Project: Procedurally-generated 2D Role-playing Game

Marcos Sanson, Jerod Muilenburg, William Krol, Brendon Do, Ely Miller

CIS 350: Winter 2024

Table of Contents

Project Information

This project consists of the development of a 2D role-playing game (RPG). Our team consists of 5 people. In the game, players will assume the role of a customizable character navigating through diverse procedurally-generated terrains filled with interactive objects and water bodies. The game will feature a storyline with encounters, random events, and quest-driven gameplay, allowing players to shape their own adventure. Players will engage in combat with various creatures and monsters inhabiting the game world, utilizing their character's abilities and equipment upgrades. The game will provide regular updates and patches to enhance gameplay mechanics and address any issues.

Features

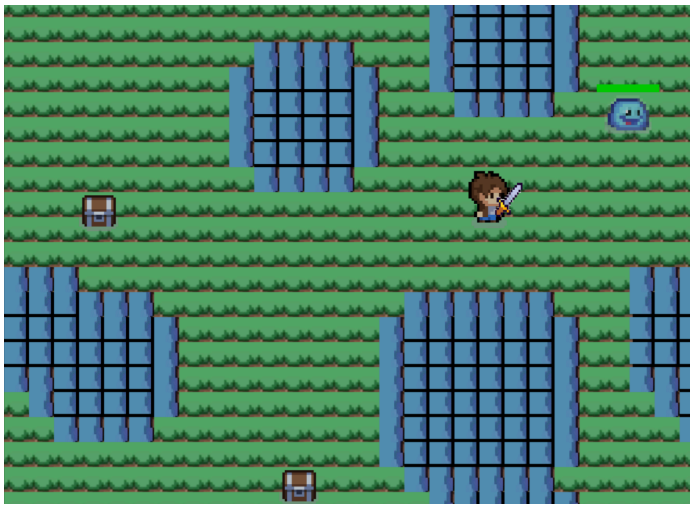
Has the following features for release 1:

- Character movement with arrow keys or WASD
- Collision detection with water and objects.
- Randomly generated terrain with clusters of water.
- Randomly generated objects with collision detection.
- Randomly generated enemies with collision detection, health systems, and defeat animations.

Additional features for the final release:

- Proper weapon combat with enemies
- Cooldown between attacks
- Different weapon types
- Different weapon damages
- Different weapon ranges
- Images for weapons
- Health Bar for enemies

Screenshots



Project Plan

Our project plan follows the waterfall model with a prototype as customization to better address iterative development needs. We initially focused on gathering requirements and designing the game mechanics, including character customization, terrain generation, and object interactions. Then, we started with developing a prototype of the game using Python and game development packages/tools/resources. Testing and verification was then conducted later in the process (in accordance with the waterfall model) to ensure functionality and identify any bugs or issues. Through testing, we verified that the game meets all requirements and is ready for release. Finally, we'll deliver the finished product to players and provide ongoing support and updates as needed to enhance the gaming experience.

Requirements & Definition

In the requirements and definition phase, we brainstormed ideas to come up with user needs and project scope. We identified key requirements, such as character customization, interactive terrain elements, and combat mechanics. These requirements were put into the use case diagram we have further down in this document. Following the document is our traceability matrix covering the cross of functions and the requirements they accomplish. Additionally, we created physical drawings and outlines to visualize and refine these concepts. This phase laid the foundation for the development process, guiding the next stages of design and implementation.

Development

For the development and implementation phase, we used Python as our primary programming language, along with relevant packages such as Pygame for game development. We divided the project into manageable tasks and assigned them to team members based on their expertise and interests. Throughout this phase, we followed coding standards (e.g., docstrings) and best practices to ensure code quality and maintainability. Regular code reviews and testing were conducted to identify and address any issues promptly. Collaboration tools such as version control systems (Git) were used to manage collaboration between team members.

Verification

For verification, we will put the project under 4 different verification methods. The first is integration testing. This is where we test the code as a whole to make sure that the code runs and is free of bugs. Next is the unit testing, where we make tests to make sure parts of the code runs well. After that is the line coverage testing, where we see how much of the program is being tested from the tests written for unit testing. Lastly, there is requirement coverage, where we see if the program meets the requirements we set before starting to program.

Maintenance

For maintenance, we will address any problems found by the professor or during updates for the next release. We'll prioritize fixing any bugs or issues reported by users to ensure the game operates smoothly. Additionally, we'll regularly update the game with new features and improvements based on feedback from players. We will also fix any issues identified by individual members of our group, if they discover problems. For tracking the issues we are internally discussing any issues that arise in our discord so we all can prioritize the fixing process. We will also look at the github issues page for any bugs that are reported.

