Living Document

Team Info

Team Members:

- Douglas Sandford - Developer
- Kaden Allen - Developer/QA Engineer
- Joseph Liefeld - Designer / Developer
- Jonathan Hotchkiss - Developer, Tester
- Graham Glazner - Coordinator, Developer
- Wyatt Fujikawa - Developer / Designer
- Owen Wickman - Developer/QA Engineer

Github:
https://github.com/LitchDoctor/winter2025-group2-rpg.git
Trello:
https://trello.com/invite/b/6781aa06f4132bde26a088ae/ATTI281728df6d053a134dd1f0f5fcf09ab6E74188A9/pt-2-group-2

Communication

Discord and Text Group chat

Rules

- Check Discord regularly
- Respond within 8 hours if pinged on discord
- Text Group chat for high priority messages

Project Description: Protect Humanity RPG

Abstract

Protect Humanity is a role-playing game where the player plays a robot who travels through a dangerous, apocalyptic world. Humanity is escaping to Mars. All of the humans have left Earth or have been mutated into horrifying monsters, except for one. Your job is to escort the last living human, safely to the launchpad.

The robot leads the team of two with the human following behind. They travel through the world, encountering enemies and items. Colliding with an item allows the player to press a button “E” to pick it up. In the corner of the screen there is a button to open inventory. When the player picks up an item it is added there. The player can equip only a single item at a time. Equipping an item updates the player's stats.

The map includes enemy encounters that enter the player into a turn-based battle. To initiate combat, the player moves until the robot collides with an enemy. A pop-up comes up to choose whether to initiate combat. If the player selects “start", the fight starts after a brief transition. If the player selects "run", the pop-up closes. For each turn, the player must choose an action to keep the human alive. The player starts with 3 options every turn: defend, taunt, or empower. The enemies either target the robot or the human. Each round, the human will also act, though they may not always be helpful. His move will be randomly generated. He has a percent chance to attack or do something stupid. During a battle, a text box shows the actions taken by each participant. The robot gains a new ability after each of the first two bosses. The abilities gained will be “heal” and “stun”. If the robot dies in combat, but the human manages to defeat the enemy, a new identical robot replaces the old one.

At the end of the first two stages are mini-boss enemies which block the path to the next stage. At the end of the last stage is the final boss. Defeating the final boss allows the human to get on a spaceship and escape to mars, joining the other survivors.

	Robot	Human
Health	yes	yes
Cooldown Speed	yes	no
Speed(turn order)	yes	yes
Damage	no	yes
Defense(damage reduction)	yes	yes

Items
Armor: Increases Robot + Human defense
Book: Reduces Human stupidity
Leash Backpack: Reduces Human follow distance
Shield: Improves ‘defend’ effectiveness
Clown Mask: Improves ‘taunt’ effectiveness
Pom Poms: Improves ‘empower’ effectiveness
Running Shoes: Increases Robot; Increases Human speed & Movement
Machete: Increases Human Damage
Nun-Chucks: Increases Human Damage; Increases Human Stupidity
Coolant: Reduced Cooldowns on abilities
Hamburger: Increases Human Health
Steel: Increases Robot Health
Cell Phone: Improves Robot Health, Speed, and Cooldowns; Increases Human stupidity;

Goal

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

Current Practice

Today’s RPGs are typically developed with game engines such as Unity or Unreal Engine. These games often feature vast open worlds and real-time combat. Many RPGs also integrate online multiplayer and dungeon raids with a group of players.

Novelty

Unlike previously made RPG’s, our game will have the player be protecting another entity, which differs from other games where the focus is just on the player's survival.

Effects

In our day and age, the video gaming industry is at an all-time high demonstrating how highly our society values entertainment. The game we propose will inject itself into the said industry and provide the community with a new and valuable source of entertainment.

