DES310 Personal Portfolio

Cameron Thustain 1900594

Team – Sosig Studios (Team 04)

Role – Programmer (Gameplay, UX, Tools Audio) & Level Designer

Client Brief – L2, Anteaters and Highways

Gameplay Video (For Evidence) - https://youtu.be/iolA0Aq3CaM

Contents

Research

1a. Game Inspiration/Similar Games

For this project a lot of research was done to build a good idea of what the team's vision and scope was. We drew a lot of inspiration from similar games with similar target audiences. One game with similar features and mechanics was the 2014 mobile game by Hipster Whale "Crossy Road". In this game a player takes control of an animal by tapping and swiping their mobile screen to cross a road and avoid cars and trucks (Crossy Road - Endless Arcade Hopper Game, n.d.). Originally the team's idea was to create a game where the player would take control of an anteater and avoid cars but after some research and discussion we felt that the tone of the game wouldn't be right. This is when it was decided that the game's main mechanic would be users controlling the cars rather than the anteater. The tone / theme changed from "surviving" to "preservation" instead.

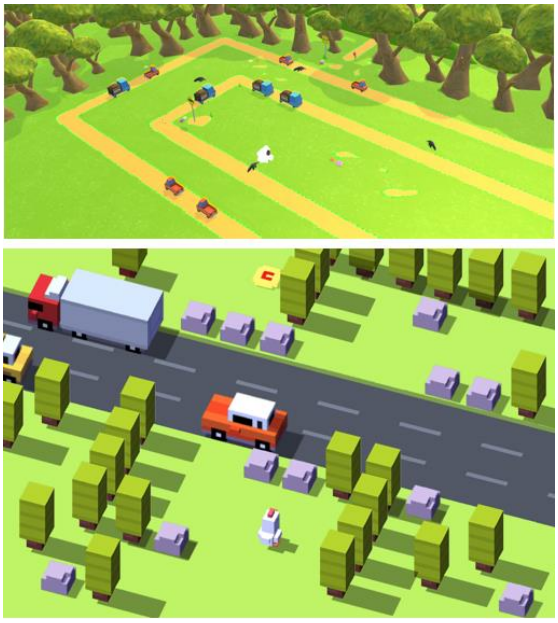

Figure 1 Anteater Highway LVL 1 & Crossy Road LVL

1b. / 1c. Target Audience & Target Platform

The clients ideal target platform was their website Zoodle (Discovery and Learning, n.d.). Because of this, many old web browser games and websites from the team's youth were investigated to give us some inspiration for example Bloons Tower Defence, Cool Maths Games or Friv. What was found was many of the games had little controls and were easy to use for their target audience which like the clients were children. Furthermore, the clients wanted an average playtime of 5-10 minutes which is roughly how their other games play as well. (Cool Math Games - Free Online Math Games, Cool Puzzles, and More, n.d.), (Friv : Free Games!, n.d.), (Ninja Kiwi - Free Online Games, Mobile Games & Tower Defense Games, n.d.)

1d. How Research Impacted Decision

When researching and looking at some other web browser games, it was found that a lot of them had short playtimes and if they didn't it was made up for being up to drop and pick up the game again with ease in the future. It was these key parts of research which helped decide that for the game we would like to keep some controls to a minimum and decided the main mechanics of the game would be controlled by mouse clicks and design the game to be frantic and quick to play.

Game Mechanics & Features

2a. Vehicle Game Mechanics

Vehicle Movement

One of the main mechanics within the game was vehicle movement. Being one of the main inspirations of the game Crossy Road was researched to see how this mechanic could be included in a similar fashion. Within Crossy Road the cars move themselves along a road while the player must move the animal when the time is right. For this I investigated different ways to create a patrolling game object. This is a simple yet effective way to create an object that follows a scripted path and can be modified if need be. To create this mechanic for not only the cars but also the anteaters a simple patrol mechanic tutorial (Studio, 2020) was used to help with this feature. This being one of the first mechanics implemented into the game with no prior knowledge in Unity, it was something that myself and the other programmer Joseph Roper felt we had to get near perfect straight away. That way we could show the team we can adapt to a new engine easily and that any further tasks won't be too difficult.

Figure 2 Car Movement – see video at 1:03

To ensure that the vehicle's movement was scripted and follows a path this was done within a C# script called Car Script. This script sets a private integer called waypointindex to zero. This variable is then used to calculate what waypoint the vehicle should move to first, the vehicles rotation is changed to face the next waypoint. Within the scripts update function, which is called every frame, a private variable called distance of type float is made. A check is done to see if that variable is less than one. If so the waypoint index variable is increased and the vehicle will then look at the next waypoint, by that point the vehicle moves. The waypoints are set within the vehicle's inspector. An array of game objects is taken and the speed at which the vehicle will move as well. It was done this way to ensure the order of waypoints could be changed very easily for any testers or the designers. It also makes it easier to script the movement for the vehicles and increase or decrease the number of waypoints. Having the speed set within the vehicles inspector was also done for ease of testing. If something needed to be tested but the vehicles were getting in the way by moving or colliding with the anteater their speed could easily be set to zero until the testing was finished.

Stopping Vehicle

The next mechanic that was created for the vehicles was stopping the vehicle with mouse clicks. This was done within Car Script. This mechanic was created using the engines OnMouseDown function (Unity - Scripting API: MonoBehaviour.OnMouseDown(), 2022). The function gets called when the player has pressed the mouse button while over the game objects collider. If this happens a coroutine is started, this coroutine will check if the cars current speed isnt zero. If the condition is met the cars brake animation is played and the current speed is set to the current speed minus one. When the vehicle begins to move again after an initial click, Unity's OnMouseUp function is then used. This function works the same as slowing the vehicle down, but instead will wait for half a millisecond before setting current speed to current speed, plus one. The mechanic was made this way as with coroutines you can set delays and that's ideally what the team had in mind. When the vehicle would be stopped by the players it wouldn't start moving immediately there would be a short wait. A wait was put in place so when the vehicle has stopped the anteater can get across the road without being hit almost immediately.

