Isle Of Heidren

Heidi Raber, Bella Brickler, Ren Paxson, Joe Claborn

Elevator Pitch - Inspiration

- Isle of Dread DnD Campaign (Fig.1)
- Oregon Trail Game (Fig. 2
- Take inspiration from the simplicity of pokemon combat

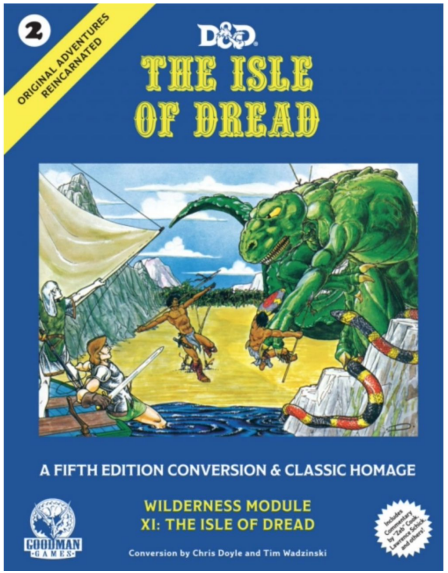

Figure 1: Isle of Dread

Figure 2: Oregon Trail

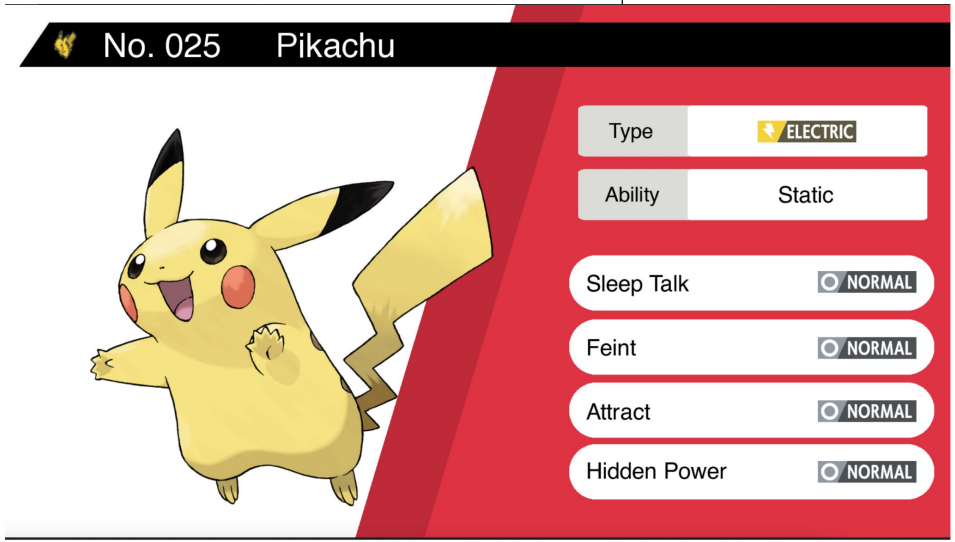

Figure 3: Pokemon moves

Elevator Pitch - Our Story

- Four friends stranded on an island
- Find notebook that tells of great treasure in the center of the island
- Fight monsters along the way
- A village filled with characters that players can interact with to find out information about the notebook, monsters, and the legendary City of Gods

User Stories / Components of Complexity

User Story	Component of Complexity
As a gamer, I want a text-based desktop adventure game with an interactable GUI so that I can make knowledgeable choices within the game.	Main GUI Window that shows current environment or battle map Panel showing current actions for game state Panel displaying all characters showing which is selected
As a gamer, I want to engage in complex monster and NPC battles because it is fun.	Simple monster and npc battle "AI"
As a gamer, I want the game to update efficiently without lag so that the game operates smoothly.	Generalized game framework using JSON objects for each NPC/Monster/Location/Battlemap
As a gamer, I want to play a game with a throughline so that the story make sense to me and I can follow it clearly.	One main quest with two side quests and events spurred by NPC interaction