Use Cases

Use Case 1: Player movement
Author: Owen Wickman

1. Actors
    a. Player
2. Triggers
    a. The player is in the overworld.
3. Preconditions
    a. The player is in the overworld (not in a menu or battle)
4. Postconditions (success scenario)
    a. The player moves in a direction depending on the key pressed
        i. W -> Up
        ii. A -> Left
        iii. S -> Down
        iv. D -> Right
    b. If two keys are pressed, the player will move in the average direction
        i. W and D for example will cause the player to move up and to the right in a diagonal.
5. List of steps (success scenario)
    a. The user presses a movement key
    b. The player moves in the correct direction ex: W -> Up
6. Extensions/variations of the success scenario
    a. The user presses multiple movement keys simultaneously
    b. The player moves in the average direction ex: S and D -> Diagonally Down and Right
7. Exceptions: failure conditions and scenarios
    a. The player fails to move when movement keys are pressed
    b. The player moves in the wrong direction

Use Case 2: Enter a battle
As a player, I want to enter an enemy battle encounter so that I can gain xp and unlock new areas.
Author: Graham Glazner

1. Actors
    a. Player
2. Triggers
    a. The player sees an enemy on screen.
3. Preconditions
    a. The enemy is visible on the screen.
    b. The character is not currently in a battle.
    c. The player’s character has a clear path to the enemy.
4. Postconditions (success scenario)
    a. The player’s character is in a battle.

    b. The enemy, the character and the human npc are displayed.
    c. The enemy sprite matches that of the enemy in the overworld view.
    d. The character's battle options are shown.
5. List of steps (success scenario)
    a. The player moves their character so that it collides with the enemy.
6. Extensions/variations of the success scenario
    a. The transition to battle animation plays as the view changes from the overworld to the battle.
7. Exceptions: failure conditions and scenarios
    a. The Character goes through the enemy sprite without entering a battle.
    b. The player can't move the character to the enemy due to an error.

Use Case 3: Battle Option: Defend

As a user, I want to defend the human during a battle to prevent the human losing all of its health.

Author: Jonathan Hotchkiss

1. Actors
    a. Player
2. Triggers
    a. The human loses hp in a battle.
3. Preconditions
    a. The human is alive (hp > 0)
    b. The human's current hp is lower than their max hp
    c. The defend option is not on cooldown
4. Postconditions (success scenario)
    a. The human's defense stat is increased.
    b. The human takes reduced damage from attacks.
5. List of steps (success scenario)
    a. Battle options are shown.
    b. Player selects the heal option
6. Extensions/variations of the success scenario
    a. If the enemy attacks the robot, the robot takes damage as normal
        i. The human's defense stat remains at an increased amount.
7. Exceptions: failure conditions and scenarios
    a. The human's defense stat remains unchanged
    b. The human's defense only changes superficially and it still takes normal damage.

Use Case 4: Defeating an Enemy

Author: Joseph

1. Actor: Player
2. Trigger(s): Using an attack that reduces the enemy's health points to less than or equal to zero

3. Precondition(s):
    a. In battle state
    b. Enemy has remaining health (hp > 0)
4. Postcondition(s) :
    a. Sprite disappears from screen
    b. “Enemy defeated” text box is displayed

5. List of Steps (success scenario):
    a. User enters a fight
    b. User uses abilities that deal damage to enemy

6. Extension/Variation:
    a. User dies before defeating enemy

7. Exceptions (failure conditions + scenarios):
    a. Enemy’s HP is reduced to <= 0, yet the enemy remains one screen and continues taking turns

Use case 5: Pick up an item
Author Douglas

1. Actor: Player

2. Triggers:
    a. The player sees an item in the overworld.

3. Preconditions:
    a. The item is visible on the screen.
    b. The Player is not currently in a battle.
    c. The player’s character has a clear path to the enemy.

4. Postconditions:
    a. The item no longer shows on the map
    b. The item is added to the player’s inventory
    c. The item picked up is the same as the one shown on the map

5. List of steps (success scenario):
    a. The player moves to the Item
    b. The player presses E to pick up the item
    c. The player opens their inventory

6. Extensions/variations of the success scenario
    a. The item may be equipped from the inventory menu

7. Exceptions: failure conditions and scenarios

    a. The Item still displays on the map
    b. The Item disappears, but is not added to the inventory
    c. Player can't reach the item due to an error

Use case 6: Player Defeat
Author: Kaden

1. Actor: Player

2. Triggers: Player is in a battle with low hp.
3. Preconditions:
    a. Player is in a battle
    b. The robot has more than 0 hp.
    c. The robot has less than 10% hp.

