Design Document

Concept

The concept of the game is that you play as a rat trapped in a garden, trying to escape. You must dodge and hide from the Owner while scratching your way through a weak fence panel. Since the human is faster, you'll need to terrorize garden interactables to distract him, giving you a chance to slip away, avoid being kicked or stomped, and make your escape.

Mechanics

The interactables in the scene are designed to be reusable and modular. Each follows a simple structure consisting of a "before" (clean) state and an "after" (broken) state. When the player interacts with an object, it triggers an input prompt and swaps between these states.

This system is adaptable across different objects, including the escape fence. It is implemented using switch cases and tags each interactable is manually assigned a specific tag, with its corresponding models set as child objects.

Additionally, interactables can be reset, which is the method used by the Owner when repairing items in the garden.

The Owner (enemy AI) uses a NavMesh for movement and pathfinding to chase the player. When the player enters the Owner's field of view, a chase begins. However, if the player breaks an object, the Owner will prioritise repairing it, temporarily abandoning the chase unless the player is spotted again. This behaviour is managed using a FIFO queue to organise tasks.

The player uses an input system sourced from the Unity Asset Store and can interact with objects to break them while avoiding detection.

The escape fence acts as the win condition. The player must damage it enough to escape while avoiding the Owner. If the Owner catches and stomps the player too many times, the player loses and must restart the game.

Use of materials, particles, animation, terrain

TThe materials used in this project were sourced from the Unity Asset Store and other online resources. For some of the worn or broken assets, I applied normal and roughness maps to improve visual detail, for example, making the fertilizer appear more realistic. Textures were also applied to the rat model and the surrounding backyard structures to enhance overall visual quality.

Particle effects were used to add feedback and atmosphere, such as dust being kicked up during movement and water spraying during certain interactions.

The animations for both the owner and the rat were primarily sourced online; however, I created a custom run animation for the rat to fill in a missing movement state.

The backyard environment was built using Unity's Terrain tool. I shaped the terrain with slight elevation changes to avoid a flat appearance and painted it with different textures for variation. Instead of relying on billboarding for grass and trees, I used the tree tool with mesh-based assets, placing them sparsely to achieve a more subtle and natural-looking environment.

Challenges Encountered

For the owner character, I used a NavMesh to restrict movement to a defined area and allow it to chase the rat more effectively. I also implemented NavMesh obstacles to prevent the owner from walking into interactable objects. I had some prior experience with NavMesh from an earlier module, so when it was covered again in class, it reinforced my understanding of the system and how to apply it more effectively.

The rat model I sourced online was already rigged, which was very helpful, and it included some basic animations. However, it didn't have a run animation, so that had to remain unimplemented for a period of time. To resolve this, I created a run animation manually by using keyframes and adjusting the model's bones directly. I also attempted to use a placeholder avatar to retarget an existing animation (a pug running), but this was unsuccessful. If I were to revisit this project, I would like to attempt this approach again.

Because of the theme of this CA being "Miniature Mayhem," many of the assets imported into the scene were either extremely small or not properly anchored. While I could have adjusted the scale earlier in development, and would do so if repeating the project, time constraints meant I worked with what I had and manually scaled objects as needed to fit the scene.

The buttons in each scene weren't working correctly with my GameManager script, so I created a separate script and object for each scene. This script acts as an intermediary and calls methods in the GameManager. Since the GameManager is a singleton and only exists once rather than in every scene, this approach ensures the buttons can still interact with it properly.