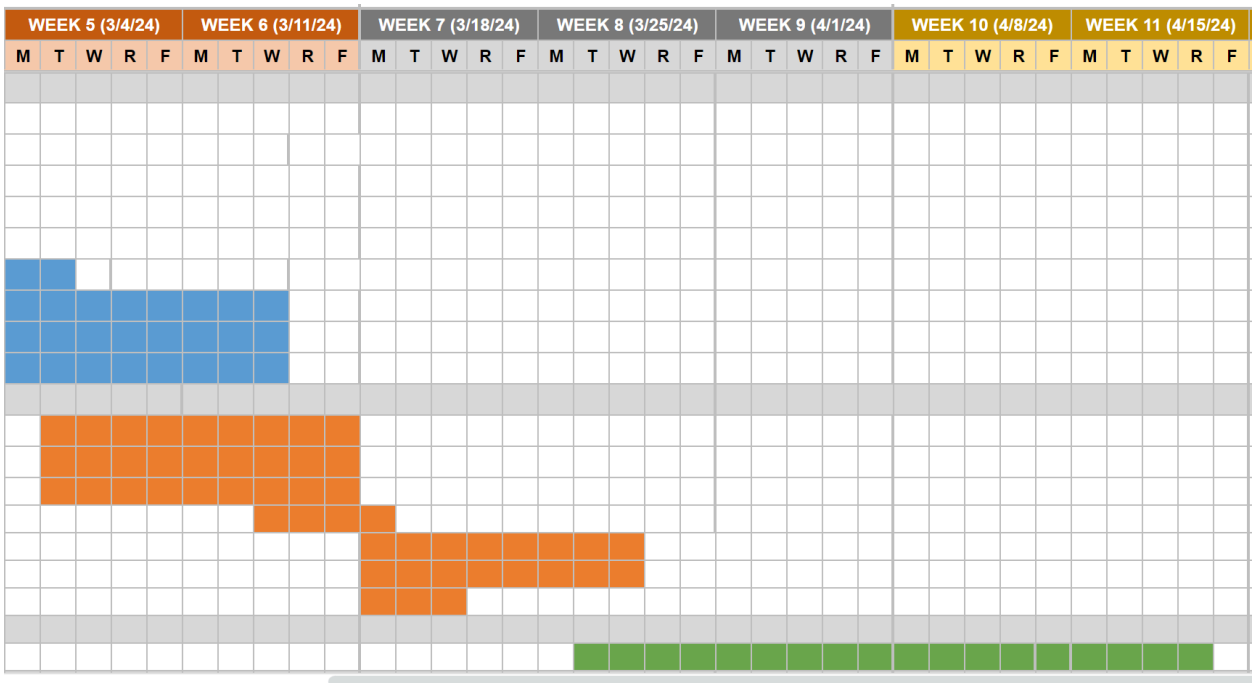
Umbrella Activities

For project management and status tracking, we will hold regular meetings to discuss progress, address any issues, and make decisions regarding the project's direction. We also constructed a Gantt Chart to help keep track of tasks needed to be done. We currently meet and talk over Discord every week. Meetings involve all team members to ensure everyone is informed and aligned. During these meetings, we review the current status of tasks, identify any obstacles, and determine the next steps to keep the project on track. This will help us to manage the project efficiently and successfully complete the project.

PROJECT TITLE	2D Procedural Generation RPG	GROUP NAME	Group 4
GROUP MEMBER NAMES	Brendon Do, Marcos Sanson, Ely Miller, William Krol, Jerod Muilenburg	DATE	3/10/24

Week 1 Starts at 2/5/24

TASK ID	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION	DEPENCENCIES	PCT OF TASK COMPLETE	WEEK 1 (2/5/24)					WEEK 2 (2/12/24)					WEEK 3 (2/19/24)					WEEK 4 (2/26/24)				
								M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F
1	Basic																										
1.1	Character Movement	Marcos S	2/5/2024	3/12/2024	5	None	100%																				
1.1.1	Combat System	Brendon D	2/12/2024	2/17/2024	5	1.1	100%																				
1.2	HP System	Brendon D	2/12/2024	2/17/2024	5	1.1.1	100%																				
1.3	Restart	Ely M	2/15/2024	2/21/2024	5	1	50%																				
1.4	Barrier/Obstacle Blocking	Marcos S	2/22/2024	2/27/2024	4	1.1	100%																				
1.5	Enemy Movement	Marcos S	2/26/2024	3/6/2024	7	1.1	20%																				
1.6	Loading/Unloading Rooms	Jerod/Marcos	2/27/2024	3/13/2024	12	1.1	100%																				
1.7	Chest/Door Interaction	Jerod M	3/4/2024	3/13/2024	8	1.8	10%																				
1.8	Inventory System	William K	3/4/2024	3/13/2024	8	1.1.1	100%																				
2	Making things look nicer																										
2.1	Adding different enviroment designs	William and Ely	3/1/2024	3/15/2024	9	None	50%																				
2.2	Item and Stat	Jerod	3/1/2024	3/15/2024	9	None	100%																				
2.3	Enemy type and stats	Jerod/Brendon	3/1/2024	3/15/2024	9	1.1.1	30%																				
2.4	Displaying HP	Brendon	3/14/2024	3/18/2024	4	1.2	100%																				
2.5	Coding items	Brendon/Jerod/Ely	3/20/2024	3/27/2024	8	1.8, 2.2	100%																				
2.6	Coding Enviroments	Marcos/William	3/20/2024	3/27/2024	8	2	0%																				
2.7	Updating Combat	Brendon	3/21/2024	3/25/2024	3	1.1, 2.5	100%																				
3	Verification																										
3.1	Verification	Everyone	3/26/2024	4/11/2024	13	1.1 - 2.7	100%																				



Responsibilities

Brendon Do:

- First and Final Release: Developing HP and Combat system

Marcos Sanson:

- First and Final Release: Developing basic movement and terrain, enemy, and object procedural generation algorithms

Jerod Muilenburg:

- First and Final Release: Developing weapons, enemies, NPCs, and all their associated stats/sprites/animations, along with descriptions and use with procedural generation

William Krol:

- First Release: Creating new, initial terrain features, inventory system and help with procedural generation
- Final Release: Finalizing inventory system, weapon creation and integration including attributes of weapons.

Ely Miller:

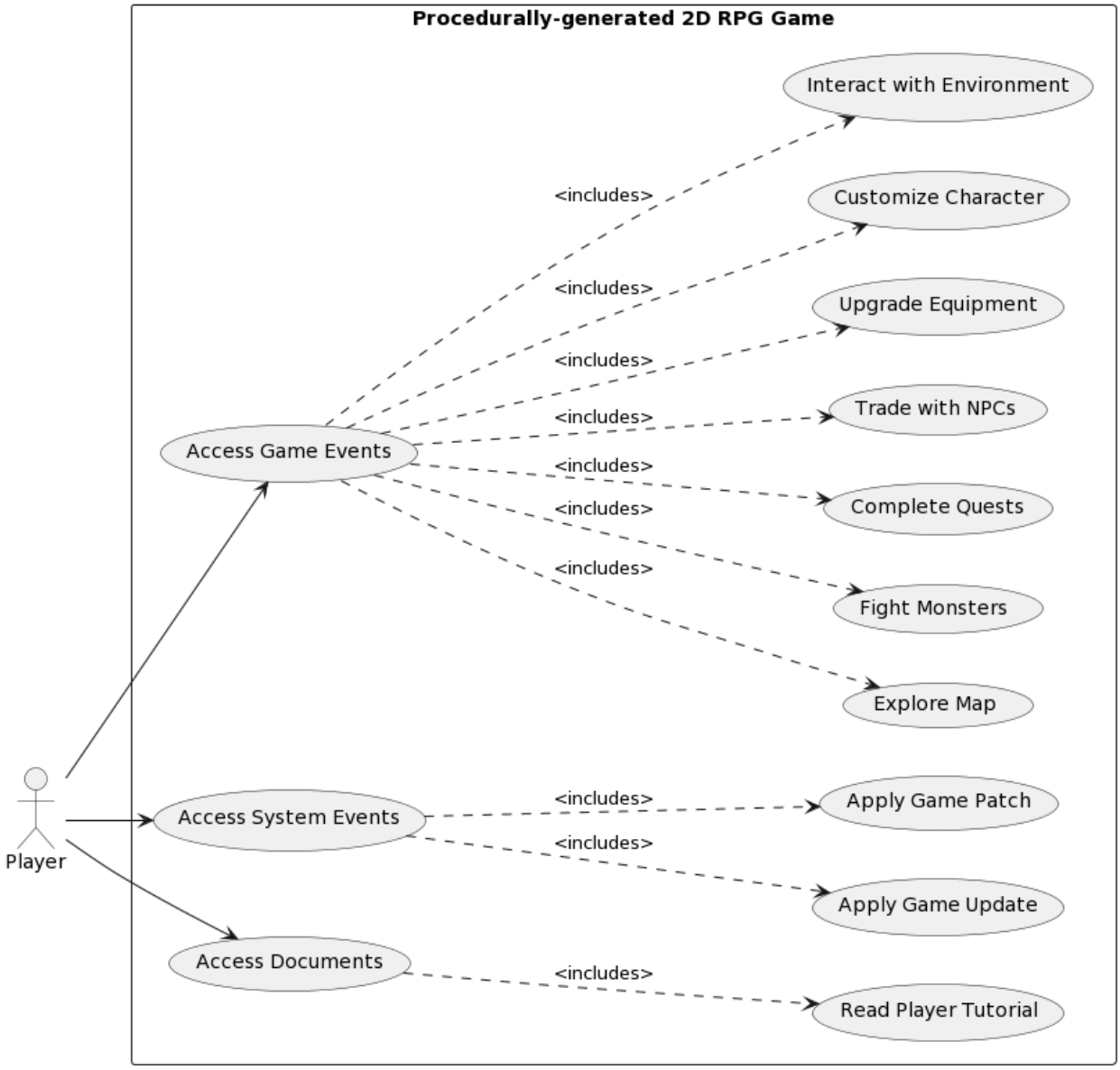
- First and Final Release: Designing different room environments and obstacles to be procedurally generated, making visual improvements and splitting code into classes/functions.

We break down the work by each team member's interests, and communicate over Discord and GitHub to collaborate effectively and ensure our code works with each other's designs.

Requirements & Specification

We used multiple different methods in our requirements and specification, including a use-case diagram, use case description, natural language requirements, and a traceability matrix.

Use-Cases



Use Case Descriptions:

Use Case: Access Game Events
Actors: Player
Description:
1.) The Player experiences various in-game events that affect gameplay, progression, and narrative.
2.) These events may include scripted encounters, random encounters, and story-driven plot developments.
3.) The Player's interactions and decisions during game events influence their gameplay experience and shape the outcome of the game.
Pre-Condition: Player must have started the game.

Use Case: Access System Events
Actors: Player
Description:
1.) The Player receives notifications and updates regarding system-related events and processes within the game.
2.) These events may include game updates and patches.
3.) The Player's awareness of system events ensures they stay informed about updates.
Pre-Condition: Player must have started the game.

Use Case: Access Documents
Actors: Player
Description:
1.) The Player accesses various documents and resources provided within the game for reference and information.
2.) These documents may include player tutorials, game manuals, and in-game help guides.
3.) The Player can read documents to gain insights, strategies, and background information about the game world and its mechanics.
Pre-Condition: Player must have started the game.

Use Case: Interact with Environment
Actors: Player
Description:
1.) The Player interacts with various elements and objects within the game environment, including interactive scenery, environmental puzzles, hidden secrets, and interactive objects.
2.) The Player's can discover hidden treasures, unlocking shortcuts, or triggering events that affect gameplay.
3.) The Player can interact with NPCs, objects, and elements using contextual actions such as talking, examining, or using items.
Pre-Condition: Player must have started the game and accessed game events.

Use Case: Customize Character
Actors: Player
Description:
1.) The Player personalizes and customizes their character's appearance, attributes, and abilities according to their preferences.
2.) The Player selects character classes, modifies physical features, chooses skills and talents, and allocates stat points to customize their character.
3.) The Player may unlock additional customization options through gameplay achievements or progression.
Pre-Condition: Player must have started the game and accessed game events.

Use Case: Upgrade Equipment
Actors: Player
Description:
1.) The Player enhances their character's equipment, weapons, and gear to improve combat effectiveness and performance.
2.) The Player upgrades weapons, armor, accessories, and other items using in-game currency, materials, or resources.
3.) The Player may need to meet certain requirements or fulfill specific conditions to perform equipment upgrades.
Pre-Condition: Player must have started the game and accessed game events.

Use Case: Trade with NPCs
Actors: Player
Description:
1.) The Player interacts with non-player characters (NPCs) to engage in buying, selling, and trading items, equipment, and resources.
2.) The Player can acquire valuable goods, obtain rare items, or sell surplus inventory to NPCs encountered throughout the game world.
3.) The Player can negotiate prices with NPCs during trade interactions.
Pre-Condition: Player must have started the game and accessed game events.

Use Case: Complete Quests
Actors: Player
Description:
1.) The Player starts various quests and missions offered within the game.
2.) These quests may involve tasks such as defeating specific enemies, retrieving items, escorting NPCs, or exploring hidden areas.
3.) The Player earns rewards, experience points, and progresses the storyline by completing quests.
Pre-Condition: Player must have started the game and accessed game events.

Use Case: Fight Monsters
Actors: Player
Description:
1.) The Player engages in combat with hostile creatures and monsters inhabiting the game world.
2.) The Player utilizes their character's skills, abilities, and equipment to defeat enemies and progress through the game.
3.) The Player may use items such as potions or buffs during combat to enhance their performance.
Pre-Condition: Player must have started the game and accessed game events.