Figure 4: Table showing the User Story vs our Component of Complexity

Software Development Plan - Administrivia

- Meetings two times a week, tuesday and friday
- Had breakout meetings and shorter meetings as needed to collaborate on sprint items
- Adherence to a meetings minute template

Software Development Plan - Methodology

- We did a agile system
- 1 week sprints starting on tuesdays each week, with the second meeting in the week acting as a longer stand-up
- Utilized github branches and pull requests to keep merge conflicts to a minimum
- No commiting to main unless working during meetings

Software Development Plan - Milestones

- Requirements documentation: 2/20
- Requirements Evaluation: 4/5
- Design documentation: 3/19
- Framework, GUI start game screen
- Main Game Frame
- Map
- Story - integrate and implement
- Simple Combat

Architecture - Overview

- Written in java
- Utilizes existing game framework LibGDX
- Game consists of a few LigGDX screens with UI components that change.
- UI built in similar way to HTML pages with each component being a child to a given parent

Architecture - Game Flow

- Enforce SOC by having multiple “controllers” handle separate aspects of the game
- Repository Design Pattern for game objects
- Main GUI passes references of implemented components to state managers

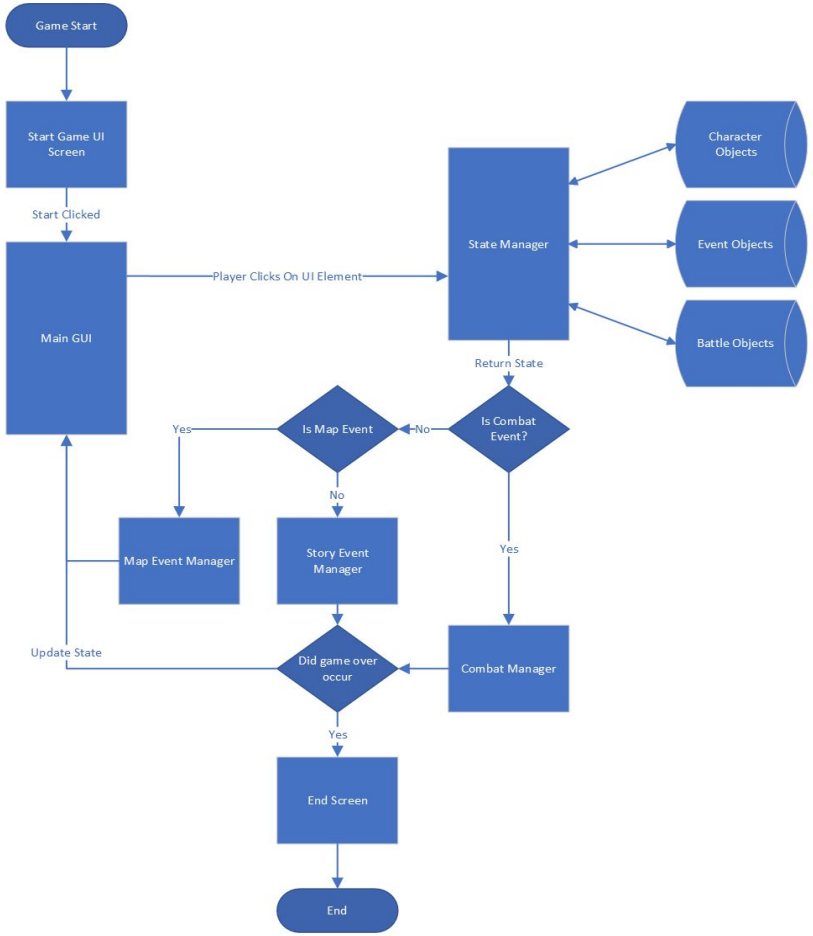

Figure 5: Overall Game Flowchart

Architecture - GUI Flow

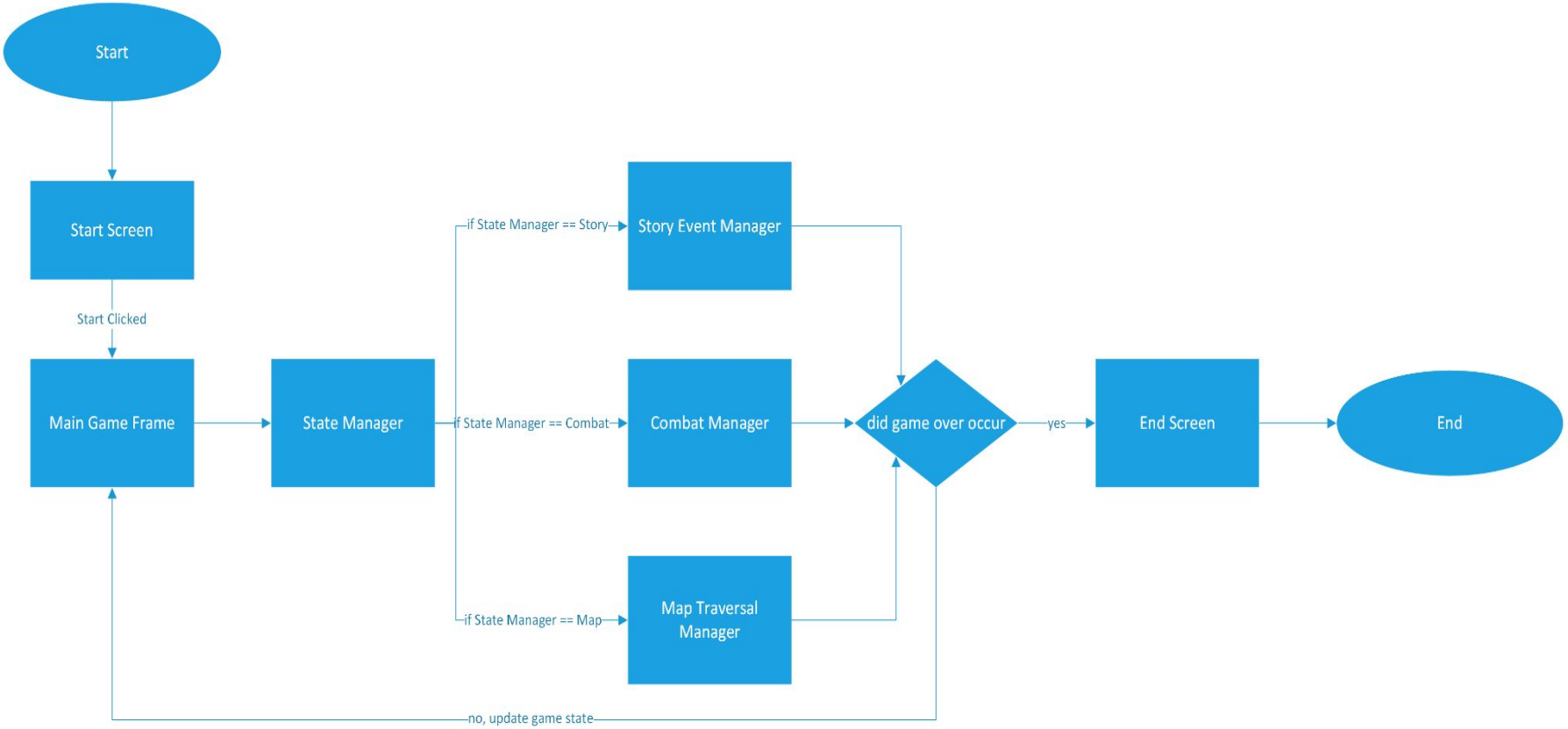

Figure 6: Flow chart of UI Components

Architecture - Repository model examples

- Generalized Interface for all game objects
- Each type gets a specific implementation

```java
public interface JSONRepo<T> {
    public T get(String id);
}
```

Figure 7: Repo Interface

```java
    public StoryEvent get(String id) {

        //Load file from directory
        String filePath = "data/events/" + id + ".json";
        FileHandle file = Gdx.files.internal(filePath);

        Gson gson = new Gson();

        StoryEvent se = null;
        try {

            //Parse json data as a combat event first
            se = gson.fromJson(file.reader(), CombatEvent.class);

            //If the monsters are null then its not a combat event
            if (((CombatEvent) se).getMonsters() == null){
                se = null;
            }
        } catch (Exception ex)      You, 3 days ago • adding combat
        {
            se = null;
        }

        //Parse the json as a story event
        if (se == null) {
            se = gson.fromJson(file.reader(), StoryEvent.class);
        }

        return se;
    }
```