4. Postconditions:
    a. Depending on which entity dies, either combat continues without user input or the game over screen displays
        i. Combat continues without user input (robot death)

5. List of steps (success scenario)
    a. Player chooses a battle option.
    b. One of the allied is damaged by enemy
    c. Player's Hp falls to 0
    d. Death message displays
    e. Combat continues without player input. (Until AI dies or battle is won)

6. Extensions/variations of the success scenario
    a. If the AI companion dies instead of the player character, combat ends with a game over screen.

7. Exceptions: failure conditions and scenarios
    a. Combat continues after the human companion dies
    b. Player remains able to act after reaching 0 hp
    c. Game over screen fails to display
    d. Combat is treated as a victory even if the player loses

Use Case 7: Levelling Up
As a user, I want my character to level up as a reward for defeating an enemy so that I have a reason to defeat enemies and get stronger.
Author: Wyatt

1. Actor: Player
2. Trigger(s): The player's character has almost enough xp to level up.
3. Precondition(s):
    a. The player's character has 90% of the xp needed to level up.

4. Postcondition(s):
    a. Pop up message saying “You have leveled up!”
    b. Increase in all player stats
    c. Experience is “reset”
5. List of steps (success scenario):
    a. Player defeats entity
    b. Player xp increases
    c. Player levels up and a pop up message is generated
    d. Another pop up is generated showing statistical increases
        i. Example: 1 → 4
    e. Pop up closes and user is returned to the overworld
6. Extensions/variations of the success scenario
    a. Skipping levels if enough experience is gained leads to greater stats increase
7. Exceptions: failure conditions and scenarios
    a. Experience isn’t properly saved to the player class after an enemy is defeated
    b. Messages don’t pop up when player levels up
    c. Player stats aren’t properly updated or saved
    d. No new abilities are learned even if they’re supposed to be (if we add new abilities)

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.

Team Process Description

Software Toolset

- Unity includes a 2D preset that we can build off of and is designed for game creation. Our art requirements are not large, so we should be able to make everything that we need in Piskel.

Team Structure

- Kaden: Developer/ QA Engineer
  - Our team needs many developers since there is a lot of different types of development work to be done. I have played many different video games so I should be able to identify quality problems before they become ingrained in our systems.
- Douglas: Developer/ Combat Engineer
  - Our team needs people to design each part of the game. I have experience in playing multiple different kinds of turn-based games. I should be able to efficiently make changes based on testing.
- Jonathan: Developer/ Tester
  - Including developers in the team is crucial to being able to create a working program. Without the developers, there would be no running program. Testers are also important to ensure the program will run without errors and each system component will function properly. I am suited to take on these responsibilities because I have experience developing programs and small games throughout previous classes. I also have some experience with testing, and I understand the importance of carefully testing each portion of the program.
- Graham: Coordinator/ Developer
  - Our team needs a coordinator so that we can keep everyone on the same page. The coordinator will communicate across the development team to make sure people know what they need to do. I am fit for the position because of my communication skills and awareness. I will also be contributing to the development of the top-down exploration portion of the game.
- Joseph: Designer / Developer
  - Our team needs several developers to work on various parts of the game, to ensure that no part falls behind schedule. I am fit for the role of Designer because I have played a wide variety of video games that all have their own unique traits and design choices, and I understand how developers use certain game mechanics to teach and engage with the player.
- Wyatt: Designer/Developer

        - Our team needs multiple developers in order to address multiple areas of focus required to build the game we're envisioning. As someone who has played several games where you customize your own character, build your own teams, and create your own buildings, I am fit for the role of designer. I also pay close attention to the niche details most people skip over when designing characters and npc's which will be a helpful trait when designing characters and the map throughout this project.
    - Owen: Developer / QA Engineer
        - Our team requires a diverse set of developers to handle the various types of development work. Having played numerous games, including several RPGs, and with experience developing an RPG in Unity, I bring relevant expertise to my role as both a QA Engineer and developer.

