CPSC 386-01 - Unity Assignment 4

Brite Bartnek, Wayne Muse
https://github.com/WayneMuse/cpsc-386-assignment-1.2
12/4/25

Introduction

Our Unity assignment is a top-down zombie shooter where the player must survive wave after wave of zombies in a constantly changing environment, all while attempting to achieve the highest score. This project is a continuation of Wayne's Unity Assignment 1. The player spawns in with 20 bullets and must be careful with every shot, as zombies only have a chance of dropping bullets. Each round adds more and more zombies, and each round is only complete when all zombies are eliminated, or the player is eventually taken down by the zombie horde.

Scripts

Player.gd

The Player.gd script is responsible for handling player movement and rotating the player model based on the direction of the cursor. I used part of the code from the AI explorations assignment, where I created a similar top-down shooter. Instead of using projectiles to check if they hit a zombie to do damage, the script employs a simple raycast-based shooting logic and checks if the zombie is within the raycast to kill them. The player script also handles the player's UI and updates the player's remaining bullet amount, the number of zombies remaining, and the number of zombies killed. Throughout the game, the player can obtain additional bullets from dead zombies. Player.gd also handles the player's ammo usage and pickup. Finally, the player script also handles the player's death. It displays the 'You Died' UI when the player dies; they cease to move and shoot, and are given the option to return to the main menu or exit the game.

Added to Player.gd from project 1 is player animation and player sprinting. Player sprinting allows the player to traverse the larger map and works when the player presses the sprint key (default: left shift) and multiplies their speed. The player has two animations, one for standing still and one for moving. If the player has a velocity of 0, the idle animation is played. If the player is moving, the player's walking animation is called.

Zombie.gd

The primary function of the zombie script is to have the zombie face the player and move towards them. If the zombie touches a player, then the player dies. If the zombie is in the player's raycast when they fire their gun, then the zombie dies. When the zombie dies, it plays a sound, the zombie animated sprite is hidden, and the dead sprite is shown. Additionally, when the zombie dies, there is a chance of an Ammo appearing right above the zombie's corpse.

Zombie.gd is used for two different types of zombies: dumb zombies and smart zombies. Dumb zombies rely on cone vision and proximity detection: they only chase the player if the player enters their detection Area2D and is either inside their vision cone or close enough to be heard by the zombie. In contrast, smart zombies use full pathfinding through Godot's NavigationAgent2D, allowing them to pursue the player from any distance as long as a navigable path exists. Each frame, a smart zombie updates its path target to the player's global position and calculates the next navigation point along the route. The zombie then moves toward this point rather than directly toward the player, enabling them to naturally navigate around walls and other obstacles. Smart zombies also use the NavigationAgent2D's avoidance system, which smooths their movement and prevents collisions with other zombies by computing a safe velocity that keeps them from bunching or getting stuck.

AmmoPickup.gd

The AmmoPickup script is exactly what the name implies, and it checks if the player's collision shape enters its own. If it does, call the player's add_ammo function. Additionally, there is a slight delay before it disappears, allowing the ammo pickup noise to play.

Main.gd

Main.gd acts as the central director of the game world, managing map generation, wave progression, navigation, and enemy spawning. When the scene loads, it builds the playable arena by generating outer boundaries, creating randomized obstacle blocks, and baking a NavigationMesh that smart zombies use to pathfind around the environment. Main.gd tracks active zombies and monitors when a wave is cleared. Once all zombies are dead, it triggers a brief transition before starting the next wave. Each wave increases in difficulty by spawning more zombies, including a growing proportion of smart zombies that use navigation and avoidance. The script also periodically regenerates obstacle layouts every five waves, requiring players to adapt to this changing environment. Additionally, Main.gd handles the cleanup of old zombie corpses to prevent clutter and oversees safe spawn logic, ensuring that new zombies never appear inside walls or too close to the player.

Start.gd

In the start scene of the game, the player can start a new game, load a previous game, access controls to change them, enter settings to adjust the audio settings, view high scores, or exit the game.

Controls.gd

Allows the player to remap their inputs and save them to a controls.json file, ensuring their configurations are never lost between games.