Figure 8: Repo Implementatio

Architecture - Controller/Manager Example

- Controller gets copies of needed UI components
- Send updates back to the component that the UI thread updates

```
CombatController(ConsoleComponent console, StatPanel panel) {
    this.console = console;
    this.panel = panel;
}
```

Figure 9: Code Example of Constructor

```
console.appendMessage(message, MessageType.COMBAT_RESULT);

message = attackTarget.getName() + " has " + attackTarget.getHitPoints() + " remaining!";
console.appendMessage(message, MessageType.COMBAT_RESULT);
```

Figure 10: Code Example of Component Interaction

Requirement 10: Will be a desktop game

Implementation: runs in a desktop window

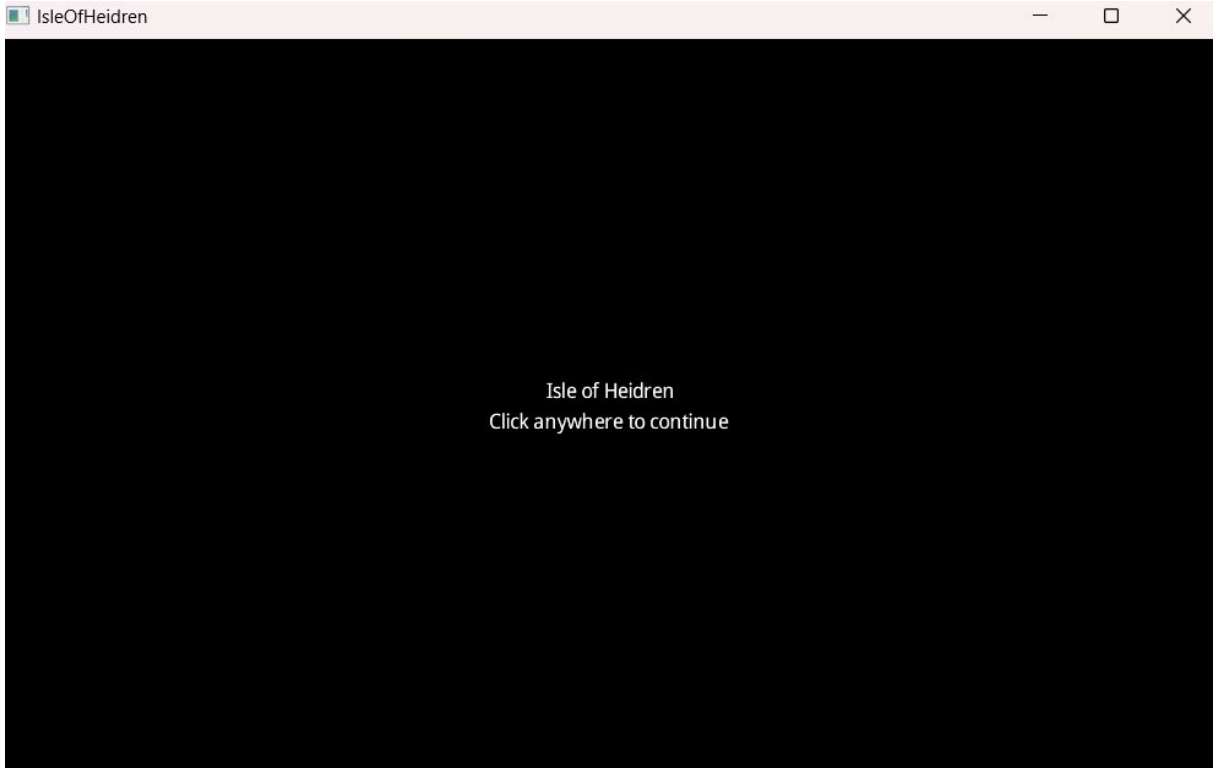

Figure 11: Start Screen

Requirement 20: Will be based on DnD

Implementation: The attacks listing in the buttons of this screenshot are DnD attacks