- Provide a schedule for each member (or sub-group) with a *measurable*, concrete milestone for each week. "Feature 90% done" is not measurable, but "use case 1 works, without any error handling" is.
    - Combat Team (Doug/Kaden)
        - Week 1: Explore different ideas for the combat system
        - Week 2: Formalize combat requirements
        - Week 3: make characters classes
        - Week 4: Create an ability structure
        - Week 5: Simple combat system
        - Week 6: Add in-game over system
        - Week 7: Create an Item system
        - Week 8: add the ability to use the items
        - Week 9: Bug hunting/polish game
        - Week 10: Ensure successful product launch

    - Overworld Team (Graham/Joseph)
        - Week 1: Form the team.
        - Week 2: Connect unity + github.
        - Week 3: Player Movement Use Case works, no error handling.
        - Week 4: Obstacles stop the player's movement.
        - Week 5: Map design is complete; The Player can only move within the set bounds of the map.
        - Week 6: Encounter Use Case works without error handling. At least 4 combat encounters are included on the map.
        - Week 7: Game over resets the player's position to the start.
        - Week 8: Pick up item use case working. At least 3 NPCs are included on the map.
        - Week 9: Use cases mentioned above work with error handling.

        - Week 10: All sprites, animations and assets are loaded in. Use cases still work as expected with error handling.

    - Player Progression (Wyatt)
        - Week 1: Flesh out a basic character design
        - Week 2: Establish an experience system and player class with stats that will be used
        - Week 3: Create basic UI including messages and a level up screen
        - Week 4: Work with other teams to determine specific animations and/or sprites for levelling up
        - Week 5: Assign multiple player classes to all entities that can level up
        - Week 6: Begin implementing and testing in actual game to find errors
        - Week 7: Error checking
        - Week 8: continue to error check and work with other groups to fix issues
        - Week 9: IF done, consider adding more functionality (skills, abilities, etc.)
        - Week 10: wrap up

    - Sprite/Animation Team (Jonathan/Owen)
        - Week 1: Formalize team assignments and determine key steps to complete for each week
        - Week 2: Determine the number of characters that will be necessary and develop 2-3 ideas for the main user character
        - Week 3: Create a basic working character design to allow other teams to test working capabilities
        - Week 4: Create a working animation for entering and leaving battle scenes without testing UX Design components
        - Week 5: Develop characters for enemies that will be fought throughout the game progression
        - Week 6: Create all “bot”/NPC characters that will be throughout the map
        - Week 7: Review all characters created to this point. Test each character to make sure they are properly implemented.
        - Week 8: Make adjustments to characters not working/appearing properly
        - Week 9: Test battle scene animations, and create an appealing animation that keeps users engaged
        - Week 10: Review all created characters/animations. Correct any remaining errors that are affecting the game flow.

- Specify and explain at least three major risks that could prevent you from completing your project.

    - None of our group members have worked on a project like this before, so if there is not a relevant tutorial that we can reference, we may not be able to create the features that we want.
    - Creating art may take too much time so we may have to cut corners on looks.
    - Different teammate schedules may keep us from meeting consistently.

- Describe at what point in your process external feedback (i.e., feedback from outside your project team, including the project manager) will be most useful and how you will get that feedback.
    - External feedback will be most helpful once we get the majority of our systems online towards the beginning of beta testing. This feedback can be given by the TA or anyone not involved in the project. We will mostly be interested in game feel and player experience. This info should allow us to tweak our final product to be more suitable for a wider audience.

Technical Approach

We will create the game using the Unity game engine and C #. The sprite assets will be created using pixel art and the online sprite editor Piskel.

Features

- **Top-Down Exploration**
    - Player Movement
    - Player Movement Animation
    - World Design
- **Combat System**
    - Satisfying Encounter-Based Combat
    - Player Decision Menu
    - Enemy/Human Decision Making (RNG, Decision Tree, etc.)
    - Battle Graphics
    - Descriptive Combat Text (e.g., “The enemy uses laser attack!”, “Jason trips on a rock!”)
- **Player Progression**
    - Leveling System
    - Sprite Designs
- **Item System & Inventory**
    - Items & Player Inventory
    - World Interactions (e.g., use axe to cut down a tree)
- **Story & Worldbuilding**
    - NPC Dialogue & Encounters
    - Overarching Story

- **Stretch Goals**
    - Save/Load Game
    - Multiplayer
    - Celebration Animations
    - Music & Sound Effects