Figure 3 Vehicle Stopping – see video at 1:13

Basic Prototype

With these very basic mechanics made in the first week or so, a very basic prototype with place holder assets was made to show the client a rough idea of how the game would play before the team got to work on the first playable. This prototype consisted of anteater movement, vehicle movement, slowing the vehicle and a very basic menu. When showing the client this prototype they were very pleased with how the team started planning the game and almost all feedback was positive.

With the vehicle mechanics implemented and Joseph finishing up working the anteaters, the next task that was considered was the implementation of collision between the game objects.

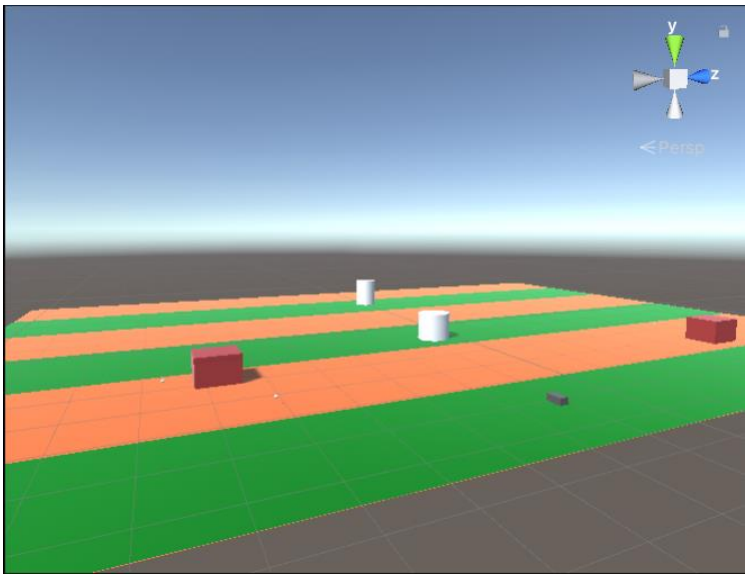

Figure 4 Early Level Prototype

Vehicle Highlight

The team discussed that some sort of feature should be implemented so the players know what to click on. This is when the idea of a highlight around the vehicles came into mind. For this some research was taken and a YouTube tutorial was found with a linked (FREE OUTLINE SHADER in Unity, 2020), free Unity highlight script for download (Nolet, 2022). Rather than trying to make our own the team decided to download this and implement it as it would allow us to test different colours, different widths or whether the outline is highlight or the whole object. The team also decided to use this because with the tutorial it would be a very simple implementation process and it would save time so other tasks could be focused on.

Figure 5 Vehicle Highlight – see video at 1:03

2b. Collisions

Anteater & Anthill Collision

Within the game multiple collisions take place between a combination of different game objects. The first collision task that was assigned was collision between the anteaters and anthills. This was needed as the anthills were the scoring system within the game. Every anteater in the level needs to go to and eat from their anthill and then leave the level. A collision response was made between these two game objects so players would be given a visual change to show they are progressing through the level. The idea that was came up with would be for the anthills to disappear after the anteaters ate from them. Some research went into this task and what was found was that unity game objects can be destroyed upon collision. This meant that the visual progression could be achieved in a much more robust way. Rather than making the game object invisible within the level it could be destroyed and would help with performance and would free up memory.

To implement the collision check and response a C# script was made and assigned to the anteater. Within the script Unity's collision function OnCollisionEnter was used (Technologies, 2022). This function gets called when a collider touches another collider, both anteater and anthills had been given one. Inside the function an if statement is made and is used to check if the object collided with has a specific tag. Objects were given different tags so they could be separated from other game objects. This would stop any errors from occurring like the wrong collision check or response happening. For the anthill collision it checked for the tag "Anthill" if a game object that the anteater collided with had that tag it would start a coroutine and play the sound effect for that collision. Within the coroutine the anteaters speed would be set to zero, so it appears that has stopped to "eat" once done a delay is set for 0.2 seconds then a particle effect is played which was created by one of the other team members. After the effect is played another delay is active but for 0.3 seconds and once it has completed another an if statement checks if the collision game object is not null, it will then call the destroy function and destroy that game object. After the anthill has been destroyed the anteater will begin to move again.

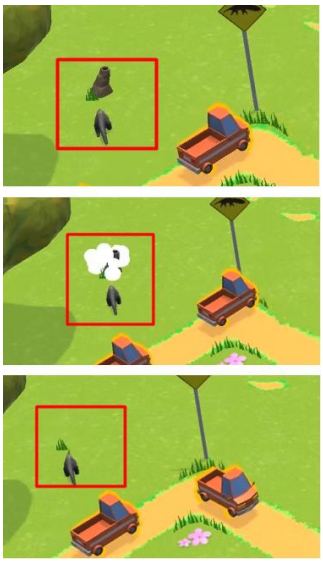

Figure 6 Anteater - Anthill Collision – see video at 1:09

Collision Between Vehicles

Another collision check that had to be implemented was collision between vehicles. Without this, vehicles would drive through each other and would make the game unfair if the player stops a vehicle then another would drive through, and this would ruin the level attempt and the experience for the player. To create collision between different vehicles a script called **Traffic Collision** was created, although further into the semester the code within this script was moved to the **Car Script**. For this collision a function called **TrafficDetect** was created. Within this function a layer mask called mask is created and the layer is set to 8. There was no reason for choosing the layer as 8 others than its one of the last values so if a layer of another object was changed manually in the inspector it wouldn't affect the traffic collision (Technologies, 2022). All the vehicle objects are moved to this layer so when the raycast is created to check for collisions it will only look for the game objects within that layer, overall stopping any errors from occurring.