Use Case: Explore Map
Actors: Player
Description:
1.) The Player navigates through different areas of the game world, discovering new locations, landmarks, and points of interest.
2.) The Player encounters various terrains, biomes, and environmental elements during exploration.
3.) The Player may encounter non-player characters (NPCs), monsters, and other entities while exploring.
Pre-Condition: Player must have started the game and accessed game events.

Use Case: Apply Game Patch
Actors: Player
Description:
1.) The Player receives notification of an available game patch.
2.) The Player initiates the patching process from within the game menu.
3.) The game downloads and installs the patch, addressing specific issues, bugs, or balancing adjustments.
Pre-Condition: Player must have started the game and accessed system events.

Use Case: Apply Game Update
Actors: Player
Description:
1.) The Player receives notification of an available game update.
2.) The Player initiates the update process from within the game menu.
3.) The game downloads and installs the update, applying changes to the game environment, mechanics, and content.
Pre-Condition: Player must have started the game and accessed system events.

Use Case: Read Player Tutorial
Actors: Player
Description:

1.) The Player accesses the tutorial section from the game menu.
2.) The Player navigates through tutorial topics covering game controls, mechanics, and features.
3.) The Player gains knowledge and understanding of gameplay concepts and strategies through tutorial instructions.
Pre-Condition: Player must have started the game and accessed Documents.

Natural Language Requirements

Natural Language Requirements:

Req1: The game system shall present in-game events to the Player (including scripted encounters, random encounters, and story-driven plot developments) when the Player is actively engaged in gameplay.
Req2: The game system shall allow the Player to interact with in-game events when the events occur within the Player's current game session.
Req3: The game system shall ensure that the Player's decisions during game events impact their gameplay experience during the current playthrough.
Req4: The game system shall notify the Player of system-related events and processes within the game when the Player is logged into the game.
Req5: The game system shall provide the Player with updates to ensure they stay informed about system events when the Player is logged into the game.
Req6: The game system shall grant access to various documents and resources within the game for reference and information when the Player accesses the game's menu or help section.
Req7: The game system shall include player tutorials, game manuals, and in-game help guides among the accessible documents when the Player accesses the game's menu or help section.
Req8: The game system shall allow the Player to read documents to gain insights, strategies, and background information about the game world and its mechanics when the Player accesses the game's menu or help section.
Req9: The game system shall enable the Player to interact with various elements and objects within the game environment when the Player explores the game world.
Req10: The game system shall allow the Player to customize their character's appearance, attributes, and abilities when the Player accesses the character customization menu.
Req11: The game system shall enable the Player to upgrade their character's equipment, weapons, and gear using in-game currency, materials, or resources when the Player accesses the equipment upgrade menu.
Req12: The game system shall facilitate trading between the Player and non-player characters (NPCs) for items, equipment, and resources, allowing buying, selling, and trading interactions when the Player interacts with NPCs within the game world.
Req13: The game system shall allow the Player to start various quests and missions offered within the game when the Player interacts with quest NPCs or quest boards within the game world.

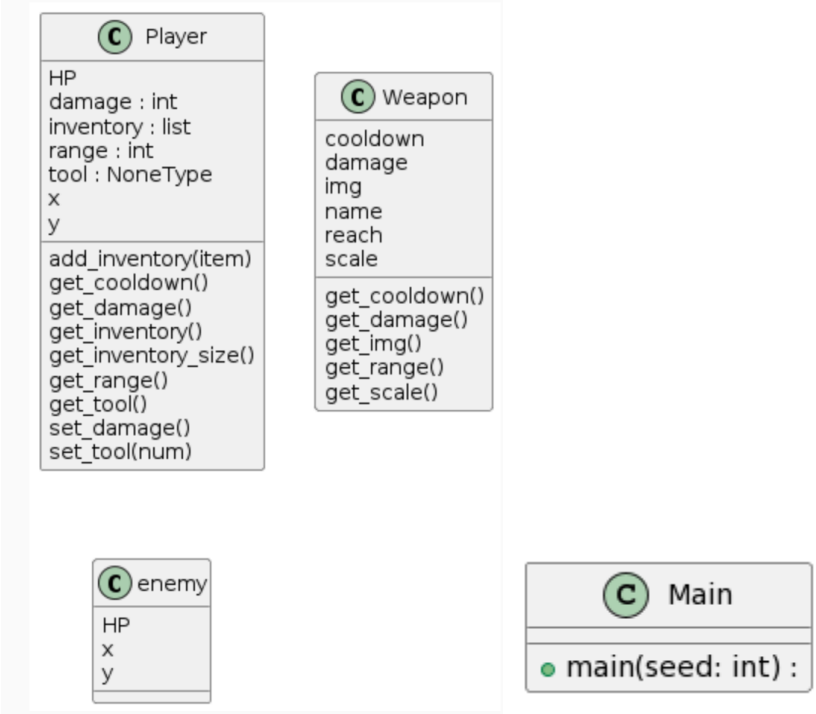
Req14: The game system shall reward the Player with experience points, rewards, and storyline progression upon completing quests when the Player completes quest objectives.
Req15: The game system shall enable the Player to engage in combat with hostile creatures and monsters inhabiting the game world when the Player encounters these enemies during exploration.
Req16: The game system shall provide the Player with the ability to explore different areas of the game world, discovering new locations, landmarks, and points of interest when the Player moves through different regions within the game world.
Req17: The game system shall notify the Player of available game patches and enable the Player to initiate the patching process from within the game menu when the Player is logged into the game and a patch is available.
Req18: The game system shall download and install game patches, addressing specific issues, bugs, or balancing adjustments when the Player initiates the patching process.
Req19: The game system shall notify the Player of available game updates and enable the Player to initiate the update process from within the game menu when the Player is logged into the game and an update is available.
Req20: The game system shall download and install game updates, applying changes to the game environment, mechanics, and content when the Player initiates the update process.
Req21: The game system shall provide the Player with access to player tutorials from the game menu, covering game controls, mechanics, and features when the Player accesses the tutorial section from the game menu.
Req22: The game system shall enable the Player to navigate through tutorial topics to gain knowledge and understanding of gameplay concepts and strategies when the Player accesses tutorial topics from the game menu.