Software architecture

- Software Components:
    - Combat Scene
        - Enemy AI
        - Combat system
        - A dialogue system that explains what attack each person is using
        - Stat systems (xp, and strength, health)
    - Overworld Scene
        - Maps
        - NPC system to allow you to talk to NPCs
        - Interactable environment (use an axe to cut down a tree)
    - Player Component
        - Movement
        - Player Class (stats/items)
            - inventory
    - AI class
        - AI pathfinding
        - Stats
        - inventory;
        - Dialogue when talked to
    - Movement
        - Movement speed

- Specify the **interfaces** between components.
    - Overworld Scene to combat scene
        - Passes stats, AI stats, abilities
        - Triggers the combat UI and loads the combat scene
    - Movement to Overworld scene
        - Updates the position of the player based on the movement imputed
        - Triggers collisions to make a pop-up to allow the user to choose whether to initiate combat
    - Ai and player classes to combat system

        - Stores stats, abilities, and inventory so you can use it in combat
    - Dialogue system
        - Gets the dialogue based on the encounter and NPC
    - Interactive environment
        - Allows for gathering items from the environment

- Describe in detail what **data** your system stores, and how. If it uses a database, give the high-level database schema. If not, describe how you are storing the data and its organization.
    - Player class
        - Player stats (HP, resists, damage, xp, level, speed)
        - Player abilities list (array)
        - Player inventory
            - Uses an array to store objects within unity
    - 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

- If there are particular **assumptions** underpinning your chosen architecture, identify and describe them.
- **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

Software design

Provide a detailed definition of each of the software components you identified above.

- What **packages, classes, or other units of abstraction** form these components?
    - Combat Scene
        - The combat scene takes information from the overworld scene in order to render the correct enemy encounter.
        - Combat relies on the Player and AI classes to know what both the player character and the AI Human are allowed to do at that point of the game.
    - Overworld Scene
    - The overworld is formed by textures and sprites to create the environment that the player will walk through, which will be found using the Unity Asset Store. The overworld will also use colliders to create boundaries, ensuring the player cannot walk outside the design of the map or through objects on the map.
    - Player Component
        - This component encompasses all that has to do with the player that the user is going to control. It includes classes to manage the player's movement in the overworld, tracking of stats and ability progression, as well as ability usage during combat encounters
    - AI Class
        - This component encompasses all that has to do with the AI human. In the overworld it has a follows class to cause it to follow the players movements. During combat it has another class to dictate the actions it takes during its turn, and keep track of its remaining health.
    - UI
        - Packages include TextMeshPro, and it used in all scenes. It includes components to show stats, menu options, inventory menu, and combat options.
- What are the **responsibilities** of each of those parts of a component?

Coding guidelines

www.geeksforgeeks.org/c-sharp-coding-standards/

The coding guidelines from the above link will be used for the project. These specific guidelines were chosen because they enhance the readability of the code. They also will force the developers to keep a clean code appearance that allows someone joining the project to easily understand what is going on.

To enforce these guidelines throughout the project, members of the team will review the code. One plan is to have coding “buddies” where members will get together and work on developing the program. During these meetings, one member will be coding, while the other watches and assists. This will improve the consistency of the code and ensure that the code will follow the guidelines.

Process description

Expand your process description to address the following five topics:

i. Risk assessment

Identify the top five risks to successful completion of your project.

For each, give:

1. Inexperience with the given technology
    a. Low probability
    b. Low impact
    c. Relevant tutorials were easy to find through a simple search on YouTube. Based on the ease of that search, it’s realistic to assume that if a more specific search is conducted, the desired information will be found.
    d. Bookmarking relevant information, group collaboration, setting goals to give a clear idea of what needs to be searched for in order to achieve goals.
    e. Looking in advance for videos that might pertain to the project beforehand. This allows for more time to plan out how to program the desired feature.
    f. Group collaboration and if the group is still unable to figure out how to code the desired feature or resolve an issue, consult the TA’s and professor for advice or recommendations on resources to troubleshoot.
2. Creating art may take too much time so we may have to cut corners on appearance.
    a. High probability of occurring

