

CSC2034 Game Engineering Project Report

Abstract

This report is written as part of my assessment for CSC2034. In it, I aim to accurately present my work on the Game Engineering topic project, including new scenes, along with new or updated prefabs and scripts, explaining how the artefact reached its final state. Moreover, I give my evaluation of how well the artefact meets the task description and perform some self-reflection, expressing what I could have done better, as well as what I learned from the experience as a whole.

Stefanos Larkou (c0034805)

Breaking Down My Work

In this section of the report I will go over all of the project's scenes, describing the purpose and functionality of each, as well as that of the prefabs I either created or modified. From this point on, when referring to "the original version of the game", I am talking about the game as it was at the end of the Game Engineering practical.

Game Scenes

MenuScene

The main menu of the game and the first thing the player sees upon launching it. Its purpose is to allow the player to navigate to any part of the game they wish. Clicking play will take them to the game mode selection scene, scoreboard takes them to the scoreboard scene and exit quits the application. The player can return to this scene from almost any other scene.

ScoreboardScene

The scoreboard table was likely the one aspect of the game I spent the most time and effort on. I completely scrapped the high score table from the original version of the game and instead made a scene dedicated to it. This happened for a few reasons. Firstly, having the scores as a layer over the game did not seem too practical, as with the original implementation the scores would eventually go out of the bounds of the screen. Secondly, as more levels and game modes were implemented into the game, managing which entries would need to be shown became increasingly complex. Lastly, it just seemed much more pleasing to the eye.

I also scrapped the text files containing the high score information. Instead of that, the information is now stored in a binary format (Stuart's Pixel Games, 2020 and Zdravko Jakupec, 2015) in .myf files (since the file extension does not matter I made the files .myf after mellow yellow fellow). I chose binary over .json (Prasetio Nugroho, 2019) or .xml (John Leonard French, 2021) as it is the hardest for the average player to manually edit. These files are stored on the player's computer using `Application.persistentDataPath`. They are updated whenever a submission is made and loaded whenever the player enters the scoreboard scene or makes a submission (to select the correct file, as there are multiple levels), using functions in the `SaveLoadSystem` script and the `ScoreboardData` class. The score entries of each level are stored in separate files, including the time-based game mode's times. The hardcore game mode's scores are all stored in one file, because of how the game mode works (more on that under `GameModeScene`).

```
public static void SaveScores(int level, string mode){
    BinaryFormatter formatter = new BinaryFormatter();

    string path = FilePathScores(level, mode);
    FileStream stream = new FileStream(path, FileMode.Create);

    ScoreboardData data = new ScoreboardData();

    formatter.Serialize(stream, data);
    stream.Close();
}
```

The function that handles saving score entries in binary

```
public static ScoreboardData LoadScores(int level, string mode){
    string path = FilePathScores(level, mode);

    if(File.Exists(path)){
        BinaryFormatter formatter = new BinaryFormatter();
        FileStream stream = new FileStream(path, FileMode.Open);

        ScoreboardData data = null;

        if(stream.Length != 0){
            data = formatter.Deserialize(stream) as ScoreboardData;
        }

        stream.Close();

        return data;
    }
    else{
        return null;
    }
}
```

The function that handles loading score entries stored in binary

It should now become clear that displaying the entries is not as simple as grabbing them from a file and sorting them. There are buttons allowing the user to pick which game mode's scoreboard they are interested in. There are also buttons for picking a level. Because of the limited space, the scoreboard only contains up to ten entries at a time, but if more exist, next page/previous page buttons appear. This scene can be accessed through the main menu, or the game over screen of any level.

GameModeScene

This scene is loaded when the player clicks the play button in the main menu and only contains a few buttons for selecting a game mode:

- **Original:** The game mode from the original version of the project. Pick up pellets for score, at the end the score is submitted to that level's scoreboard. ScorePowerup provides some variation in the results when winning.
- **Time-based:** Play to complete the level as fast as possible instead of playing for score.
- **Hardcore:** Only one life to play through all of the levels (you cannot pick a level; you have to start from level 1). As such, there is no level split for this game mode's scoreboard. If you lose, you go back to the beginning.

SelectScene

After selecting a game mode, SelectScene is loaded, which allows the player to pick the level they would like to play. If a level has not been completed yet, the ones after it are locked and completed levels are saved similarly to the score entries (see ScoreboardScene section). While you can go through all the levels in each game mode, they only become available to select if the previous level has been completed in the Original game mode, giving it some more value than the rest. If Hardcore mode was picked, only level one will be unlocked because of how the game mode works, regardless of how many levels were completed in the Original game mode.

LoadingScene

This scene is loaded before any level is supposed to be loaded. It features a loading bar that fills up as the level's scene is loaded (Prime Games Bulgaria, 2017). As this is quite a simple game though, this scene is not visible for more than a second or two.