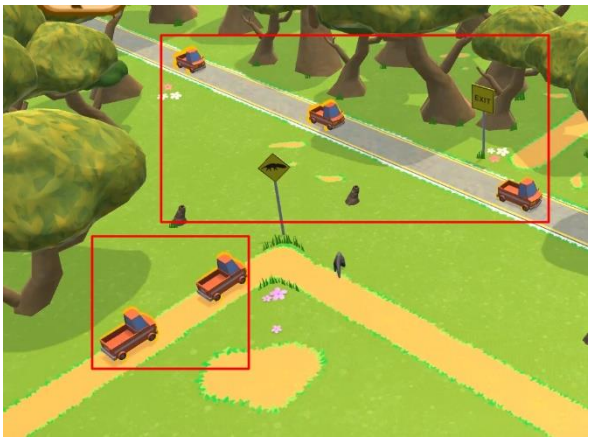

Figure 7 Traffic Collision in action (bottom left) compared to freely moving traffic (top). See video at 1:17

Next a variable for the distance of the ray called dist was made and set to 5. The variable was set to this value after multiple tests, 5 was a reasonable distance between vehicles and would give the anteater a chance to get across the level if there was a build-up of traffic. A raycast hit called hit1 is made, next this was made so the data object gets returned when a ray hits an object during a raycast (Technologies, 2022). A third variable is then created called forward this is used to set the transform direction. This transform is a given direction from local space to world space. With these variables created they could then be used as the Raycast function parameters within the collision check. If a game object within the same layer mask is hit by a ray it will set a Boolean called **inTraffic** to true and will then play the braking animation, implemented by Josh Heron one of the teams' designers. If the object isnt hit by the ray the boolean is false. The boolean variable is checked within the car scripts **update** function. If the vehicle isnt in traffic the vehicles **Patrol** function is called (the function that makes the vehicles move).

Overall, the team were very happy with this feature, and it added some complexity to the game as traffic within the levels had to be managed and players couldn't cheat the game. As mentioned the traffic collision implementation was once in its own script but it was moved and slightly tweaked as little issues were occurring that as a team it was decided as a major bug to be fixed.

2c. UI & Gamestate Programming

Summary

Within the game multiple different game states are used for many different things whether that's teaching the player how to play, an interval between levels or a main menu. The task at hand was to implement a splash screen, a main menu with multiple buttons which lead to a how to play section, a credits section and a level select. The game states within the levels consisted of level 1, level 2, level 3 (which unfortunately had to be cut last minute) as well as a win and lose state and finally a pause menu which Joseph took into his own hands.

Over the course of UI implementation lots of feedback was received from the artists and designers if certain pieces of UI had to be rescaled or repositioned. This helped massively making the process a lot easier.

Original Menu

One of the first states which were implemented was the menu. The level was also another state. Within the first few weeks this menu consisted of two buttons play and quit on a solid blue background. This was done within the very first prototype before the first playable. It was made to show the clients that the game would have an expanded menu which included a how to play section and more. The client was happy with this and had no issues with having a menu and multiple other game states.

Once collision was implemented into the game a lose and win state had to be made. The plan for these states was for them to load when the player has completed the level or lost, this would occur when all the anteaters have made it to the exit or if one of the anteaters gets hit by a moving vehicle.

Figure 8 Placeholder Menu

Lose & Win States

Within these states a message would be displayed to the player telling them “Level Completed” or “You Lose! Try Again?”. The states would also display a random fact about the anteater it would either be a happy fact or a sad fact depending on which state the player is in. Multiple buttons are present within the lose and win state as well. The win state is supposed to show the player’s progression. It does this by having a “continue” button which will allow the player to go to the next level, whereas the lose state has a “retry” button this will allow the player to start the level again and try completing it. The player can do this as many times as they like. The buttons are also accompanied by a menu button which returns the player to the main menu.

The functionality of the win and lose state was done within two scripts. It was done within two scripts as they do have a lot in common, but the buttons require different functionality.

LoseState is the first script, within two functions Retry and MainMenu are made. Both use Unity’s SceneManagement class which gives access to the LoadScene function (Technologies, 2022). The main menu function is very small and calls that function with the parameters “MainMenu”. These parameters are required as that is the name of the scene which is loaded upon the button press. The retry button functionality is created by searching for a game object with the tag “prev scene”. An if statement is made checking that condition and if it’s not null. When the condition is met a variable called “prev” is made. This variable is set to the game object with that tag and gets its component “prevScene”. This variable is set within the PrevScene script which Ross Gauld made. This script will check the current scene by getting the active scenes name using scene manager. It will then set the previous scene to the current one. Back in the lose script once the variable “prev” has been initialised it is used as the parameter within the LoadScene function. if there is no previous scene It will simply load the level select state instead.

