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

Trello

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. If the robot dies in combat, but the human manages to defeat the enemy, a new identical robot replaces the old one.

Stats:
Level: Player level
HP: Max health
Durability: damage reduction
Energy: healing effectiveness
Support: damage amp

Items
Armor: Increases Robot durability
Cheeseburger: Increases healing effectiveness
Machete: Increases robot damage amp
Big Steel Bar: General Stat increases

Goal

We are trying to provide the large video gaming community with a novel and entertaining game that brings a new way of playing games.

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. While many games use one character that the user has full control over, Protect Humanity does not. The human is supposed to be protected and must survive, while the robot is the character that the user controls. It adds a new twist that requires the user to be extremely careful how they choose to fight enemies to ensure their human teammate survives.

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.

Scheduling

- 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.

    - Inventory (Wyatt)
        - Week 1: Flesh out a basic inventory system
        - Week 2: Create test items to begin testing inventory functionality
        - Week 3: Create basic UI for item pickup
        - Week 4: Create inventory UI with 9 slots
        - Week 5: Refine inventory UI and create button to toggle inventory UI on and off
        - Week 6: Test to see if items in inventory appear properly in assigned slot
        - Week 7: Error checking
        - Week 8: Create item description UI and equip/unequip buttons for armor
        - Week 9: Test, refine, test again, and troubleshoot
        - Week 10: wrap up

    - Menu/Testing (Jonathan)
        - Week 1: Formalize team assignments and determine key steps to complete for each week

- Week 2: Determine the necessary scenes/components that will be used throughout the project
- Week 3: Organize clear, achievable requirements and architecture for the remainder of the project
- Week 4: Develop a menu page to guide users into gameplay
- Week 5: Improve appearance and functionality of main menu
- Week 6: Add controls page with basic movement, inventory, and item pickup controls
- Week 7: Test functionality and user experience with main menu
- Week 8: Make adjustments based on tests done previously
- Week 9: Clean up errors that occurred during beta testing and test for other bugs
- Week 10: Polish documentation for living doc, developer guides, and player guides

Processing 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.

We will also seek feedback on use cases that we determine to be working. Allowing users to test out the use cases will help us find bugs in the program that other users might encounter. Getting feedback on these use cases will give us a guideline as to what needs to be cleaned up to have a fully functional program with minimal errors.

Technical Approach

We will create the game using the Unity game engine and C#. Unity supports various types of games. It has C# scripting, editing, and a physics engine. Additionally, it allows for many different platforms including Windows, MacOS, and Linux.

The sprite assets will be created using pixel art and the online sprite editor Piskel. The pixel art allows our game graphics to be consistent throughout, where many scenes are slightly pixelated and blocky-looking. Piskel works great for creating sprites from scratch and provides a slightly larger variety of editing options compared to the pixel art.

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)
    - Battle Graphics
    - Descriptive Combat Text (e.g., “The enemy uses laser attack!”, “The human trips on a rock and misses!”)
- **Player Progression**
    - Sprite Designs
- **Item System & Inventory**
    - Items & Player Inventory
    - World Interactions (e.g., picking up items off of the ground)
- **Stretch Goals**
    - Save/Load Game
    - Multiplayer
    - Celebration Animations
    - Music & Sound Effects
    - Levelling System

Software architecture

Software Components

- Combat Scene
    - Enemy AI
    - Combat system
    - A dialogue system that explains what attack each person is using
    - Stat systems (strength, health, support, etc.)
- Overworld Scene
    - Maps
    - Interactable environment (picking up items, running into obstacles)
- Player Component
    - Movement
    - Player Class (stats/items)
        - inventory
- AI class
    - AI pathfinding
    - Stats
    - Inventory

        - Attacking in battle
    - Movement
        - Movement speed/direction
    - Menu
        - Playing game
        - Learning Controls
        - Quitting Game

Interfaces

- Overworld Scene to combat scene
    - Passes human 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 input
    - Triggers collisions to make a pop-up to allow the user to choose whether to initiate combat
    - Triggers item pickup ability when close to items
- AI and player classes to combat system
    - Stores stats, abilities, and inventory so you can use it in combat
- Interactive environment
    - Allows for gathering items from the environment
- Menu Scene
    - Sends user to starting scene
    - Presents user with control options
    - Exits the entire program

Data and 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

Event-driven Architecture Approach

Information is stored in memory and tracked through event listeners and updated in real time. Protect Humanity includes many different event listeners to control the flow of the program. When particular events occur, some data is passed to various aspects of the game to produce the correct results. Some pros and cons to this style of programming are listed below.