Level1Scene-Level5Scene

The rest of the scenes are the game levels. The only thing that needs to be covered here which has not been or will not be analysed is the user interface and the game over panel. The GameUI object contains the main menu button, whose purpose is obvious, and some TextMeshPro UI objects. Except for the current level, the text these objects display can and will change for most throughout the game. The UI includes the current life count, current score, and current high score for the level. If the game mode is Time-Based, it displays current time and best time for the level instead, and if it is Hardcore, it shows the high score for completing all the levels in one go.



Original Mode, UI displays score



Time-Based Mode, UI displays time

When the player completes a level (or in the case of Hardcore mode all of them) or loses, the game over panel pops up. There, the player also types in their name to submit their score to the scoreboard, and after that, if they lost they are given the option to retry the level, or if they won to move on to the next level (if there is one). From the game over window the player can also navigate to the scoreboard scene, or the main menu. The only limitations as to what can be input as the name for the scoreboard entry is that it cannot be empty (or filled with whitespaces) and it cannot exceed eight characters. In my opinion, this gives the game more of an arcade feel.

Prefabs

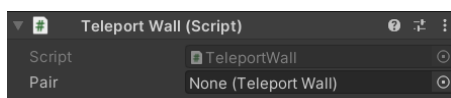
ScarePowerup-ScorePowerup-SlowPowerup-WallPowerup

All are essentially the same prefab as Powerup was in the original version of the game, just with different default materials so they can be told apart (the ScorePowerup is also slightly bigger). I have also renamed the Pellet script into Collectable and added it to these powerups as well, so they can only be obtained once.

TeleportWall

The task of making it so the player and the ghosts can teleport using the tunnels was fairly straightforward, but it did have a couple of complications that I will get into shortly. The simplest way to complete it was to create invisible game objects on each side of the map that act as colliders, with a script to teleport the player or the ghosts to where the opposite end is. The only issue was that I decided to make multiple levels, all of which had multiple pairs of tunnels, so creating a script for the tunnels of each level was not practical.

For this reason I ended up creating a new prefab for the tunnels, which comprised of a BoxCollider component and a generic script that would work for any pair of tunnels. The last issue is that these teleport triggers being on the edge of the map work well for teleporting the player, but the ghosts which can only navigate the NavMesh will never use them, as they just cannot set a destination outside it. The fix for this was simple, just move the TeleportWalls in a little, which works surprisingly well, both when the ghosts are wondering and when they are chasing Fellow.



TeleportWall pair set through the inspector window.

```
if(pair.transform.position.x > transform.position.x){
    other.transform.position = new Vector3(pair.transform.position.x - 1.5f, other.transform.position.y, transform.position.z);
}
else if(pair.transform.position.x < transform.position.x){
    other.transform.position = new Vector3(pair.transform.position.x + 1.5f, other.transform.position.y, transform.position.z);
}
else if(pair.transform.position.z > transform.position.z){
    other.transform.position = new Vector3(transform.position.x, other.transform.position.y, pair.transform.position.z - 1.5f);
}
else if(pair.transform.position.z < transform.position.z){
    other.transform.position = new Vector3(transform.position.x, other.transform.position.y, pair.transform.position.z + 1.5f);
}
```

Teleportation logic.

Wall

A simple script was added to the wall prefab to switch between being transparent or not depending on if WallPowerup is active.

Fellow

The Fellow prefab itself has not seen any change. However, its Script component has seen quite a bit of modification. A lot of new fields were added so that the game's behaviour changes when playing different game modes. The part of the script that is the most different though is the OnCollisionEnter function. There are now also checks for all of the alternative powerups mentioned previously. Here is what happens when Fellow collides with each of them:




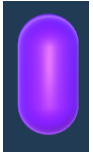
- **ScarePowerup:** The powerup from the original version, when activated the ghosts will run away from the player. It also allows the player to "eat" the ghosts, making them run back to the ghost house and stay there until the powerup is no longer active. Changes ghosts' colour to blue to indicate that it is active and makes the ghosts move faster.
- **WallPowerup:** Allows the player to pass through walls. When picked up, the walls also become partly transparent to indicate that the player can pass through them, as well as so they can see where they are. To avoid the player getting stuck inside walls when the powerup time is up, there is an extra condition that the player is not inside a wall for it to become inactive.
- **SlowPowerup:** Makes the ghosts slower. Changes the ghosts' colour to red to indicate that it is active. ScarePowerup takes precedence over this powerup.
- **ScorePowerup:** Doubles the score earned from pellets while active.

While all powerups' timers are in the Fellow script, ScarePowerup and Slowpowerup's effects are handled in the Ghost script.

Since Fellow now has multiple lives, colliding with a ghost does not end the game. If Fellow has lives left after colliding with a ghost, the positions of all characters are reset and a countdown timer of 3 seconds appears before the game resumes.