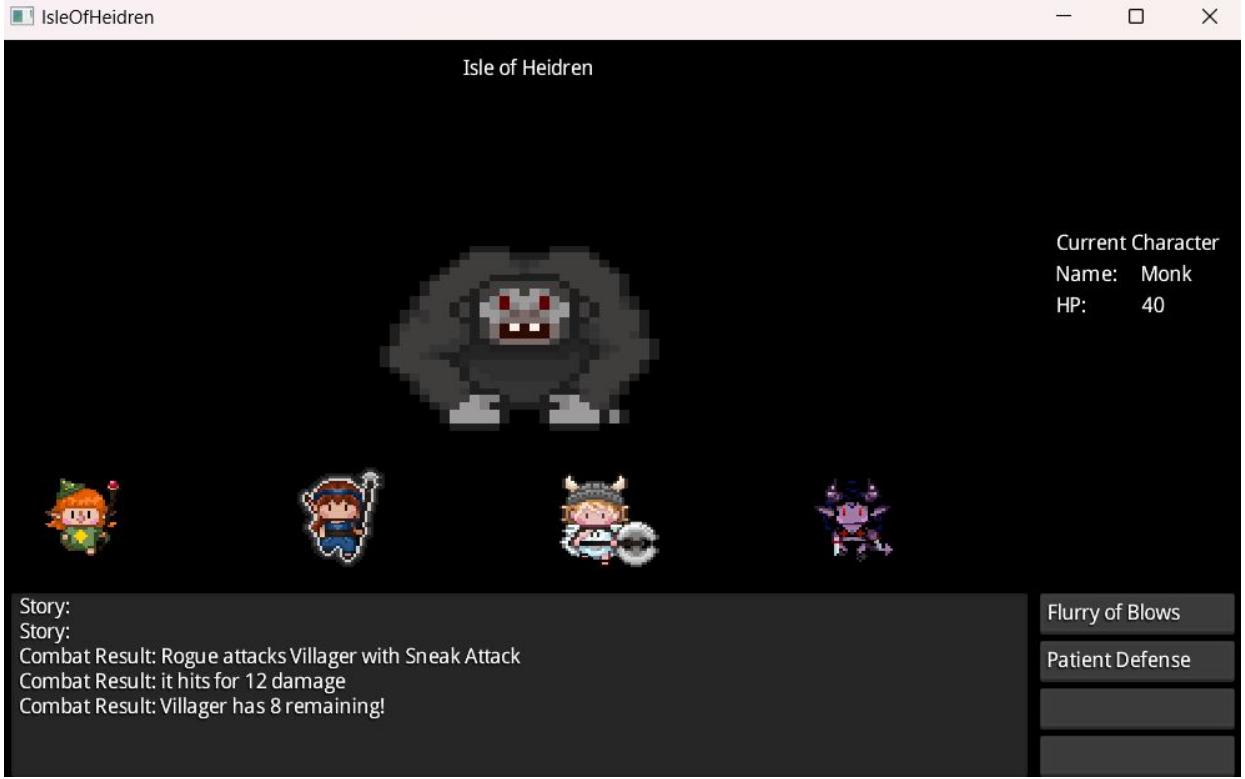

Figure 12: Screenshot of game combat

Design: Overview - Requirements 10, 20

- The software will be a game
- The game will utilize the theme and story elements of the Isle of Dread campaign.
- The game will be implemented using a game engine. LibGDX is suitable engine for the requirements https://libgdx.com/ and will be used for this project

Design: Overall Game Flow - Requirements 30, 60, 70, 130

- Main GUI
- State Manager
- Story Event Manager
- Combat Event Manager
- Map Event Manager
- Game over

