Alpaka Recruitment Case

Yoldaş Ulaş TALAŞ

Approach:

After analyzing the provided gameplay video, I set the following objectives:

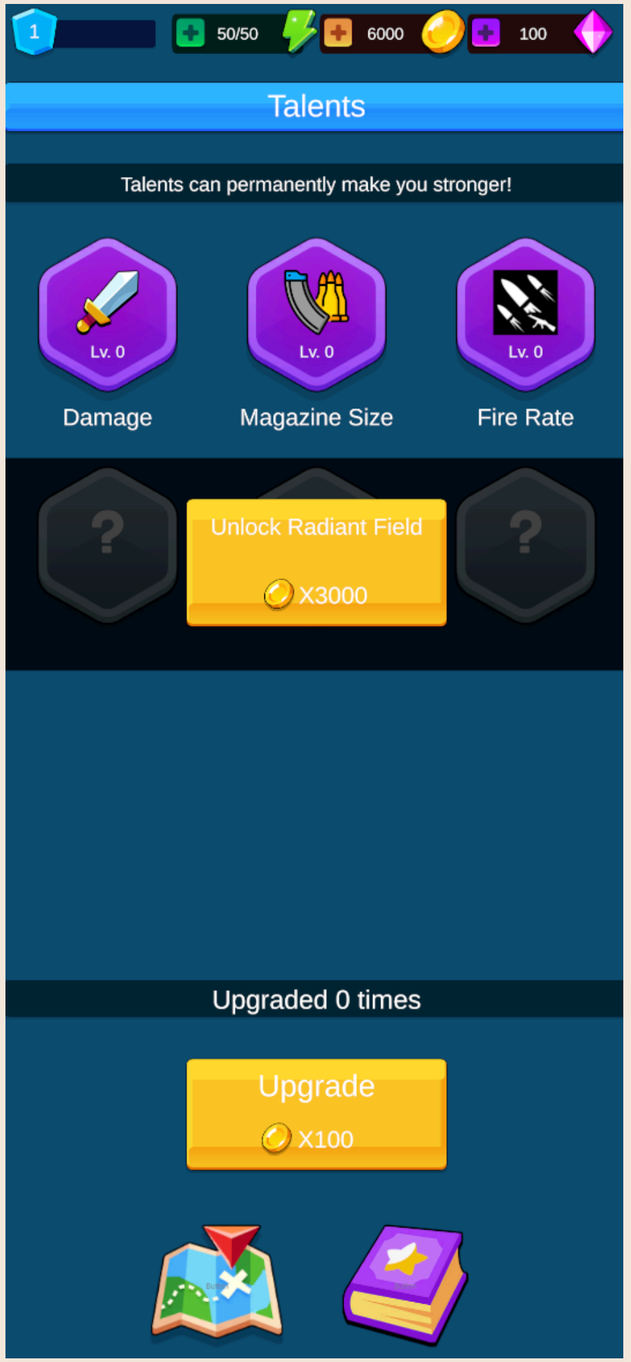
- Player Movement Logic and Animations: Implement fluid player movement with responsive animations.
- Enemy AI and Animations: Develop AI that simulates zombie behavior with proper animations.
- Weapon Effects and Animations: Include particle effects for weapons and a reload animation.
- Main Menu UI with Persistent Data: Create a menu for navigation and saving/loading player progress.
- Environment and Lighting: Design a simple, clear environment for gameplay.

Asset Selection

- To streamline the development process, I first identified and gathered all necessary assets. This approach allowed me to focus entirely on the game logic once development started, avoiding back-and-forth searching for assets.

Development Process

1. Player Movement and Animation:
   - I began by implementing player movement using Unity's Blend Trees for smooth transitions between animations. I also set up a rig system to blend the gun movement with the player's body animations, ensuring the player's weapon movements were realistic during gameplay.
2. Enemy AI and Animation:
   - For the enemies, I imported animations from Mixamo and used a simple state machine to control enemy behavior, including idle, chase, and attack states. After some fine-tuning, I achieved a gameplay flow that felt responsive and challenging, staying true to the original game's feel.



3. Weapon Particle Effects and Reload Animation

- I integrated particle effects for shooting and implemented a reload animation that smoothly transitions with the player's movement. This added a level of polish to the gameplay, enhancing the overall feel.

4. UI and Data Binding

- I implemented a clean Main Menu UI with persistent data storage, using JSON serialization for saving player progress and weapon attributes. I designed the weapon attributes page and added a scroll view to allow players to unlock additional skills. The focus was on both visual appeal and ensuring that no errors would arise during gameplay.

5. Environment and Lighting

- To keep things simple early on, I created a basic plane as a placeholder for the environment. Once the core mechanics were functional, I revisited the environment and implemented lighting to ensure clear visibility and atmosphere during gameplay.

Key Implementation Details And Notes

Enemy AI and Movement

- For enemy movement, I opted to use NavMesh strictly for local avoidance, rather than full pathfinding. To replicate the fast-paced, swarming behavior seen in Zombie Waves, I moved the enemies directly toward the player using a MoveForward approach. This provided the closest gameplay feel within the project's time constraints.
- While this method worked for the case, I believe a more optimal solution would involve a multithreaded local avoidance system. This could be combined with baking non-walkable areas into the avoidance system to eliminate unnecessary collision checks. Also for enemy and player damage detection, I would shift to transform-based checks to further streamline performance.
- During development, I tested a local avoidance system, but encountered challenges with obstacles like cars and fences, which disrupted the flow of the AI. These objects made existing local avoidance solutions less effective without further adjustments.

Scene Management and Initialization

The game's scene management starts with an Initialization Scene, which contains persistent game objects such as the Save Manager, Scene Controller, and Audio Manager. These objects are essential across all scenes, and to prevent execution order issues, they are initialized in this scene. Once the Initialization Scene is complete, the Scene Controller loads the Main Menu scene, where the UI pulls player data from the Save Manager to initialize the interface.

From the Main Menu, the Play button loads the Gameplay scene. My original plan was to reuse the Gameplay scene across multiple levels by dynamically loading enemies, the player, and environmental elements based on the specific level and character choice. This approach would have made the scene more flexible and efficient for handling different levels without needing separate scenes for each one.

Addressables and Memory Management

Towards the end, I began working on Unity Addressables to improve memory management by dynamically loading assets only when needed. The idea was to optimize resource usage by loading assets like enemies, environments, and characters on a per-level basis. However, given that the full scale of the game was not yet clear, I refrained from making too many assumptions about future performance needs.

The intent was to implement a system that allows for reusability of the core Gameplay scene, but since the time limit required focusing on core functionality, I prioritized ensuring that the groundwork was laid and made the Gameplay Scene load as addressable .