Ghost

The Ghost prefab is in a similar situation as the Fellow one in the sense that little change has come in terms of adding or removing components but the script has changed dramatically. In the Ghost script almost all alterations are in the update function. All changes here have to do with ghost movement. As I mentioned when I was describing the powerups, ghosts can now be eaten, and when they are they go to the ghost house. Ghosts now also have a scatter timer. When the scatter timer reaches a threshold specified in the Unity editor, the ghosts will scatter to a position also set manually through the Unity editor (usually a corner of the map) before continuing their normal movement. ScarePowerup takes precedence over this. After the ghosts scatter, the threshold to scatter again increases by an amount specified in the Unity editor. All ghosts in each level have different values for the threshold and the threshold increase. Furthermore, there are now different tags for ghosts to make for different behaviour (Jamey Pittman, 2009):

Tag	Ghost	GhostIgnorant	GhostChaser	GhostMad
Mesh-Material				
Speed compared to Fellow	Faster	Much faster	Slightly slower	Faster (but slower than Ghost)
Movement	Random unless it sees Fellow, in which case it chases. Affected by powerups normally.	Completely random. Affected by powerups normally.	Always chases Fellow. Affected by powerups normally.	As with Ghost but unaffected by powerups and chases if one is active.

Description and Evaluation of the Final Artefact

Description

The final artefact is a very much playable (but not especially good looking) game, at least through the Unity Editor. It could be tweaked to work with different screen resolutions as a .exe though. It features a main menu, a scoreboard screen to view scores for all levels and game modes, three different game modes and 5 levels for the player to go through. The player's level progress and score submissions are saved on their machine so they can be loaded during later plays of the game. The goal of each of the different game modes has been defined under GameModeScene in the Game Scenes section. There are now ghosts with different behaviours, as well as more powerups. Ghosts can also be eaten now, if the player has picked up the ScarePowerup. Fellow and the Ghosts have the ability to use the tunnels of each level to teleport from one side of the map to the other, instead of simply falling off. Furthermore, a user interface with information about the game is now present.

Evaluation

When comparing the final artefact to the task description, I would argue it holds up quite well. Most tasks have been completed – tunnel teleporting, ghosts returning to the ghost house after being eaten, game UI, saving of scores. I did take the “new level” task a bit too literally, nonetheless I am proud of the end result. In addition to the tasks, I have completed several of the extensions – more powerups, ghosts with different behaviours and different game modes.

Conclusions

Future Work

While the artefact meets the criteria set by the tasks, it is far from a perfect game. As with any game ever made, there are lots of things that can be done to improve or expand it.

What I am the least proud of in my project is the Ghost script's Update function. It is about 100 lines long, and way more complicated than it needs to be. That is 100 lines per frame, in a project where the second longest Update function is about 20 lines long. It can be split into separate scripts for each ghost behaviour (so can the Ghost prefab for that matter) and even then it can still be rearranged to be more efficient.

Furthermore, I feel that the ghosts' navigation decision-making could be improved to be more intelligent, instead of being random, chasing, hiding or scattering.

Apart from that, the game can be expanded by adding music and more sound effects, more levels, more powerups, more game modes.

Personal Reflection

I definitely was not as efficient as I could have been. Prime example of that is the Ghost's update function I talked about previously. There are also other scripts like the GameOver script which could have been handled better by instantiating buttons instead of constantly searching for them in the scene and setting them as active or inactive.

I mentioned in my reflective log for the Game Engineering topic that I had used Unity a little before, but only on a fundamental level and had never done any scripting. This project has changed that. As a result of working on the tasks of this project, I now have a much better grasp on what a game object is, learned about many useful components, as well as how to manipulate all of these through scripting. Moreover, I have widened my knowledge in regard to using and managing scenes. As a consequence of that, I now know the difference between the Awake and Start functions and when they are different in practice.

References

Stuart's Pixel Games, How To Do Secure Saving with Binary – Unity C# (2020)

Available at URL: <https://stuartspixelgames.com/2020/07/26/how-to-do-easy-saving-loading-with-binary-unity-c/>

[Accessed 7 May 2022]

John Leonard French, How to keep score in Unity (with loading and saving) (2021)

Available at URL: <https://gamedevbeginner.com/how-to-keep-score-in-unity-with-loading-and-saving/>

[Accessed 7 May 2022]

Prasetio Nugroho, Saving Data as JSON in Unity (2019)

Available at URL: <https://prasetio.medium.com/saving-data-as-json-in-unity-4419042d1334>

[Accessed 7 May 2022]

Zdravko Jakupec, Saving and Loading Player Game Data in Unity (2015)

Available at URL: <https://www.sitepoint.com/saving-and-loading-player-game-data-in-unity/>

[Accessed 7 May 2022]

Jamey Pittman, The Pac-Man Dossier (2009)

Available at URL: <https://www.gamedeveloper.com/design/the-pac-man-dossier>

[Accessed 7 May, 2022]

Prime Games Bulgaria, HOWTO: Create a loading screen with a loading bar in Unity

Available at URL: <https://www.primegames.bg/en/blog/howto-create-a-loading-screen-with-a-loading-bar-in-unity>

[Accessed 8 May, 2022]