Traceability Matrix

Use Cases / Requirements	Req1	Req2	Req3	Req4	Req5	Req6	Req7	Req8	Req9	Req10	Req11	Req12	Req13	Req14	Req15	Req16	Req17	Req18	Req19	Req20	Req21	Req22
Access Game Events	X	X	X																			
Access System Events				X	X																	
Access Documents						X	X	X														
Interact with Environment									X	X	X											
Customize Character										X	X											
Upgrade Equipment											X	X										
Trade with NPCs													X	X								
Complete Quests															X	X						
Fight Monsters																	X	X				
Explore Map																			X	X		
Apply Game Patch																		X	X			
Apply Game Update																				X	X	
Read Player Tutorial																						X

Design

In our current code, we use multiple functions and classes. Player, Weapon, and enemy are all classes. Main is a function.
Shown below is a UML diagram of these classes and functions:



Class diagrams and class descriptions and function decomposition and descriptions:

Function main(seed: int): This function initializes the game environment, including setting up the Pygame display, loading game assets (such as character images, terrain tiles, and enemy sprites), and generating the game world (terrain with water clusters and object/enemy positions). It then enters a continuous loop where it handles player input, updates game state (including character movement and collision detection), and renders the game elements on the screen. The function also manages events such as window resizing and quitting.

Class Player (HP, x, y): The Player class has HP for hit points and X and Y coordinates for the player's position on the map. Other variables not passed when calling the class are; damage, inventory, range, and tool. Damage is the number of hit points the player deals per attack which is set by the current weapon on hand. get_damage() returns the player's damage number. set_damage() will set the player's damage from the weapon's damage value. Inventory is a list that holds all items/weapons for the player. add_inventory(item) will add an item or weapon class to the player's inventory. get_inventory() will return the inventory to find out what the player has. get_inventory_size() is specifically used to return the size of the list. This is used to

determine if the player can select item 4 if the player has less or more than 4 items. Range is the distance between the player and enemy which can deal damage. get_range() will return the range of the weapon that the player has equipped. Tool is the item or weapon currently equipped by the player. get_tool() will return the item the player has equipped. set_tool(num) will set the currently equipped tool to whatever item is passed as a number for the list.

Class Weapon (name, damage, reach, cooldown, img, scale): This class holds information regarding the weapons the player has at their disposal. Name isn't currently used in any part of the code but is present for future development where someone can sort by name and display it on a menu. The damage corresponds to the damage the weapon will do to enemies. get_damage() will return the current damage value. Reach is used to determine how close the player has to be when they attempt to attack an enemy. get_range() is used to return the range. Cooldown is a set time where the player can not attack after successfully hitting an enemy. This is used to prevent 'spam' attacks making the enemies way too easy to defeat. get_cooldown() retrieves the cooldown amount for the weapon. Img is the image file used to represent the weapon on the screen in game. get_img() grabs the name of the image file and returns it. Finally, Scale is used to determine how big the image will be when it is drawn on the player. get_scale() returns the number to scale the image for the weapon. All of these variables are designed to be customizable depending on what was input to the class call.

Class enemy (HP, x, y): This class will house the basic enemy. HP is the hitpoint of the enemy. X and Y are the positions of the enemy on the screen. As it stands currently, we don't need any other variable for our enemies but this can be easily expanded upon to include more for future development.

Development

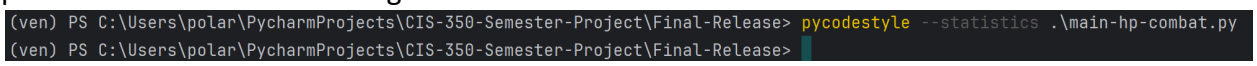
For this project, we are primarily relying on the Pygame library as it is one of the easiest and biggest game libraries that allow us to do what we want to do for this project. Pygame is used extensively in our code to handle multiple aspects of game development, such as creating the game window, loading and displaying images, handling user input (keyboard events), managing the game loop, and rendering game elements on the screen. It specifically provides functionalities like loading image assets (pygame.image.load()), drawing images on the screen (screen.blit()), handling keyboard events (pygame.key.get_pressed()), and updating the display (pygame.display.update()).

We also use Asyncio. which is a Python library used for asynchronous programming. Asyncio is used to run the main game loop asynchronously (our main method, the async def main() function, runs asynchronously). The “await asyncio.sleep(0)” statement inside the game loop ensures that everything runs concurrently, which maintains smooth gameplay without blocking the event loop.

We also use the random module in Python, which provides functions for generating pseudo-random numbers. This is the basis of our procedural generation algorithms. The random module is used to seed the random number generator (random.seed(seed)) at the beginning of the program to ensure reproducibility of random sequences, similar to other video games or procedural generation algorithms. It’s also used throughout our code to generate random values for various game elements, such as the number of clusters of water, the size and position of terrain clusters, the number and position of objects, and the number of enemies.

Code Standards

Currently, the code does not pop up with any errors when run with pycodestyle –statistics. Most errors from the previous release were that the lines were too long, though that was fixed by breaking arguments into different lines of code. This was a significant improvement from the previous release. See following screenshots:



```
def load_weapon():
    scale = p1.get_tool().get_scale()
    weapon_img = pygame.image.load(p1.get_tool().get_img()).convert_alpha()
    weapon_img = pygame.transform.scale(weapon_img, size: (scale, scale))
    weapon_rect = weapon_img.get_rect()
    return weapon_img, weapon_rect

weapon_slash_imgs = [pygame.transform.scale(
    pygame.image.load('slash.png').convert_alpha(), size: (40, 40)),
                     pygame.transform.scale(
                         pygame.image.load('slash.png').
                         convert_alpha(), size: (30, 30))]
```

```
# Generate random positions for objects
num_objects = random.randint( a: 1,  b: 4)  # Random number of objects
object_positions = []
object_collided = []  # Track collision status for each object
for _ in range(num_objects):
    object_x = random.randint( a: 0, len(terrain_grid[0]) - 1) * tile_size[0]
    object_y = random.randint( a: 0, len(terrain_grid) - 1) * tile_size[1]
    object_positions.append((object_x, object_y))
    # Initialize collision status as False for each object
    object_collided.append(False)
```

```
def attack_animation():
    attack_index = (pygame.time.get_ticks() // 200) % len(
        weapon_slash_imgs)
    print("'slash'")
    slash_imgs = pygame.transform.scale(weapon_slash_imgs[attack_index],
                                        size: (60, 60))
    screen.blit(slash_imgs, r)
```