    b. Low impact
    c. While timelines are still adjusting slightly, we have based the timelines on our progress to this point
    d. To reduce the likelihood of this risk having a large impact, map creation was started early on to give sufficient time. It can be difficult to determine exactly how long the art will take, but keeping similar designs across maps will help us get a better estimate for the other maps we plan to develop
    e. This problem is relatively easy to detect by looking at the progress of the map every couple of days and seeing how much is left to complete. Another plan to detect issues with art is to run automated tests for character movement interacting correctly with art features
    f. Should there be a need to make adjustments, we have determined a range for how many maps are necessary, allowing us to remove maps or adjust the sizes to increase production time.
3. Different teammate schedules may keep us from meeting consistently
    a. Low → High probability
    b. Low → High impact
    c. Out of the 6 total meetups that have been conducted outside of class time, only 2 meetups had all 7 group members present. Group members were brought up to speed during mandatory meetings and work was still done in a timely manner but it might be more difficult as bigger deadlines approach.
    d. Planning to meet far in advance, keeping the trello board with goals and tasks to do updated, letting team members know in advance if they're available or not and how they're doing on the completion of their respective tasks.
    e. Everyone states what days during the week that they're free and if they're unable to attend a meeting in person, their online availability. If they're unable to attend online, what they plan to work on and communicate that with the group.
    f. If no one is able to meetup, a discord message should be sent stating this as well as what they intend to work on.
4. Not meeting deadlines/time management
    a. Medium likelihood
    b. Medium impact
    c. Each week, there are usually at least 2 different goals that have to get pushed back. They sometimes come from being too aggressive with the deadline and expecting more than feasible. Other deadlines have been pushed back because of issues/errors with developing the program.

    d. To reduce this risk, we have decided to review one another's weekly goals to verify that it is a reasonable amount to complete within the time frame. On top of this, errors are discussed and worked through as a group as quickly as possible to keep on track with the rest of the timeline.
    e. This problem is very easy to detect. Either errors will appear when trying to run the programs, or goals/tasks will not be completed when initially specified.
    f. As soon as this problem is detected, we will get multiple people working on the task together to complete the tasks efficiently and get back on track. Another mitigation plan is to push back less important tasks to allow for more time to solve errors and complete the task at hand.
5. Scope Creep/ Too many features
    a. Medium likelihood
    b. Medium impact
    c. No tests or experiments were available to base estimates off of. However, adding many features can cause us to spend too much time on less important aspects of the program, taking away valuable time on other necessities. When initially designing the project, we might add more features than feasible because of not recognizing the potential time commitments that the features may require.
    d. To minimize the likelihood of this occurring, we kept features relatively simple to begin with. Sticking to a simplistic design allows for flexibility to add more features as the project progresses and time permits.
    e. This problem is tricky to detect initially, but will become more apparent as deadlines approach. The difficulty to identify the problem in the beginning stems from a lack of knowledge for how long features will take to implement. As we continue, it becomes more clear how much time we have and what expectations are for the completed project.
    f. The mitigation plan will entail identifying which parts of a feature are necessary and which ones can be done without. Once identifying this, we will focus on completing the necessary components of a feature and only add the additional features if time allows.

ii. Project schedule

Identify milestones (external and internal), define tasks along with effort estimates (at granularity no coarser than 1-person-week units), and identify dependences among them. (What has to be complete before you can begin implementing component X? What has to be complete before you can start testing component X? What has to be complete before you can run an entire (small) use case?) This should reflect your actual plan of work, possibly including items your team has already completed.

Major Milestones:	Tasks to Achieve Milestones:
Map Creation	● Create Paper Prototypes for Maps ● Gather assets (textures, sprites) ● Create tilemap for floor + edges ● Create tilemap for sprites (items, interactable objects,) ● Add physics to objects ● Add pop-up menus for item pickups, points of interest, enemy interactions
Player Movement	● Create player movement script ● Create terrain collision logic ● Create collision logic for triggering encounters
Player Inventory	● Create the inventory script to create the actual inventory ● Create interactable items that can be stored in the inventory ● Assign effects to the given items ● Gather assets (sprites) ● Create inventory UI ● Refine and test all features
Combat Scenes	● Create combat animations ● Develop enemy attacks ● Animate character attacks ● Display Damage and health components
Player Progression	● Create the abstract item object ● Create public player stats ● Make player stats scale off of xp ● Make player xp increase on encounter completion ● Make abilities unlock on level completion

iv. Test plan & bugs

We will test all of our use cases as previously outlined. These were made with the purpose of encapsulating every core and essential functionality of our game, and testing them will be sufficient. Additional bug discovery may come from user bug reports, or random discovery during development.