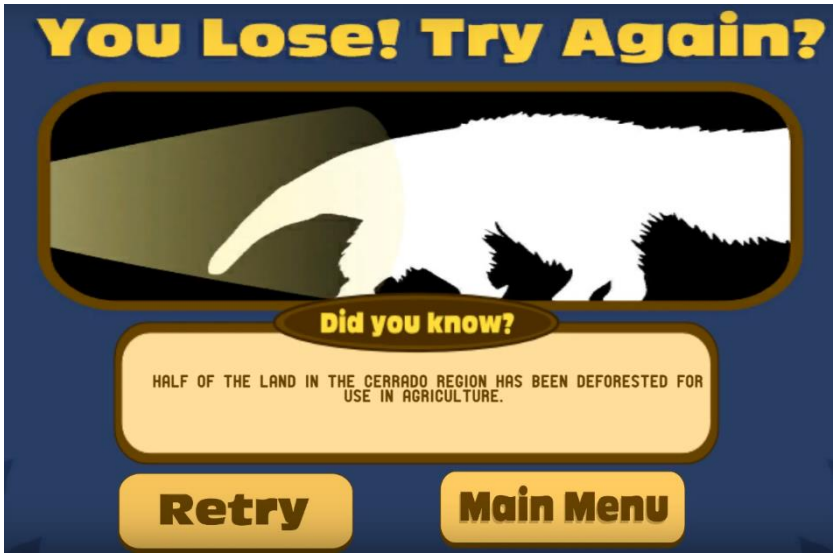

Figure 9 Lose State – see video at 1:28

The WinState script is like the lose state. It has a LoadMenu function and a NextLevel function. This function checks for the same conditions to create the "prev" variable but once created, the variable is used within a switch statement. A switch statement is used since it has better optimized implementation and code execution is much faster over an if statement. This switch statement will check the previous scene that was played and will then load a specific scene when the player presses continue depending on the previous one. There are three cases, although the final release only uses two. Case one checks if the previous scene was "LVL1" its scene manager will load scene "LVL2". If the previous scene was "LVL2" it will load the scene "CreditsScreen". Although originally it would load "LVL3" and then the credits. However, the third level couldn't make the final cut.

The functionality of the buttons is added within the scenes. Both scenes have canvas' which the buttons and images are added to using Unity's pre-existing UI game objects. A game object is made which only has one component. Either the win or lose script. This game object is then drag and dropped into the pre-made OnClick function for the buttons (Technologies, 2022).

Figure 10 Win State – see video at 1:58

Random Fact Display

As mentioned previously the two states will display a random sad or fun fact every time it loads. To make this some research was taken into a similar feature which was random hints/tips displaying in games (Vegas, 2021). The functionality for this was done within a script called RandFact. This script holds one function and two coroutines. Firstly, two variables and a game object are made. An integer called "randValue" which is used to get a random number between 1 and 5. The second variable is a boolean called "winState" which is set to true in the win state, it's used as a check condition. Finally, the game object is made and called "factDisplay". This is used to display the fact in the states. Within the function,

which is named **Start** there are two if statements. The first checks if the boolean which was created is true, if so the **WinFactTrack** coroutine is started. The other checks if the boolean is false. If that condition is met the coroutine **LoseFactTrack** is started. These two coroutines set the "randValue" to a number between one and five using Unitys **Random.Range** function (Technologies, 2022). This returns a random float within the parameters, which in this case is one and six (there is five facts for each). The variable is then used within a switch statement and depending on which number is returned a specific piece of text is displayed. Two coroutines are made since there are different facts for different states.

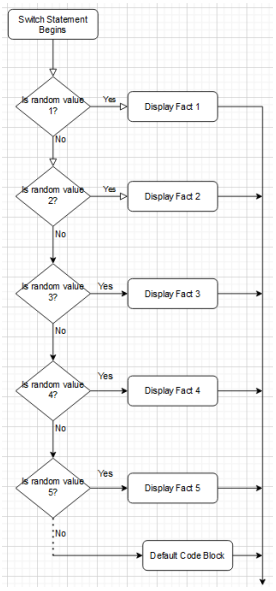

Figure 11 Random fact switch statement flowchart - see video at 2:31 for demonstration

Splash Screen

When the game is loaded up a splash screen is the first thing the player will see. This is a short and simple state that just displays the team's logo, the university logo and the client's logo. This was implemented to credit the university and the clients and its something that many games use nowadays and a lot of the games that the team drew inspiration from used them. It was also implemented so the game doesn't jump to the menu almost immediately. The feedback from the clients were positive and double checked that they would be

referenced within this state, which the team reiterated that they would be as well as the university.

Figure 12 Splash Screen - see video at 0:01

This state is made in its own scene. Its very small and simple and just has a game object with a video render component which is set to the splash screen video (Playing Videos in Unity, 2018). The only required functionality for this state is when to load the menu. A script is made for the splash scene. It uses two functions **Awake** and **CheckOver**. The awake function will play an instance of Unity's **VideoPlayer** class called "video" which is created at the start of the script (Technologies, 2022). The video is played on awake then the videos variable "loopPointReached" is set and added to CheckOver. This function takes an instance of video player as a parameter. If the loop point of the video is reached (the end of the video) the menu is loaded.

Final Menu

As mentioned previously the first menu was just a placeholder within the very first prototype. The final menu contains all of artist, Alisha Qurban's final UI art. The menu consists of a title and background image as well as three buttons: play, how to play and credits. The functionality for these buttons is within the **MainMenu** C# Script. Within the script there are five functions and two coroutines. The first function is **Start**. This function is called at the start of the runtime the only thing this function does is set the level select game object to false. This means the level select state (canvas) won't be rendered until it's true.

Figure 13 Final Menu Implemented - see video at 0:03

The level select is within the same scene as the menu. This is because they have the same music source, so making another scene would be costly. The next function is for the play button, and it was called PlayGame. This function does the opposite of start and instead sets the level select active to true, which will render the level select state instead of the menu. There is also a LevelSelectReturn which when in the level select will return the player to the menu.

