Protect Humanity

—

Owen, Graham, Kaden, Douglas, Wyatt, Joseph, Jonathan

Overview

Goal

- We are trying to provide the large video gaming community with a novel and entertaining game.

Description

- 2D RPG based in Unity
- A robot (player character) must defend a human (computer controlled) from danger while they try to escape the planet.
- The robot cannot directly attack, and thus must prioritize the life of the human.
- The player will roam a 2D map with enemies dispersed throughout.
- Encounters consist of a turn-based battle
- Items can be found in the world and added to the player inventory
- If the robot dies, he can be replaced by a new one
- New abilities will be unlocked as the game progresses (after boss encounters)

Software design

Combat Scene
- Battle sprites
- Enemy Class (stats/move data)
- Animations(maybe)

Overworld Scene
- Maps
- Interactables
- Enemy Movement

UI
- Unity UI Tools(TextMeshPro)
- Options, Inventory, Combat Menus

Player
- Player class (holds stat/ability data)
- Movement
- Inventory (holds item data)

AI
- Stats, move list & chances
- Dialogue options
- Sprites

Software architecture (components & interfaces)

Interfaces

-Overworld Scene to Battle Scene

-Player (movement) to Overworld Scene

-Ai to Combat system (stats, ability list, & dialogue)

-Player to Combat system (stats, abilities, and items)

Software architecture (data systems, assumptions)

Player class

Player stats (HP, resists, damage, xp, level, speed), Player abilities list and Player inventory
AI class

AI stats (HP, resists, damage, speed), AI abilities list (and chances for each), Possible dialogue

Enemy Database

Stats for each encounter (types, stats), Attack patterns

Assumptions
Currently we don't plan to have the ability to save the game because the game should be short enough to play in one session

Software architecture (alternatives)

Central Database

Holds all information about Player, AI, and Enemies in one, easily accessed location

-Pros: Everything is stored in one place

-Cons: Need access to database at all times

Event-driven approach

Information stored in memory and tracked through event listeners and updated in real time

-Pros: Improves performance by reducing data fetches, Makes it modular and easier to expand

-Cons: More complex to implement due to event dependencies, Requires more management to prevent memory leaks

Coding Guidelines

To maintain well structured and organized code, we will be following c# coding guidelines which include the following rules

1. Use Pascal case for class and method name
2. Use Camel case for arguments and local variable names
3. Avoid underscores in variable names
4. Avoid System data types when predefined data types exist
5. Preface all interface names with I
6. Align curly braces vertically
7. Declare variables as close as possible to their use
8. Declare variables as private when possible

Process Description (Risk assessment)

1. Lack of experience using Unity/difficulty finding resources on using Unity
    a. Low probability
    b. Low impact
2. Creating too much unique art is too time consuming
    a. High probability
    b. Low impact
3. Conflicting schedules
    a. Low probability
    b. Low → high impact
4. Github pushing errors
    a. Medium probability
    b. High impact
5. Github merging errors
    a. Medium probability
    b. High impact

Project Schedule: Overview

Weeks 1 - 2: Explore ideas, design methods

Weeks 3 - 4: Implement basic functionality and design tests

Weeks 5 - 6: Develop to prototype level

Weeks 7 - 8: Test and add stretch goal features

Weeks 9 - 10: Debug and Deploy

Team Structure

- Overworld: Graham, Joseph, Owen, Jonathan
  - Design the overworld map and implement the movement system, discovering and acquiring items as well as initiating encounters.
- Combat: Kaden, Douglas
  - Design and implement the turn based combat system.
- Player Progression: Wyatt, Owen
  - Design and implement player stats with xp scaling, as well as equipable items to give bonuses in combat and an inventory system to store them.

Use Cases

1. Move
2. Collide with environment
3. Enter a battle
4. Battle Event: Human Action
5. Battle Event: Enemy Action
6. Battle Option: Defend
7. Defeat an Enemy
8. Pick up an Item
9. Lose the game
10. Level Up
11. Gain new ability

Use Case: Movement

Actors: Player

Triggers: The player is in the overworld.

Precondition: The player is in the overworld, the player is not in a battle or menu

Postconditions (success scenario)

The player position changes along the direction depending on the key pressed

i. W -> Up
ii. A -> Left
iii. S -> Down
iv. D -> Right

Use Case: Battle Option: Defend

Actor: Player
Trigger: The human loses hp in a battle.
Preconditions
- The human is alive (hp $>$ 0)
- The human's current hp is lower than their max hp
- The defend option is not on cooldown

Postconditions:
- The human's defense stat is increased.
- The human takes reduced damage from attacks.

Use Case: Pick up an item

Actor: Player
Triggers: The player sees an item in the overworld.
Preconditions:

- The item is visible on the screen.
- The Player is not currently in a battle.
- The player's character has a clear path to the item.

Postconditions:

- The item no longer shows on the map
- The item is added to the player's inventory
- The item picked up is the same as the one shown on the map

Documentation Plan

- Guidebook on game mechanics
  - Movement
  - Combat basics
  - Progressing through stages
  - Using items
- Wiki
  - Enemy attacks
  - Unlockable abilities
  - Map tips/tricks

Test plan & bugs

- We will to test each of our Use cases as previously outlined
- We will use Unity's built in Test Runner tool to conduct Unit and System (Integration) testing.
  - Facilitates automated tests on specific functions and systems while asserting their outputs
- We will conduct usability testing by running through our game and observing any potential usability issues
- Upon discovering a bug, we will use Github Issues to track it.

Non-functional Requirements

- Transitions between the overworld and battle should finish within 5 seconds.
- Transitions between stages should finish within 5 seconds.
- The game should maintain a consistent frame rate of at least 30 FPS.
- All third-party assets and music used must have proper licensing or be public use.

External Requirements

- The program will be able to withstand reasonable user input errors and will respond accordingly.
- The game must be downloadable and playable for users on their own devices.
- Software used for the entire program should be accessible to others so that they can develop their own versions. As a result, the program should be well-documented and understandable by others.