Static Analysis

This is the result from running Pylint to see if there are any static analysis errors.

main-hp-combat.py:117:50: E1101: Module 'pygame' has no 'RESIZABLE' member (no-member)
main-hp-combat.py:249:29: E1101: Module 'pygame' has no 'QUIT' member (no-member)
main-hp-combat.py:252:31: E1101: Module 'pygame' has no 'VIDEORESIZE' member (no-member)
main-hp-combat.py:255:40: E1101: Module 'pygame' has no 'RESIZABLE' member (no-member)
main-hp-combat.py:262:16: E1101: Module 'pygame' has no 'K_LEFT' member (no-member)
main-hp-combat.py:262:39: E1101: Module 'pygame' has no 'K_a' member (no-member)
main-hp-combat.py:265:16: E1101: Module 'pygame' has no 'K_RIGHT' member (no-member)
main-hp-combat.py:265:40: E1101: Module 'pygame' has no 'K_d' member (no-member)
main-hp-combat.py:268:16: E1101: Module 'pygame' has no 'K_UP' member (no-member)
main-hp-combat.py:268:37: E1101: Module 'pygame' has no 'K_w' member (no-member)
main-hp-combat.py:270:16: E1101: Module 'pygame' has no 'K_DOWN' member (no-member)
main-hp-combat.py:270:39: E1101: Module 'pygame' has no 'K_s' member (no-member)
main-hp-combat.py:272:16: E1101: Module 'pygame' has no 'K_q' member (no-member)
main-hp-combat.py:274:16: E1101: Module 'pygame' has no 'K_z' member (no-member)
main-hp-combat.py:284:16: E1101: Module 'pygame' has no 'K_1' member (no-member)
main-hp-combat.py:287:16: E1101: Module 'pygame' has no 'K_2' member (no-member)
main-hp-combat.py:293:16: E1101: Module 'pygame' has no 'K_3' member (no-member)
main-hp-combat.py:299:16: E1101: Module 'pygame' has no 'K_4' member (no-member)

Pylint says there are errors with pygame not having those keyboard presses as a member, though the game controls still work when running the code. In real use and gameplay, this causes no errors or issues, as seen in our game demonstration video(s).

Code Documentation

Link to generated documentation report:
https://github.com/Marcos-Sanson/CIS-350-Semester-Project/blob/main/Final-Release/html/main-hp-combat.html
To view the report, download the HTML file to view the documentation generated from pdoc.
Or, view the following images (see next page):

▶ View Source

class Player: ▶ View Source

Player(HP, x, y) ▶ View Source

Holds the information about the player. Args: HP (int): The amount of health the player has. x (int): The horizontal position of the player. y (int): The vertical position of the player.

inventory

HP

tool

damage

range

x

y

def get_inventory(self): ▶ View Source

Gets the current inventory of the player. Returns: List

def add_inventory(self, item): ▶ View Source

Adds the argument into the inventory. Args: item (Weapon): Weapon to be added into the inventory.

Returns: None

def set_tool(self, num): ▶ View Source

Puts the item from the inventory into the hand. Args: num (int): The number from the argu

Returns:

def get_tool(self): ▶ View Source

Gets the tool in players hand. Returns: Weapon

def get_inventory_size(self): ▶ View Source

Gets the size of inventory. Returns: int

def set_damage(self): ▶ View Source

Sets the damage the player does to the weapon in hand. Returns: int

def get_damage(self): ▶ View Source

Gets the damage stat of the weapon in hand of player. Returns:

def get_cooldown(self): ▶ View Source

Gets the cooldown stat of the weapon in the hand of the player. Returns: int

def get_range(self): ▶ View Source

Gets the range stat of the weapon in the hand of the player. Returns: int

The Player class holds information about the player, being their location, inventory, range of the attacks, the attack damage and also the cooldown of the attack.

class enemy: ▶ View Source

enemy(HP, x, y) ▶ View Source

Holds the location and the amount of health the enemies have. Args: HP (int): The amount of health the enemy has. x (int): Is the horizontal position of the enemy. y (int): Is the vertical position of the enemy.

HP

x

y

The enemy class holds the HP and the location of the enemy in the map.

class Weapon: ▶ View Source

Weapon(name, damage, cooldown, reach, img, scale) ▶ View Source

Holds information for the weapons in the game. Args: name (string): Name of the weapon. damage (int): How much damage the weapon deals to enemies. cooldown (int): Amount of time before the weapon can be swung again. reach (int): How far the player can be to attack the enemy. img (string): The image of the weapon. scale (int): How to scale the weapon to the player model.

name

damage

cooldown

reach

img

scale

```
def get_damage(self):                                    ▶ View Source
```

Used to get the damage stat of a weapon. Returns: int

```
def get_cooldown(self):                                  ▶ View Source
```

Gets the cooldown state of the weapon. Returns: int

```
def get_range(self):                                     ▶ View Source
```

Gets the reach stat of the weapon. Returns: int

```
def get_img(self):                                       ▶ View Source
```

Gets the image of the weapon Returns: string

```
def get_scale(self):                                     ▶ View Source
```

Gets the scaling variable of the weapon for loading the image on screen. Returns: int

The weapon class has each weapon's damage, range and image, along with the cooldown between each attack.

```
async def main(seed):                                    ▶ View Source
```

And finally, the main function that runs the game.
(See link to report for more details and documentation).

Configuration Management

Our configuration management relies on branching in our GitHub repository. The way we are tracking releases is creating branches after the release that are not to be modified after the release.

Currently for the final release, we have a branch named Final-Release that shows the state of the Git repository during the final release.

Git log for the remote repository:
https://github.com/Marcos-Sanson/CIS-350-Semester-Project/commits/Final-Release

Verification

For the final release, the program was put under different methods of verification to check and see how the code runs.
These methods are Integration Tests, Unit Tests, Code Coverage and Requirements Coverage.

Integration Tests

The classes in the program work well with the main function we have to run to play the game. There are no issues with the classes interacting with each other and there are no issues when playing the game.