The two other functions for the main menu are LoadHTP and CreditLoader. These two functions both run a coroutine each set a one second delay and loads the credits or the how to play scene. There was a delay added so the states wouldn't load immediately and the sound effect for their buttons could be heard. For a while there was a minor bug where the sound effects for those two buttons wouldn't play. The functions are then set to the buttons by adding the script to a game object then dragging and dropping that object into the OnClick function in the button's inspectors.

Level Select

As mentioned in the previous section the level select is within the same scene as the menu. This is because the states are small and use the same music source so it would save making another scene which would use more memory. The level select scene is where the player will choose what level to play. In the final build there are two levels the player can click on and an additional three other level buttons which are "locked". Bringing the total level count to five if the project was to be expanded upon. The functionality for these buttons is made in a script called LevelSelect. This script is relatively small but was created if there was to be lots of levels within the game. The functions within this script use the scene manager to load the correct level depending on which button was pressed. If the level one button was pressed the scene called "LVL1" will load.

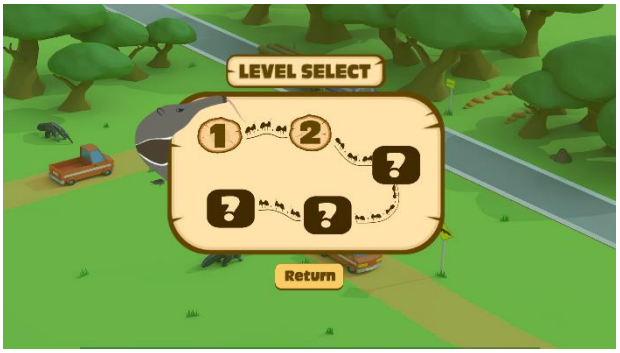

Figure 14 Level Select Implemented - see video at 0:57

How To Play & Credits

A how to play section was added to the game to teach the player how to the play game before going into a level. The credits section on the other hand was made to restate that the game was a demo and there would be more to come hence the “To be continued” at the top of the credits. The credits were also included to credit everyone who worked on the project and include the university and the clients that the team worked with over the semester. To implement these states, they both had their own scenes.

Figure 15 How to Play State - see video at 0:06

Figure 16 Credits State - see video at 2:35

Both states have a game object which has a video render component that the scenes camera is positioned to, this is done so the only thing the player can see within this scene is the video and buttons and they won’t see anything in the background. Within these states there is also a return button which will take the player back the menu.

The team and clients were both very happy with how not only these game states turned out but the menu, level select, win, and lose as well. Apart from a few updates to the UI and the scaling and positioning of some of the 2D art it was a very straightforward process with the help of the artist’s feedback.

2d. Audio Implementation

The game has a variety of different sound effects and music for different states. The music and sounds were made by the producer Ross. Multiple tasks were given over the course of the semester to implement the audio for the game. Audio would be played through a lot of different conditions whether that’s collision, a button press or changing game state. Some research was taken so the task could be a simple process.

Within Unity a game object can only have one audio source component (Sound, 2020) Because of this and the size of the game it was decided that where audio was required an empty game object would be made and the only component it would get would be one audio source.

Music

For the music in the game states a game object would be made and the audio source component would be added using the correct music file for as the source. Within the states: menu, levels, win, lose and credits all have their own different music. However, it is very easy to implement. Once these game objects are made and the source is added there is a check box called “Play on Awake”. This will start the audio source as soon as the game object is enabled, this would be when the scene is loaded. It was done this way as there is different many types of music but there is only one in the states that play music (Technologies, 2022).

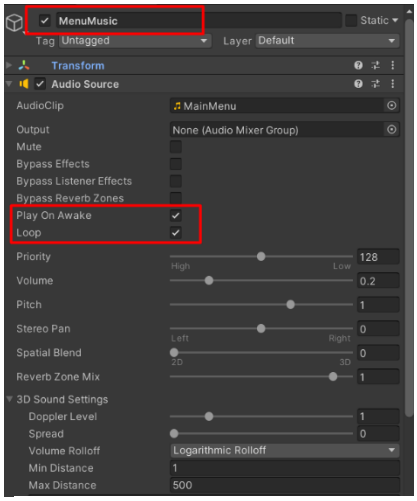

Figure 17 Menu Music object with audio source set to play on awake. See video for music.

Sound Effects - Buttons

Sound effects within the game are implemented in a similar way but the sounds aren’t played on awake and instead played when a condition is met. Once again an empty game object is made, and an audio source is added. For the buttons within the different states the game object with the audio source is then dragged and dropped in to the **OnClick** function within the button’s inspector and **AudioSource.Play** is picked from the dropdown menu (Technologies, 2022).

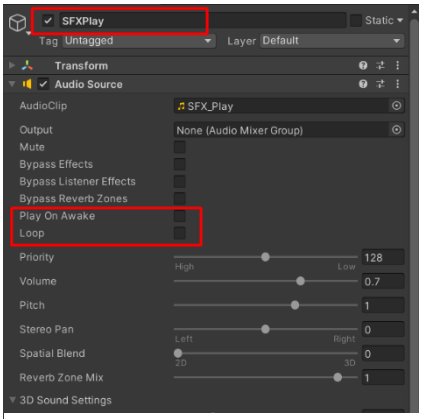

Figure 18 MenuButton Audio - see video at 0:04 for button audio

Sound Effects - Collision & Vehicles

Whenever there is a collision within the levels, or the vehicles are clicked on by the player a sound effect is played. To do this an audio source is added to the prefabs which require sound effects. This included all three of the vehicles and the anteater prefab.

