Introduction to Game Programming

Magazon

Helena Tevar Hernandez (ht222fd)

1dv437 – 19 Summer

Introduction

The game Magazon was made with Unity 2019.5. The game would be defined as a a 3D arcade, following the idea from the Atari game Paperboy. In Magazon, the player must deliver parcels in two ways, a safer way (or Robust way) and a riskier but faster (Fragile way). Robust has less probability to break the parcel, gives more points to the score but takes longer time. Fragile has a big chance to break the parcel, gives less points, it adds the difficulty of aiming but is faster. The game ends successfully when all parcels are delivered or unsuccessfully when the player run out of parcels/time.

Architecture

The original idea was to follow a layered architecture to avoid cycles, but such layers follow closely the MVC pattern. In Magazon there are three layers: UI – Level Controllers– Mechanics. The components of each layer can only pass messages to other components inside their layer or the layer above.

User Interface	Sound controller User Interface (Layout)
Level Controllers	Level Controller Persistence
Mechanics	Player mechanics NPC mechanics Colliders etc

Level Controller manage the beginning and ending conditions for the levels and access the persistence, in this case by using PlayerPrefs, to save the score of each level.

Other shell scenes, as the start shell and ending shell are managed by one script that access the persistence to show the scores.

Design Patterns and Data Structures

In order to ensure that persistence was not modified incorrectly I used a singleton pattern for the persistence script, so it manages the scores safely.

The game is simple enough to have an Ad hoc pattern, tested to avoid cyclicity but at the same time trying to keep it abstract in case of possible reusability and scalability. For instance, the fragile parcel has the basic mechanics for the parcels to arrive to a destiny safely, broken or disappearing if they get lost. Robust parcel inherits from the fragile their basic mechanics and adds few changes to its

mechanics. In case of wanting to create new parcels, I could use inheritance to keep adding more. This same could be applied to the objectives, the basic mechanics are implemented and could be inherit to more complex objectives.

The data structures I decided to use where Arrays for the number of levels the game has because they would be static, and I did not expect to change (add or delete) levels to the game. For the rest, I used Lists because of their flexibility. The script that manages the movement of the player's van and the NPC cars use a List for the number of axles the cars have. It is like this so the game can be scaled to bigger vehicles with multiple axles. Another List used is the list of nodes that the NPC cars follow to move around the map. Some maps required longer paths with more nodes because of the complexity of the map, so I needed a data structure that was flexible depending on the situation.

Collisions and Geometry

The more relevant mechanic that used collisions and geometry was what I called the van's radar. The mechanic to deliver a robust (safer) parcel was: Create a collision box at one side of the van. When an objective collides with the "radar", the radar will save its transform. The robust parcel will access the radar and move towards the objective. I decided this mechanics because I wanted to know the closest objective within a range from the player, specially in their right side[1].

Other collisions are also handled in the game, but they are not main mechanics but fun additions. The NPC's have a collision area and when the player enters, they make their claxon sound. Some of the props used have a collision area that triggers a sound of scratching when the player crashes and moves them.

Textures, Shaders, materials and lighting

No special textures, shaders or materials because they were no needed in this kind of game. I used commissioned 3d models for the game, but I decided the colours and cartoony style. This was a personal decision because of a vision problem I have. To me to be able to see the colours, they must be saturated and so cartoony style fits better. I did use a skybox from Unity Store to give it more cartoon feeling.

Animations

The NPC cars are animated, so they move around a path of nodes. Each node has a transform point. The cars will adapt the steering angle of their motor axle to the transform position of the next node they have to reach. When the car is close to the node, it will reach the next one. Multiple cars can follow the same path.

The parcels do move but they act differently depending which kind of parcel you want to deliver. The fragile parcel is throwed from the spawn point located in the van at a determined speed towards the vector direction of the spawn point. This was made so the player had to aim, which is difficult because the spawn point is located at the right side of the van.

The robust parcel moves each tick towards the direction of the objective, that was located by the van's radar, if there is one. It takes 2 seconds to the parcel to spawn, it was made this way first to discourage this kind of deliver and second, because the reaction time between the radar and the parcel was prone to errors if the player was driving too fast. In case of no objective, the parcel will follow the fragile delivery mechanic and get lost.

[1] To make the game more complex, I decided that the player can only deliver to objectives at their right, kind like Posten does with their vans, that only open to their right side.