in PlayerNetStartup. As this was only finally completed in the last two days of class, development was severely hindered. The current, and final, build of our game achieved a completion that could be comparable to a second or third milestone build. Features were heavily cut back, and few to no stretch goals were met. However, the game is playable and fun, albeit with little additional content.

After generating a map index on the server, clients would not receive the map and would in turn instantiate nothing. This was occurring because the syncList was not being properly instantiated, nor was it properly being populated. After solving this issue I was still running into the problem that nothing would be instantiated on the clients. This was solved by moving the Generation code into the Awake() function, and call the Instantiate() function within the Start() function.

Also, some of the powerup functionality had to be taken out. Only the Hider was able to pick up and gain the ability. The Seekers could collide effectively but the abilityID, which controls the ability that the player currently has, would not update correctly. We were not able to solve this issue after many hours of debugging so we took out two of the powerups, leaving only the Speed Boost, and made it only attainable by the host. We changed the seeker’s collision so the powerup would change position when the seeker collided with it but they don’t gain the ability.

**Lessons Learned:**

Initial goals:

* A single hider against a team of seekers.
* The Hider and Seeker would gain score depending on their actions during the round.
* The game would be a series of rounds, swapping who the hider would be.
* Tag only occurring when the players are close, with nothing obstructing the players.
* Power-ups spawned consistently in the map, with multiple types and effects
* Randomly generated maps
* Different themes for maps
* A relatively lagless environment
* Working chat
* Third-person Hider camera
* Custom Lobby
* Stretch Goals

What we ended up with:

* A single hider against a team of seekers.
* Tags only occurring when the players are close
* Power-ups spawning around the map with only one side being able to pick up and use them.
* Randomly generating a maze on the map.
* Chat was removed
* Bare-bones lobby
* Basic UI and art assets

**Next:**

If we had more time we would like to reduce the lag, and make sure players positions are very in sync. We would also focus on refining the tag, only allowing it to tag if the hider tagged is in front of the seeker, with nothing in the seeker’s way. We would try to have the core of the game working before adding in things like score, rounds, and team switching. Additionally, we would fix the powerup bugs so we can incorporate the original powerups into the game. We would also work on improving ease of use features, like restarting the game or returning to the lobby. A custom lobby would also be created.

**Individual Work:**

* Steven - Lobby System, Player Spawning, Debugging, Source Control/Merge Issue resolution
* Austin - Map Generation, Art Style, Clock, Win Conditions
* Josh - Powerup System, Doc Writeup

**Final Note:**

As has been stated throughout this document, the major bug encountered early on completely halted development. This bug was finally solved 5 days before the final submission date, leaving very little time to combine the individual work done by the three group members. We had ambitious goals for this game, and are disappointed with the final result. However, we are at least satisfied with the fact we were able to get the game up to a level of complete playability in such a short time. By this, we mean the game has its core mechanics, a set of win/loss conditions, and fun gameplay all fully realized. Although it is incomplete, the basic game was created. We believe this fact, and our continued dedication throughout the semester to solving these problems should be taken into consideration.