The vehicles sound effects are added into the CarScript. A public audio source is made and then set to play within the OnMouseDown function (Technologies, 2022). Within the inspector the audio source within that script is then set to the corresponding sound effect for the car for example SFX_Horn1 or SFX_Horn2.

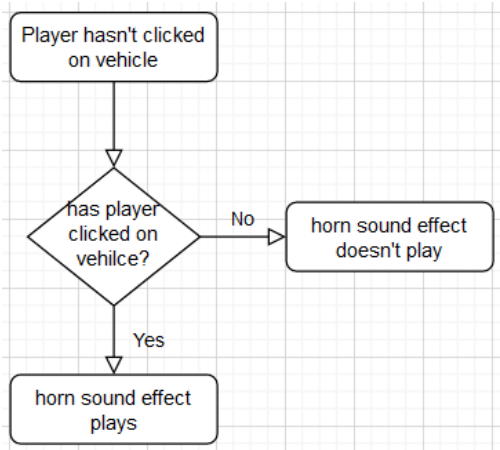

Figure 19 Vehicle click sound effect flowchart - see video at 1:04

For the anteater a slightly different method is used. As mentioned only one audio source can be added to a game object this is done. However, the anteater plays three sound effects. To play the other two, public audio clips are made in the anteater collision script as well as the audio source. Audio clips are referenced and used by audio sources to play different sounds, so this method was used to work around only being able to have one audio source per game object. Within the anteater's collision script, the audio clips or audio source is played depending on the collision. For example, if the anteater collides with an anthill the sound effect for eating is played, after it ate at the anthill the audio source is played which is a short chime sound to let the player know an anteater has eaten at an anthill.

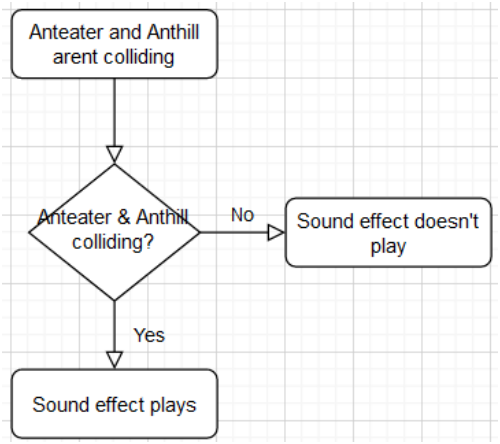

Figure 20 Flowchart for anteater & anthill collision sound effect - see video at 1:12

Overall, the team were happy with this, and the feedback was all positive. A few changes were made but that was simply changing the audio to the final version, making it loop or changing the volume of certain sound effects.

2e. Other Work

Over the course of the semester there was many extra tasks taken to work on the game or help make development easier for the team.

Prefabs

Prefabs of each main game object were made to help implementation and level design quicker and easier for not only the programmers but the designers on the team as well. Prefabs is a system in Unity which allows users to create, configure and store any game object complete with all its components, values and any child game objects as a reusable asset. Essentially a prefab is like a template which you can create instances of within a scene. These were used since all the levels use more than one instance of an object like vehicles, waypoints, anteaters, and anthills. Although not having real performance benefits prefabs were used as a convenience and save so much time (The Benefits of Prefabs - Unity Game Development Tutorial, 2020), this gave the team more time to focus on other tasks for the game (performance benefits of using prefabs, 2013).

Prefabs were worked on by me and Joseph. This consisted of adding all the existing components like scripts, audio sources, colliders, and shaders. If anything, new was made the prefab would be updated.

Technical Design Document

The TDD was one of the documents which had to be submitted alongside the prototype of the game.

Within the very first week a coding style was decided, and some naming conventions were made. This was made so anyone who works on any of the scripts, or the projects can find things with ease using the naming conventions and the code is laid out the same throughout.

Camel case was used as its one of the most common coding styles and it was one me and Joseph were used to (What is CamelCase? - Definition from WhatIs.com, n.d.). Overall, the coding style was followed somewhat throughout however, there are many cases where it has not been followed.

Other areas of the TDD which I worked on consisted of were technical goals, system requirements, game engine choice, some gameplay mechanics and ui mechanics, certain prefabs, tech diagram, scene details, scene management, use of physics engine and delivery platform.

Development Diary

Below is a development diary separated into weekly sprints which shows the tasks and progress completed as the semester went on. More detail on why and how is given on the tasks within the other chapters of the portfolio.

3a. Pitch Presentation

Sprint 1 (Pitch Presentation) – 01/02/22 – 09/02/22

Week 1

- Researched games, target audience and target platform
- Started coding style & naming conventions.
    - Decided on camel case and created document found in the TDD
- Looked at tutorials for Unity and documentation since it was first time using it
- Practiced unity
    - Made a simple patrol movement mechanic
- Set up GitHub with Joe for the team

Week 2

- Started working on basic car movement with patrol mechanic
- Created a basic scene as a basic demonstration for the clients
- Worked on a simple stop car with mouse click mechanic
- Continued setting up GitHub

- Wednesday – No work that day, attended funeral.