EndGame.gd

When the player navigates to the main menu or exits the game, EndGame.gd handles the level exit

GameManager.gd

GameManager.gd serves as the global controller for persistent game data, player settings, scoring, and pause-state logic. As an autoloaded singleton, it initializes audio settings, custom key bindings, and saved high-score data when the game starts, ensuring consistency across all scenes. The script manages the player's score, ammo, kill count, and current wave progression, and provides a full save/load system that stores game progress, settings, and control configurations in JSON files. Additionally, GameManager.gd maintains a leaderboard capped to the top ten scores, automatically sorting and saving new entries. GameManager centralizes all pause behavior as well: it intercepts menu input, opens or closes the escape menu depending on the current scene, and toggles global pausing through get_tree().paused. This unified state management ensures that the UI, gameplay, and input remain synchronized, regardless of the player's location within the game.

high_score.gd

high_score.gd manages the presentation and submission of leaderboard data, acting as the user interface for viewing high scores in high_score.tscn and entering new ones when the player achieves a qualifying result. When the scene loads, it checks the GameManager to determine whether the player's current score is high enough to be added to the leaderboard. If the player qualifies for a new high score, the script switches into input mode, where the player must enter a three-letter uppercase set of initials before returning to the list. If the score is not high enough, the UI immediately displays the existing leaderboard by dynamically generating labels for each ranked entry. The script communicates with the global GameManager singleton to add new high scores, retrieve the top ten entries, and reset the full leaderboard when the reset button is pressed. HighScore.gd also provides navigation back to the start menu.

load_menu.gd

load_menu.gd manages the interface for viewing, selecting, and deleting saved game files stored in the user directory. When the menu loads, it scans the user:// folder for all JSON save files excluding settings, controls, and metadata, and displays them in a sorted list, with the most recently modified saves appearing first. Each save entry is presented as a row containing a button to load the game and a compact delete button that removes the file instantly and refreshes the list. The script automatically shows a "No Saves" label depending on whether any valid files exist. It also handles the cleanup of the last_save.json file when the associated save is deleted, ensuring metadata never points to nonexistent data. Selecting a save communicates directly with the GameManager to load the chosen game state.

minimap.gd

The minimap.gd script manages the in-game minimap by rendering a top-down view of the entire scene. It tracks the playable area and converts the world coordinates of various objects into coordinates for the minimap. Additionally, it dynamically rescales the map representation to fit the size of the minimap. Each frame, minimap.gd redraws procedurally generated obstacles, ammo, zombies, and the player. The minimap also employs color-coded shapes to indicate the positions and orientations of both the zombies and the player.

pause_menu.gd

Pause_menu.gd handles the game's pause interface. When the player presses the menu button (default: Esc), the game pauses and displays the pause menu. From the pause menu, the player can save their game, load a different level, quit the game, return to the main menu, or trigger the next wave (used for debugging).Because the pause menu must function while the game is paused, the script forces its process mode to always run, ensuring all UI interactions remain responsive

settings.gd

settings.gd handles the user interface and logic for adjusting the game's audio configuration, allowing the player to adjust Master volume, SFX volume, and Music volume. When settings.tscn loads, it reads previously saved volume values from player.json, falling back to the engine's current bus settings if no data exists, and immediately applies the correct audio levels to the corresponding audio buses. When the player adjusts the volumes with the sliders, settings.gd automatically updates the engine's audio output, and when the player exits settings.tscn their changes are saved to a .json file, ensuring their changes are consistent throughout each game.

GameObjects

Player

- Player.gd: This is the brain of the game object operation and handles how all the GameObjects of the player interact with each other and other game objects
- Graphics: This is the player's animated 2D sprite and the hidden player's dead sprite. When the player dies, it hides the player's sprite and shows the player's dead sprite.
- RayCast2D: This is the range of the gun, and anything within the RayCast when the shoot function is called dies.
- AudioStreamPlayer: There are three that handle the player's death sound, the shoot sound, and the no ammo sound.
- Camera2D: has the camera follow the player, but doesn't rotate with the player
- CanvasLayer: Normally hidden unless the player dies, in which case it appears and gives the player the ability to restart or exit the application.
- Player UI (CanvasLayer2): Displays the ammo remaining, zombies remaining, and zombies killed
- Death Screen: When the player dies, the death screen appears, allowing them to either return to the main menu or exit the game.

Zombie

- Zombie.gd: handles zombie logic
- Collision shape: allows the zombie to get stuck on other zombies and the environment
- Graphics: Similar to the player object, it has the zombie animated sprite and the hidden zombie dead sprite. When the zombie dies, it hides the zombie sprite and shows the zombie dead sprite.
- RayCast2D: If the player is in the RayCast2D of the zombie, the player dies.
- AudioStreamPlayer2D: Unlike the player, I used AudioStream2D for the zombie noises. The reason is so that the zombie noises are louder when they're near the player and quieter further away. The only audio available currently is the zombie die sound.