Unit Tests

Unit testing is done in the test.py file, containing basic tests of the functions and classes in the main file. The tests are meant to ensure every function works as intended, and test for some error cases where they may be likely to come up. Some examples include checking for player movement, enemy attacking/death, and basic game logic. All tests showed expected results. See file for full testing.

Code Coverage

With the exception of some of the main functions, the unit tests achieve full code coverage of the game's Player, Enemy, and Weapon classes and game logic. This includes line/branch coverage for all code that test.py can access. For all other aspects of the game's development, we had to playtest the game, since aspects like rendering and collision are much more difficult to test with code. Our code coverage for unit testing and integration testing all resulted in expected outcomes, and did not show any issues or errors.

Requirements Coverage

For our requirements coverage we compare what we have to what our traceability matrix was stated to have.

- We have included:
    - Req 9: The player can interact with enemies in the form of combat and can traverse rooms by going to the edge of a room.
    - Req 15: The player is able to deal damage to enemies by pressing the 'z' key and depending on what weapon they used, the damage, range and cooldown will differ.

    - Req 16: The player is able to maneuver around the randomly generated room and can traverse between rooms going from one room to another.
- We were not able to include (text following the requirement is what happens in result of not implementing the requirement):
    - Req 1: The system doesn't create and present in-game events like scripted encounters apart from the enemy and chest generation.
    - Req 2: The system doesn't allow the player to interact with said in-game events as there aren't any.
    - Req 3: The player's choices have little to no impact on the course of the game, currently as no choice making or scripted events occur.
    - Req 4: There is no notification or heads up of said in-game events as they do not exist.
    - Req 5: The game does not notify the player of any updates to the game as we have not reached the development phase where we could upload the game to an app store and update the game.
    - Req 6: The system has no collection of documents in any regards to the player or account information.
    - Req 7: There is no tutorial in place for newer players.
    - Req 8: There is no lore integrated into the game and no strategy to take away from documents.
    - Req 10: There is no character customization apart from their choice of weapon.
    - Req 11: There is no upgrade path for any weapons and no resources that can be used for said upgrades.
    - Req 12: The game does not contain trading with NPCs or any kind of trading at all.
    - Req 13: There are no quests of any kind implemented into the progression of the players run.
    - Req 14: The game does not integrate a reward system for defeating enemies, opening chests or completing quests/levels.
    - Req 17: There are no systems in place to notify of game patches and updates.
    - Req 18: There is no automatic update downloader and installer.
    - Req 19: There is no way of tracking updates for the user and notifying them.
    - Req 20: The game does not automatically install updates because there is no system in the code for updating the game.
    - Req 21: The game does not have a tutorial section that the player can access from a menu.
    - Req 22: The game does not allow the player to navigate through the tutorial because there is no tutorial.

This overall is a big loss for the project as most of what we set out to complete wasn't finished nor attempted in some areas. Like many projects, there are always unforeseeable issues and delays that we didn't account for when we initially made these requirements. We still met some requirements, however.

- Req 1-4 related to the in-game events were always going to come after we had the base functionality of our code done, our development of that base code took much longer than we expected which is why we had to leave out those events. Req 5,6,17-20 regarding system updates and account documentation are the final features to implement after the rest of our requirements have been met so that we could polish the game as if we were to release it to the public. Req 7,21,22 are about new player tutorials and menu navigation, which we could've prioritized to complete for our final release; we decided to focus on more of the gameplay aspect to have some variety in gameplay. Req 8 about lore would've been one of the last things implemented as it doesn't have any core functionality to our program and is more for mystery and fun. Req 10,11,12 refers to character customization and trading which relies on an effective and easy to use inventory system and menu, which we had trouble creating so we had to push the requirements back until we had a functional system; which we did achieve but not until the very end of our development time. Req 13 & 14 are about quests, rewards and progression in the game which was planned to be developed after the travel between rooms was established. The rewards could've been implemented from defeating enemies and opening chests but we figured it was better to focus on the rewards and quests all in one development phase rather than splitting the work over two periods, potentially having to rework the rewards anyways when we integrated quests.

Our eyes were set on a massive game with detailed and complicated requirements that we did not realize would take much more than the allotted time we had. We absolutely 'bit off more than we could chew' for this project. With further development, one could fulfill our initial vision; our commitment and time management was not fit to tackle the scope of what we set out to do.

Postmortem

Group:

- First: When we began we were excited by the possibilities our 2D procedurally generated game could accomplish. That sense of accomplishment led us to have many ideas for use cases and mechanics in the game that overall led to many tasks to be completed for our first prototype. This caused some of the tasks to be completed easily while others were more difficult resulting in those tasks not being completed by our first deadline. As a group, we had a more relaxed approach to the work assigned and now seeing the hill of work to be done, we now have a better understanding of how we are going to proceed going forward with our project. Overall, however, we have successfully developed a functional prototype with many of the systems we planned to have, including random generation for a variety of game aspects.
- Final: Most of our core code was completed or near completion by our first release, so the end goal was cleaning up our code and finalizing some of our missing features. These features being; an enemy health bar, inventory system, weapons (including attributes) and room progression. The room progression is not present in our presentation as the player had a chance of becoming stuck in an area when moving between rooms. This can be fixed by forcing terrain features to not generate near the edges. There are other features we picked out in the beginning but our initial idea has come to fruition, just on a more basic level.

Individual:

- William:
    - **First Release**: Going into this project I knew that this wouldn't be any easy task to finish in a couple days or so. That being said, looking back at my involvement with the project, I definitely see that I have not dedicated as much time as I would have liked to. I feel that my fellow group members have put more effort into their assignments than I have of my own. To them I have nothing but complements in their work. I am glad we have a working prototype to use as my biggest struggle is getting things off the ground. With a clear path forward I can be more resourceful and useful then I was prior. In doing so, I have seen how my choices are affecting the progress and I am determined to dedicate more of my available time to the project in any way that can further our progress.
    - **Final Release**: After our first release, I aimed to contribute more in the final release of the project and I did just that. I continued my work on figuring out a functional inventory system. A few weeks after the first release, I was still struggling to create a menu that the user could interact with. It was at this point that I decided to shift the inventory system to a 'hot-key' system where the buttons 1-4 can change what item the player is holding. This ended up being easier to code and allowed me to add additional functionality in other places; that being weapons. To test my code I had to add objects so I created the 4 weapons used in our final release. The integration was relatively easy and I was

able to give the weapons distinct attributes to make them more unique rather than carbon copies of each other. This progress was a great motivator and overall I am pleased with our final result even though it is far from what we set out to accomplish.