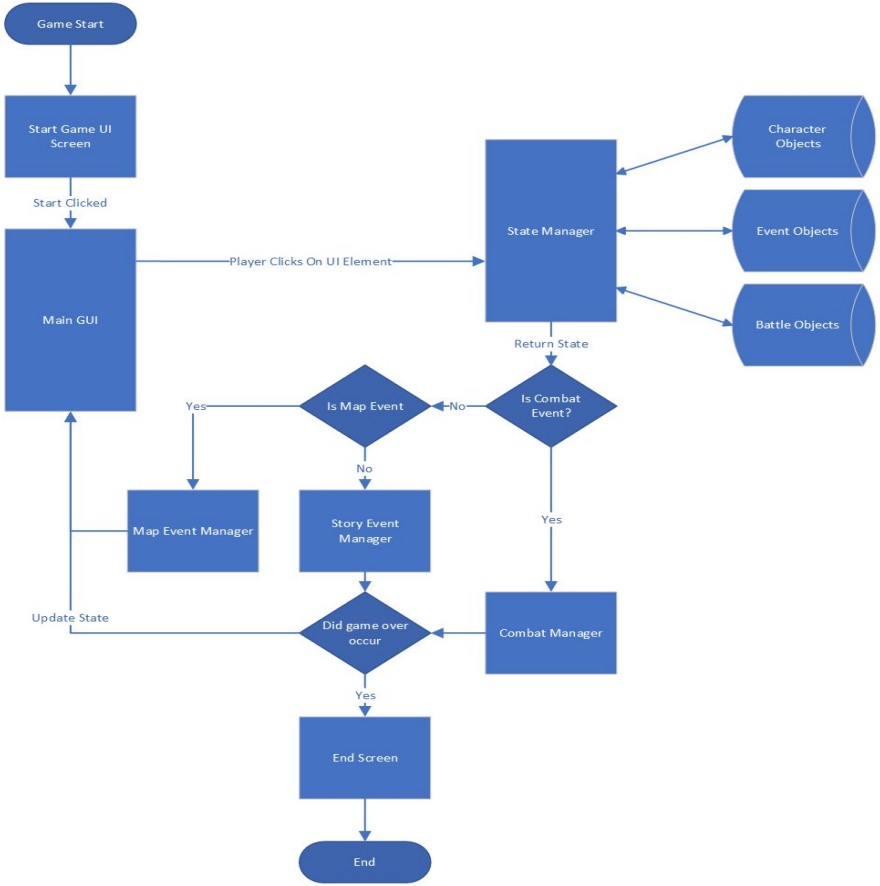

Figure 5: Overall Game Flowchart

Requirement 30: Will have a start screen

Implementation: When the game runs it opens to a start screen

Figure 13: Screenshot of start screen

Requirement 40: Will play as a predefined character

Implementation: Characters are defined in the JSON files and the game and game aspects stem from those definitions.

Ex. Wizard

```
{
  "name": "Wizard",
  "features": [
    {
      "isCombat": true,
      "damage": 15,
      "name": "Fire Bolt"
    },
    {
      "isCombat": false,
      "damage": 0,
      "name": "Poison Spray"
    },
    {
      "isCombat": true,
      "damage": 40,
      "name": "Chill Touch"
    }
  ],
  "armorClass": 15,
  "hitPoints": 25,
  "attackBonus": 4,
  "weapon": "staff"
}
```

Figure 14: Character Json

Requirement 50: Will play with four characters

Implementation: Four characters are on the main screen and combat rotates through them