Ammo

- AmmoPickup.gd: handles ammo pick up logic
- Area2D: The root node
- Sprite2D: the sprite of the ammo that I made
- CollosionShape2D: allows the player to interact with the ammo by moving over it
- AudioStreamPlayer: When the player picks up the ammo, it makes a reloading sound

Player UI

- player.gd: handles the updating of the UI labels
- Zombie kill label: counts the number of zombies killed
- Zombie count label: counts the remaining zombies in the scene
- Ammo label: counts the number of bullets the player has remaining

Static Block

- These are the blocks that get stretched and rearranged during procedural generation

Scenes

player.tscn

This houses the player object. This allows the player to interact with the game, moving the character object around and firing their gun.

zombie.tscn

This houses the zombie object. These represent the dumb zombies that approach and attack the player when they come within range.

smart_zombie.tscn

This houses the smart zombie object. Structurally the same as zombie.tscn except that the zombie animated sprite2D is red instead of green.

ammo_pickup.tscn

This houses the ammo object. If the player walks over the ammo object, then the player will “pick it up” and it will disappear.

Main.tscn

Has the NavigationRegion2D that main.gd uses to procedurally generate the level as well as the player object.

Start.tscn

Allows the player to exit the application, access the controls page, start the game, adjust their audio settings by navigating to the settings scene, load a previous game, or view the game's high scores.

control.tscn

Shows the player the game's controls and allows them to customize their keybinds.

esc_menu.tscn

Appears when the player hits Esc and brings up the escape menu. There are several buttons that allow the player to go to the main menu, quit the game, skip to the next wave, load a previous game, and save their current game after naming it.

LoadMenu.tscn

An interface screen featuring a menu that allows the player to select, load, or delete any of their previously saved games. From this scene, the player can also go back to the main menu.

minimap.tscn

Contains a singular colorRect node with the minimap.gd script attached to it. The script will draw the minimap for the player

staticblock.tscn

contains the block that is copied and used for the procedural generation

Project Requirements

Changes from Project 1

From project 1, we added the following features:

- Saving and loading
- A procedurally generating level
- Two different types of zombies
    - Dumb Zombies: Run towards the player once they are close enough
    - Smart Zombies: Run towards the player regardless of distance and pathfind around obstacles
- A minimap
- The player can change their keybinds
- Animations for the player and zombies
- Pausing and unpausing
- The main menu has been changed to include the project requirements
- There is no longer a win screen since the win condition has been changed to get the high score and survive the longest number of rounds

Saving & Loading

The player can save during the game. When the player presses ESC and opens the game menu, they can type the name of their save file and save their current game. If the player attempts to save without entering a name, a warning ("!Empty") will be displayed, and no data will be saved. From within the game or the main menu, the player can click "Load Game" and be taken to the load_menu.tscn. From here, they can load a previously named save file, and it will spawn them in that level.
Additionally, the player's audio settings are saved in settings.json, ensuring their preferences are retained between games. The same is true for the player's keybinds; when the player changes keybinds, they are saved to controls.json.

Animation

The player, zombie, and smart zombie all have animations for being idle and walking. The player was going to have a shooting animation, but it looked awful when the player fired their gun very quickly, so that feature was scrapped.

Player Objective

The player's objective is to try and get the highest score by surviving all the zombie waves and killing as many zombies as they can before they are eaten by zombies.

Player Emergence

The player's emergence in this game stems from their ability to creatively adapt to the constantly changing environment, manage their limited ammo, and respond to escalating difficulty. Because the game does not prescribe a specific strategy, players naturally develop their own playstyles in order to survive longer and climb higher on the leaderboard.

Players also experience a sense of progression each time they achieve a new high score, motivating them to refine their skills, experiment with new strategies, and continually push themselves to surpass their previous records.

External Resources

https://youtu.be/HycyFNQfqI0?si=ICdPatkM8Gr9obKB
This was used for project 1 to help create the core gameplay

https://youtu.be/UYQfVx1EIW8?si=T5CeLIizu7u-sNYT
This was the original tutorial that kick-started the project and its foundations. We used the main ideas from this video to create Project 1.

https://vittles.itch.io/rotoscoped-zombie-and-soldier
The zombie and player animated sprites were taken from this itch.io page

https://youtu.be/_DP5QLJxVVI?si=I7CvOswIsiHdb8v4
Saving and loading was taken from this video as well as ChatGPT