- Pros:
  - Improves performance by reducing data fetches
  - Makes it modular and easier to expand
  - Allows for real-time processing
  - Implementing new features is relatively easy as they can be added to the already existing events
- Cons:
  - More complex to implement due to event dependencies
  - Requires more management to prevent memory leaks
  - Troubleshooting/debugging can be more difficult
  - The same event might be created multiple times, causing confusing errors that are hard to clean up

Software Design Components

- Combat Scene
  - The combat scene takes information from the overworld scene 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
  - Consists of various textures, sprites, and colliders
  - The player/user will traverse throughout the overworld, where they may encounter various items and enemies
  - Encounters with items and enemies may lead to events that trigger the combat scene or engage the player component
- 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
        - Additionally, it holds the various items that the player is carrying and can equip
    - AI Class
        - The AI carries all data and scripts associated with the human follower
        - In the Overworld, it supports the following ability that the human has with the user-controlled robot
        - During combat, the AI class contains data on the possible fighting actions and the human's health/statistics
    - UI
        - Packages include TextMeshPro, and it is used in all scenes
        - It assists the other components to show stats, menu options, inventory menu, and combat options

Coding guidelines

Find detailed coding guidelines at the following link:
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

Risk assessment

The top five risks that we believe may interfere with the completion of our project include inexperience with technology, the time consumption of art creation, scheduling conflicts, time management, and scope creep. A more detailed explanation of each risk can be found below.

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.

Project schedule

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 ● Apply stats that will affect abilities
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
Menus	● Create scenes for the menus ● Add buttons for playing, controls, and quitting ● Determine the button controls that will be used in the game ○ Fill out the controls page with correct button functions ● Add functionality to make each button go to the correct scene ● Connect the menu with the rest of the program

Test plan & bugs

We will test all of our use cases as previously outlined. These were made to encapsulate 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.

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
- Pickup Item
    - Input collider triggered between player and item
    - Assert item appears in player inventory
- Start game
    - Input: press on “play” button while on home screen
    - Assert that the user is moved into the scene 1 map
- Encounter an enemy
    - Input: the player runs into an enemy and selects “start”
    - Assert the user is moved to the combat scene with the correct enemy they encountered
- Fighting an enemy
    - Input: the user selects an attack on the enemy during the combat scene
    - Assert that an attack was performed with damage being dealt and the on-screen text changing
- Winning a fight
    - Input: the user successfully defeats the enemy
    - Assert that the user is brought back to the map with the defeated enemy disappearing from the map
- Losing a fight
    - Input: the user’s human dies in combat
    - Assert the user is sent back to the main menu as they have lost the game

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.

Bug Detection

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. There is no specific template that will be used, but the developers will be able to clearly understand the bug taking place.

Documentation plan

For our game, we will develop a brief player guide to inform the player about the mechanics of the game, including how combat works, how to use items, etc. These will both be written once a playable version of the game is ready.

Developer Guides – Provide instructions on how to extend or modify the software, including guidelines for contributing. This will also include details on installing, configuring, and building the software. Additionally, it will provide an overview of the architecture used for the project.

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
Joseph Liefeld

1. The biggest lesson I learned is that setting realistic goals is essential for being able to use your time effectively. Being too ambitious can lead to unintentional stress and unfulfilled goals.
2. One aspect that worked extremely well for our group was using Discord to communicate. Given that it was a software we were all familiar with, it allowed us quickly get started on planning without having to worry about establishing modes of contact.
3. If given the opportunity to redo this project, the main thing I would change would be the scope/scale of the project. Since this was the first game any of us had made, we were unaware of how much trial and error we would be doing. Because of that, several of our ideas were left out of the final build, toned down in scale, or scrapped early in development.

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.

   What worked well: One thing that worked well was splitting up into smaller teams to focus on specific components of the project. It allowed for a faster implementation of the game. Also, it provided everyone an opportunity to

contribute in their own ways and gave each of us a little freedom while still adhering to the requirements. Another thing that I found to work well was setting deadlines as a group for when certain parts of the project needed to be completed. It kept everyone on track and forced us to communicate with one another often.

What could have been different: While many positives came out of the project, one thing that I would do differently is using a different game engine to develop the program. While Unity worked well, there was a big learning curve that took some time to figure out. Using a different game editor/engine that more members are familiar with could speed up the development process and save more time for other areas of the project.

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

1. The biggest lesson that I learned from this project is not to reach for the stars. When planning out a project, always start small and build off a smaller goal. Beginning with a large goal can cause a lot of implementation issues especially when the group lacks clear week to week requirements on what needs to be done.
2. The thing that worked exceptionally well was assigning roles and an early common mode of communication. When being given specific smaller goals each week, work became more manageable and there was more work completed each week since there was little to no overlap in the work being done. This was easily communicated through the use of discord and trello so that each person had a clear idea of what was done, what needed to be done, and when a group member could meet to discuss or plan further steps of the project.
3. Something that I would change about the project is the frequency of testing features. The beta testing period that we conducted during class proved to be invaluable in giving information on what needed to be improved on and fixed. If

we had multiple testing periods I think that more bugs could be found earlier and the game could be refined even further.

Owen Wickman

1. I learned the real danger of overestimating what is realistically achievable within the timeframe. We started off the project with many features and content that we planned on implementing, but as time went on, had to be cut before the final implementation, leading to a version which deviated greatly from our initial vision of the project.
2. I found that what worked best was using Discord and establishing a standard for frequently checking in with each other. Through this practice, we made sure everyone knew what their roles were for the current sprint, efficiently communicated issues that needed urgent attention, as well as facilitated planning our team meetings.
3. In hindsight, if I were to do this project again I would do several things differently. I would