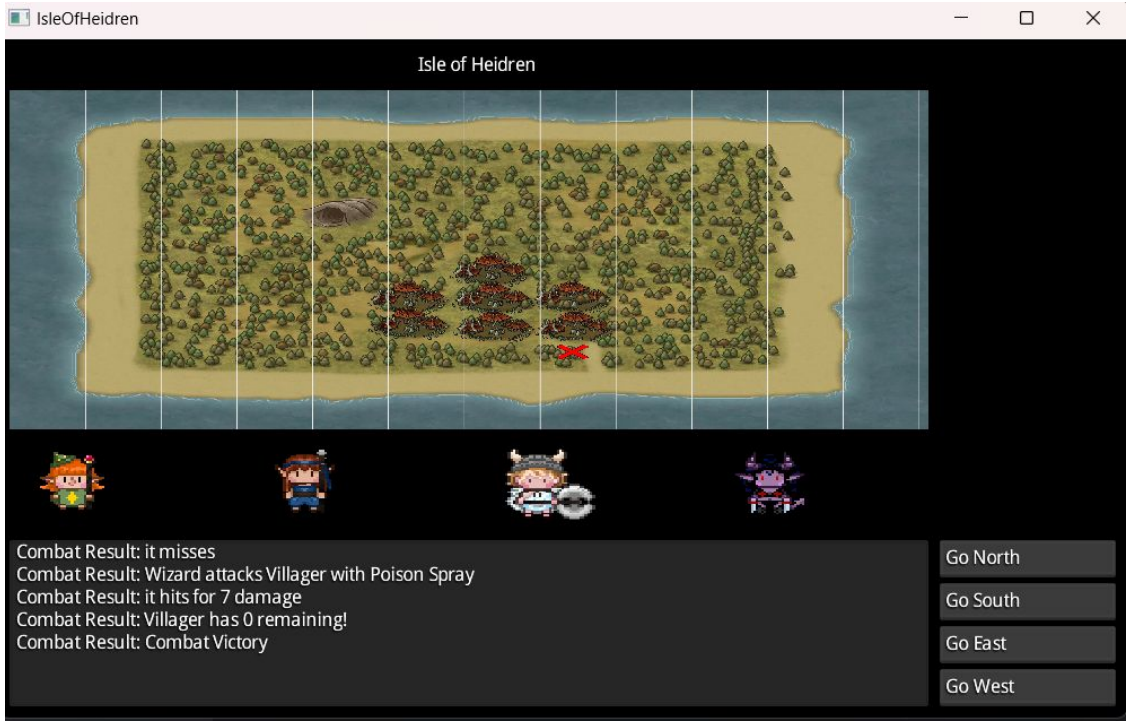

Figure 15: Screenshot of game with character sprites

Requirement 51: Each player will have a set of stats, features/abilities that differ from one another. These stats will be based from DnD

Implementation: The wizard has stats defined in their json document

```
1  {
2    "name": "Wizard",
3    "features": [
4      {
5        "isCombat": true,
6        "damage": 15,
7        "name": "Fire Bolt"
8      },
9      {
10       "isCombat": false,
11       "damage": 0,
12       "name": "Poison Spray"
13     },
14     {
15       "isCombat": true,
16       "damage": 40,
17       "name": "Chill Touch"
18     }
19   ],
20   "armorClass": 15,
21   "hitPoints": 25,
22   "attackBonus": 4,
23   "weapon": "staff"
24 }
```

Figure 16: Screenshot of Json Object

Design: Defined Characters - Requirement 40, 50, 51

- Characters will be defined using JSON objects they will follow the formatting defined to the right.
- Characters file includes combat attacks/non-combat attacks that the player can do.
- These actions will be created and implemented based from there original DND counterparts.
- All actions the player can take in game will come from the predefined characters.

Figure 17: Screenshot of gnome design

Design: GUI - requirements 60 - 65

GUI components are implemented as widgets using libGDX's table class in a black background with white text and detailing