Sprint 2 – 15/02/22 – 23/02/22

Week 3

- Sorted GitHub issues
- Started working on collisions and game states in prototype
- Spoke with Josh about a traffic system, rev bar one car has stopped
- Continued working on prototype/first playable in a 3d scene
- Made a simple play and quit button in a placeholder menu

Week 4

- made short level for prototype
- Added Alana's highlight script to prototype
- Created short video of demo

3b. First Playable

Sprint 3 (First Playable Submission) – 01/03/22 – 09/03/22

Week 5

- Added red truck model made by Daniel
- Tidy up level and mechanics for FP
- Traffic collision

Week 6

- Finished first playable
    - Added more game objects (cars, anteaters, anthills, and tree border)
- Started working on Tech Design Doc
- Added concept ui, buttons and audio
- Made first playable video and the build

Sprint 4 – 14/03/22 – 23/03/22

Week 7

- Started fixing traffic collision and car highlight bugs
- Updated prefabs

Week 8

- Implemented Daniels shaders on prefabs (metallics, normal & occlusion)
- Started working on a longer level
- Added concept sfx
- Worked on lose & win states

3c. Alpha Submission

Sprint 5 (Alpha Submission) – 29/03/22 – 20/04/22

Week 9

- Integrated random fact system
- Added new sfx for states
- Working on prefabs audio
- Working on Alpha build with first & second level
- Made second level prototype
- Added Ali's new ui on menu
- Added anteater sfx

Week 10

- Implemented Alis final ui for menu, lose, win & level select states
    - Added credits button
    - Added level select buttons
- Made alpha submission build.

3d. Beta Milestone

Sprint 6 (Beta Milestone) – 26/04/22 – 04/05/22

Week 11

- Made splash screen, credits and how to play states
- Fixed flying car bug
- Implemented state SFX
- Added win, lose themes and final level music/sfx
- Added button SFX

Week 12

- Added car sfx
- Added Menu button delay
- Sorted functionality for retry and continue button
- Added credits music

3e. Release

Sprint 7 (Release) – 10/05/22 – 17/05/22

Week 13 - 14

- Remaining bugs fixed
- Remaining assets added
- Prioritised Tech Design Doc

Reflection

4a. Does The Project Meet Requirements?

Overall, I feel Anteater Highway met the client's brief. The clients had an idea around a frogger style game. The games we drew inspiration from like Crossy Road is exactly that, a frogger type game. The brief mentioned other ideas like a Fruit Ninja style mechanic to open mounds or anthills, myself and the team did come up with ideas for this, but they were considered as additional features and something that would be out of the main scope of the game.

4b. Was Feedback Implemented?

Throughout the semester the team held multiple meetings with the clients to update them on progress, show changes we made and brought up new ideas to discuss. For the games mechanics which myself and Joseph programmed the clients were very happy, but it did take some time to get there. As mentioned in an earlier section, the original idea was to move the anteaters to avoid oncoming traffic. However, after speaking to our tutor Robin, she told us to rethink our tone as what we originally thought of could be deemed insensitive, which is when I had a lightbulb moment about controlling the cars, preserve rather than survive. From that point on myself and the team as well really thought carefully and tried to stick to the theme of preservation when coming up with new ideas or implementing the mechanics.

4c. Strong Areas of The Project

The final build of the game which was submitted to the clients I feel was strong in a variety of different areas, the basic mechanics of the game were close to perfect giving players a smooth playing experience. The game was polished and didn't contain any major bugs. The clients wanted an educational game for a younger audience, and I think we done just that. While only containing two levels the variety of different facts players can learn gives players

incentive to replay the game. All the art and audio which were implemented into the game either by myself or Joseph are all final versions, no original concepts are within the game at all because of this I think the game looks great and as a team we managed to get the aesthetic just right, this was important as the clients didn't want a very cartoony game, but we also couldn't make a game about a sensitive topic for children super realistic. So as a team I felt we got the aesthetic and the mechanics near perfect. Finally, although only two levels I think they are fun and repayable. Robins main feedback over the semester was making it fun. I think that was captured with the different level speed options as well myself and others constantly testing and balancing the levels.

4d. What Could Have Gone Better?

There are a few things that could've have gone better, firstly I feel if we chose Fork as our source control from the beginning rather than GitHub Desktop this would have saved a lot of time as Fork allowed us to resolve merge conflicts which was a necessary feature we needed as Unity uses binary files for their scenes and prefabs. Furthermore, the Jira could have been managed better with better descriptions of tasks. Finally, after losing two team members which although is out of our control I think the organisation could've have been better as I felt I was doing more tasks outside my role than in my role. For example, level design and asset integration. I also think the scope, or the focus of the project should have shifted at this point. With less time being focused on what I believe to be additional features such as extra audio and a death animation instead of focusing on bug fixing and level implementation. With this being the case, I think the playtime of the game could have been increased and having more levels would add to the user experience.

Currently I believe the game is slightly too short for players to get a real grasp of how the game should feel and play and its rather unfortunate the third level had to be cut.

Expanding on the point of better organisation and having to do tasks out with our roles, I found myself on multiple occasions scrambling to finish a build the day of or before submission. This at times became very frustrating and stressful for me, as the focus of the designers was on expanding the scope rather than implementing the existing features of the scope. Which left me to do those builds with not much time, which took away from other work I could have been doing.

4e. How Did This Project Relate to Existing Projects?

This being the first project I worked on with a team, I'd say it doesn't relate to any of existing projects I've worked on. However, adapting to a new engine like Unity isn't new to me as throughout my experience so far at university I've had to adapt to a variety of different libraries and engines like Unreal Engine or the gef library.