Unit testing and system integration testing will be conducted using Unity's built-in test runner tool. This will allow us to supply inputs to specific functions or systems and assert the expected outcomes based on post-call behavior. For usability testing, our team members will conduct hands-on testing of various interactions and systems, validating the functionality through their observations. Before the game's full release, we will also provide it to external testers to gather feedback and ensure a positive user experience.

Some Identified Test Suites

- Movement
    - Input: keypress ‘w’
        - Assert velocity vector2D (0, 1)
    - Input: keypress ‘a’
        - Assert velocity vector2D (-1, 0)
    - Input: keypress ‘s’
        - Assert velocity vector2D (0, -1)
    - Input: keypress ‘d’
        - Assert velocity vector2D (1, 0)
- Use Item
    - Input: left click on the ‘use’ button in combat UI
    - Assert specific item action function called and returned 1 for success
- Use health potion on human
    - Input: call use health potion function
    - Assert human hp post consumption is 25 higher.
- Pickup Item
    - Input collider triggered between player and item
    - Assert item appears in player inventory

System test

- Full playthrough
    - We will have both a team member and non team member attempt a full playthrough of the game. In each case, we will instruct the tester to try and test the boundaries of the game to find unique situations that may cause bugs.
- Combat Test

- A smaller version of the previous system test, we will load a tester into a predetermined combat encounter.

Upon discovering a bug, we will create a new issue using Github's issues feature and assign it to whichever team is responsible for the problematic region of the software.

We require that you use GitHub Issues

v. Documentation plan

For our game, we will develop a brief guidebook to inform the player about the mechanics of the game including, How combat works, how to progress through stages, how to use items, etc. We will also make a wiki with more detailed information on the stages of the game, such as what moves the enemies move, what abilities you unlock and more under-the-hood information. These will both be written once a playable version of the game is ready.

Technical Guides – Explain how to install, configure, and run the software, often including architecture overviews and system requirements.

The application for our game will be available as a download for Windows or a download for macOS.

Developer Guides – Provide instructions on how to extend or modify the software, including guidelines for contributing.

The Unity Project is called "Protect Humanity" This is where all of the code, including sprites settings. After cloning the repository, install Unity Hub. Using Unity hub and Unity Engine version 6000.0.34f1 you can open the "Protect Unity" from disk. This is the best way to alter or add to the program.

Reflections

Every team member should describe at least three lessons that they learned from the project experience. What worked exceptionally well and what would you have done differently?

Douglas Sandford
Kaden Allen
Jonathan Hotchkiss

1. The biggest lesson I learned from this project is that having unambiguous requirements and expectations before beginning a project will make development much easier. Vague requirements can make it more difficult to determine what should and should not be included. Setting specific requirements at the start lessens the chances of miscommunication and creates a strong foundation for the project to build off.
2. Communication is key. There is no way to complete a successful project as a group without constant communication. Whether it is checking in to make sure everyone is on the same page, or determining the goals for the week, it is vital to discuss with one another where the project is at. Throughout our project, communication was pretty good, which allowed us to quickly work with one another if issues arose.
3. Being flexible and open to new ideas is extremely important. It allows us to understand different perspectives that can aid in improving the quality of the project. Flexibility makes it possible to shift responsibilities when it is necessary and also adjust requirements if a situation comes up that requires change.

Graham Glazner

1. We did a great job regularly communicating and meeting together to work. I did learn that having specific objectives is a good idea. When planning week to week, it was sometimes difficult to come up with what our next steps should be. If we had a more detailed plan from the beginning, we could have had more success.
2. I also learned the importance of having a specific description of how a software product is supposed to work. Having something to reference and update when making decisions can keep the team from making incompatible code.
3.

Wyatt Fujikawa
Owen Wickman