- Marcos:
    - **First Release:** I believe that the project has gone well so far. At this point, we have a functional game that takes user inputs smoothly and has legitimate procedural generation, which is the primary goal of this project. At this point, the game is already unique and varied with each seed entered by the user, which is fantastic. There are definitely more features to be added, but these build on our current features and game. The procedural generation algorithms function and they create decently realistic environments, although there are still bugs, such as being "trapped" in an area blocked off by water. Overall, while there were definitely more features I (and our group) could have implemented previously, overall the project has gone well and these features can all be reasonably added by the project deadline.
    - **Final Release:** Overall, I am happy with the project's final release. Not only did we accomplish our goal of creating a procedurally-generated RPG, but we also implemented additional features, including many of which we wanted to include since the previous release. We implemented a way to move between different maps or "rooms," with each map randomly generated. We also implemented a working inventory system (different weapons) and enemy health bar system. Both of these new additions were integrated well in this final release, and they not only function properly but they also visually match the look of the rest of the game. There are definitely more features we could have added, as outlined previously, but thanks to our game design, our current (and final) release works exceptionally well and accomplishes most of what we originally set out to do.
- Ely
    - **First Release:** I think we've made good headway on our project as a group, but in hindsight, I've fallen into bad work habits so far. Mostly, I felt that while I did some work on the project, including a task I was assigned to work on from the start, I didn't accomplish as much as I had set out to do. Part of this was uncertainty about the development process (GitHub, workflow, deciding on a design/structure for our codebase), but the lost time from not being sure what to do next could have been avoided sooner with more clear communication, which I've made a top priority for myself for the final release. By working more directly with my group and making sure the work is cut out clearly for me and the other team members, I think we can make quick progress on our goals from here on out.
    - **Final Release:** For what we aimed to complete by the time we finished the semester, I think we achieved a lot. The biggest objective that we failed to meet was the 'procedural generation' element, which I think ended up not getting fleshed out enough since we were focused on the basics. However, I think that

the game in its own right had most of the features we intended, and the product was playable. It did also miss some of the content we planned, which may have also been outside the scope of what we could do in this span of time. Also, investing more time into the project collaboratively could have improved the product, or possibly more time just planning in the beginning.

- Brendon
    - **First Release Review:** Progress on the project could be better, though I am proud of what we have done so far. While we are behind on the gantt chart, the tasks ahead can still be done in a timely manner. I feel that I am pulling my weight and doing what I am tasked to do within the time frame, though it could have been improved with animations and HP being shown to the player on the screen. Overall, I think we have made good progress on the project, and can finish it in the time remaining.
    - **Final Release Review:** I am both pleased and kinda disappointed with the final product. It is cool to see what we have done with the code currently, being able to show the work we have done to the class. I am kinda disappointed because we could have done more if we had used our time more wisely throughout the semester, but I am satisfied with what we had done under the time constraints we had put ourselves into. Overall, this project could have definitely been better under other circumstances, but I do like the end point we have here.
- Jerod
    - **First Release:** I think we made a lot of progress toward our final goal. We have a good structure and a working prototype that can be expanded to what we are looking for. I struggled with knowing what I should be working on or how to contribute and getting github to finally work correctly but I think I have a good idea of the path forward and my role in the project.
    - **Final Release:** The progress on the project really started to ramp up after the first release. We were able to get much more done and make it our own. The different improvements that each team member added really helped change how the game worked and gave it a different look and feel. Despite not having everything we wanted to add, I think it turned out great and achieved our goal of a randomly generated game RPG.

Earned Value

According to the gantt chart, we currently have 11 tasks completed, and for the final release, we wanted to have all 17 tasks completed. Between the first and final release, we decided to scrap some of the tasks to be able to finish the more important tasks for the project.

At the final release, we are behind what we had planned to do, making us have an earned value of below 1.

Variances

In addition to being behind schedule according to the Gantt chart, we have encountered some other technical challenges, such as debugging issues with the procedural generation algorithms. Like what was mentioned previously, we still need to solve some issues with areas being blocked by randomly-generated water, so that the gameplay isn't inhibited at all. This essentially will require adjusting our timeline to resolve these specific problems within the larger tasks that need to be completed.

For the final release, we had planned to have so much more completed and our setback from the first release only pushed development back, keeping us behind during the final release's development.

Lessons Learned

One lesson that some of us may have learned more than others is that any small, periodic, progress and work on the project really helps more than relying on big, multi-hour work periods, especially if we're getting close to the end. The constant small work allows you to see the problems or obstacles that will have to be overcome and be able to brainstorm ideas over the course of time rather than trying to come up with a solution all at once.

Another aspect we have taken away from is the importance of our communication. Apart from our initial meetings after class, we set up a Discord group chat to communicate, but that was used less than would be ideal in an industry setting (compared to using Slack channels, in-person meetings, etc.). Now closer to the end of the first prototype we have been utilizing it much more to effectively communicate our contributions and progress so that everyone is up to date in the loop and not having to wait till we meet in-person in class.

Looking back at the semester and the project, we had a poor sense of scope and what we were capable of accomplishing in our set time window. Now with that being said, we have created a product which functions but still needs development to reach what we had in mind. With how often projects fail or get pushed back in industry, what we created is common amongst others, not to say that every group was in the same boat. From the presentations we saw that a few groups quite the exceptional project compared to us. We took this as confidence that if given more time or another opportunity like this one, we could turn up a better product that what we had completed for now. This doesn't justify the state of our project but we all have taken important factors from this experience to improve our individual capabilities to contribute to a team's goal.

References

References:

DeVries, Byron. (n.d.). Original sprite movement code. (Retrieved February 6, 2024).
Game Endeavor. (n.d.). Mystic Woods downloadable asset pack [Sprites, images]. Retrieved from https://game-endeavor.itch.io/mystic-woods (Retrieved February 6, 2024).
ChatGPT. (February 13-23, 2024). Originally created use case diagrams, use case descriptions, and natural language requirements. Modified by Marcos Sanson (February 13-23, 2024).