4f. Developing the Game Further

The team discussed over the course of the project additional features and things we considered out of the games scope. Having the opportunity to work on this project again within the future, I would consider many of these features. The first features obviously being more levels and more facts to expand on the educational side of the game. When expanding on the levels the team had ideas of implementing night-time levels where lighting and visibility is low increasing difficulty for players. The team also had an idea around a definitive ending for the game which I'd hope to work on. We wanted the game to tell a very subtle story through each level, showing urbanisation of the land around. The team discussed an idea around a final level which would be impossible to play, the level would zoom out and show a nature bridge off to the side which the Anteater would walk across, showing an end to Anteater Highway. This idea was brought up to the client and our tutor Robin and they all loved this idea. One final thought if this project was to be taken on further would be to reinforce the coding style and naming conventions as it took some time to create and was only followed most of the time and there were times where me and the rest of the team forgot to stick by it.

References

2013. performance benefits of using prefabs. [Blog] Available at: <https://answers.unity.com/questions/523037/performance-benefits-of-using-prefabs.html> [Accessed 20 May 2022].

2018. *Playing Videos in Unity*. [video] Available at: <https://www.youtube.com/watch?v=HUJ4_vFkyMc> [Accessed 20 May 2022].

2020. *FREE OUTLINE SHADER in Unity*. [video] Available at: <https://www.youtube.com/watch?v=hEth0drTuyg&t=108s> [Accessed 20 May 2022].

Coolmathgames.com. n.d. *Cool Math Games - Free Online Math Games, Cool Puzzles, and More*. [online] Available at: <https://www.coolmathgames.com/> [Accessed 20 May 2022].

Crossy Road - Endless Arcade Hopper Game. n.d. *Crossy Road - Endless Arcade Hopper Game*. [online] Available at: <https://www.crossyroad.com/> [Accessed 20 May 2022].

Docs.unity3d.com. 2022. *Unity - Scripting API: MonoBehaviour.OnMouseDown()*. [online] Available at: <https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnMouseDown.html> [Accessed 20 May 2022].

Friv.com. n.d. *Friv : Free Games!*. [online] Available at: <https://www.friv.com/> [Accessed 20 May 2022].

Ketra-games.com. 2020. *The Benefits of Prefabs - Unity Game Development Tutorial*. [online] Available at: <https://www.ketra-games.com/2020/07/unity-game-tutorial-benefits-of-prefabs.html> [Accessed 20 May 2022].

Learning.rzss.org.uk. n.d. *Discovery and Learning*. [online] Available at: <https://learning.rzss.org.uk/> [Accessed 20 May 2022].

Ninjakiwi.com. n.d. *Ninja Kiwi - Free Online Games, Mobile Games & Tower Defense Games*. [online] Available at: <https://ninjakiwi.com/> [Accessed 20 May 2022].

Nolet, C., 2022. *Quick Outline*. [online] Unity Asset Store. Available at: <https://assetstore.unity.com/packages/tools/particles-effects/quick-outline-115488> [Accessed 20 May 2022].

Sound, C., 2020. *Multiple audio sources on a gameobject in Unity. Cujo Sound*. [video] Available at: <https://www.youtube.com/watch?v=jFjg2iwuF1s> [Accessed 20 May 2022].

Studio, K., 2020. *Super Easy Patrolling AI | Unity Tutorial*. [video] Available at: <https://www.youtube.com/watch?v=22PZJlpDkPE> [Accessed 20 May 2022].

Technologies, U., 2022. *Unity - Manual: Layers*. [online] Docs.unity3d.com. Available at: <https://docs.unity3d.com/Manual/Layers.html> [Accessed 20 May 2022].

Technologies, U., 2022. *Unity - Manual: Prefabs*. [online] Docs.unity3d.com. Available at: <https://docs.unity3d.com/Manual/Prefabs.html> [Accessed 20 May 2022].

Technologies, U., 2022. *Unity - Scripting API: AudioSource.Play*. [online] Docs.unity3d.com. Available at: <https://docs.unity3d.com/ScriptReference/AudioSource.Play.html> [Accessed 20 May 2022].

Technologies, U., 2022. *Unity - Scripting API: AudioSource.playOnAwake*. [online] Docs.unity3d.com. Available at: <https://docs.unity3d.com/ScriptReference/AudioSource-playOnAwake.html> [Accessed 20 May 2022].

Technologies, U., 2022. *Unity - Scripting API: Collider.OnCollisionEnter(Collision)*. [online] Docs.unity3d.com. Available at: <https://docs.unity3d.com/ScriptReference/Collider.OnCollisionEnter.html> [Accessed 20 May 2022].

Technologies, U., 2022. *Unity - Scripting API: Physics.Raycast*. [online] Docs.unity3d.com. Available at: <https://docs.unity3d.com/ScriptReference/Physics.Raycast.html> [Accessed 20 May 2022].

Technologies, U., 2022. *Unity - Scripting API: Random.Range*. [online] Docs.unity3d.com. Available at: <https://docs.unity3d.com/ScriptReference/Random.Range.html> [Accessed 20 May 2022].

Technologies, U., 2022. *Unity - Scripting API: RaycastHit*. [online] Docs.unity3d.com. Available at: <https://docs.unity3d.com/ScriptReference/RaycastHit.html> [Accessed 20 May 2022].

Technologies, U., 2022. *Unity - Scripting API: VideoPlayer*. [online] Docs.unity3d.com. Available at: <https://docs.unity3d.com/ScriptReference/Video.VideoPlayer.html> [Accessed 20 May 2022].

Vegas, J., 2021. *HOW TO MAKE RANDOM HINTS APPEAR ON A LOADING SCREEN - MINI UNITY TUTORIAL WITH C#*. [video] Available at: <https://www.youtube.com/watch?v=QMRZepOcv5E> [Accessed 20 May 2022].

WhatIs.com. n.d. *What is CamelCase? - Definition from WhatIs.com*. [online] Available at: <https://www.techtarget.com/whatis/definition/CamelCase> [Accessed 20 May 2022].