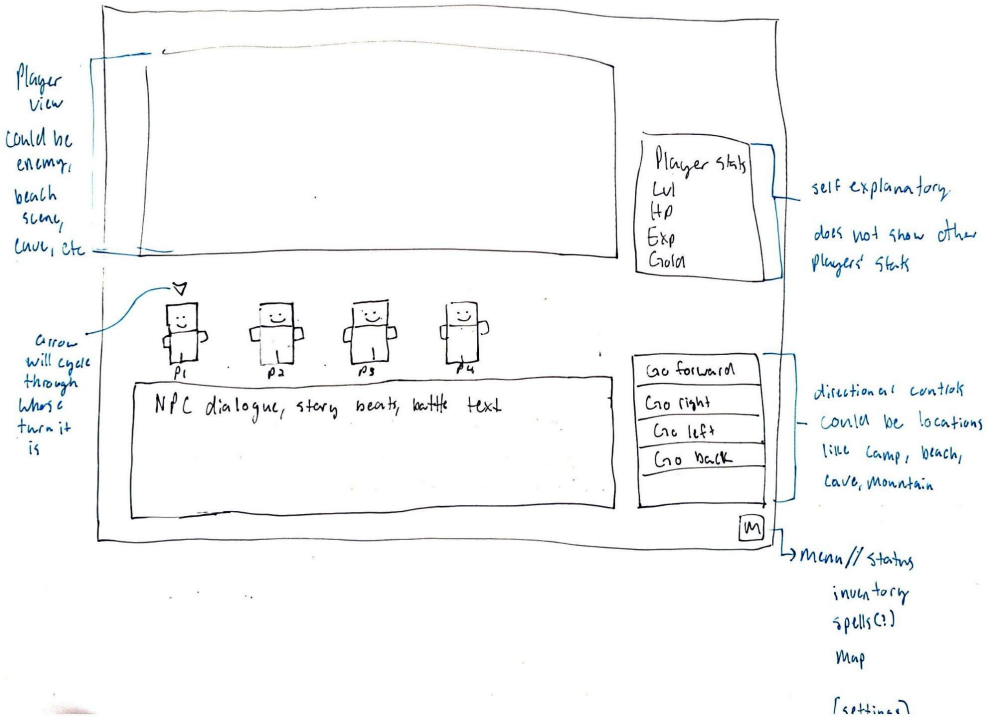

Figure 18: Drawing of Original UI Mock

Requirement 60: Will have a main GUI with components that change dynamically based on the game state

Implementation: The game changes from a map to a animated combat screen with a stats panel

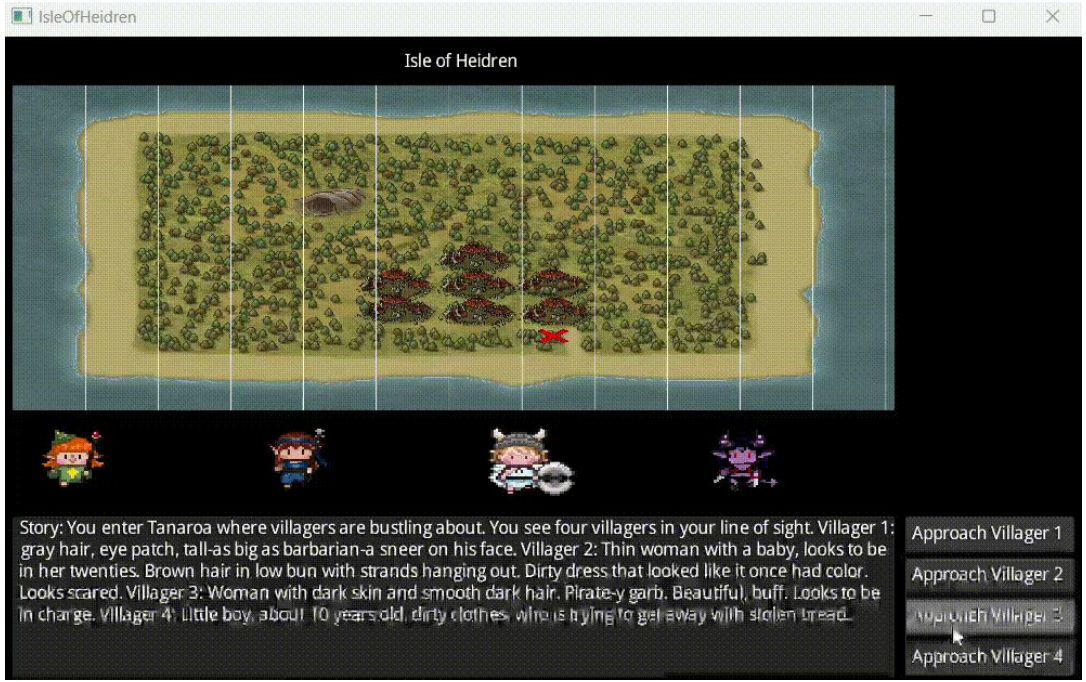

Figure 19: Gif of the main image display changing based on the game state

Requirement 61.0: Will have buttons that are mouse clickable to interact with the game

Implementation: game responds to mouse clicks

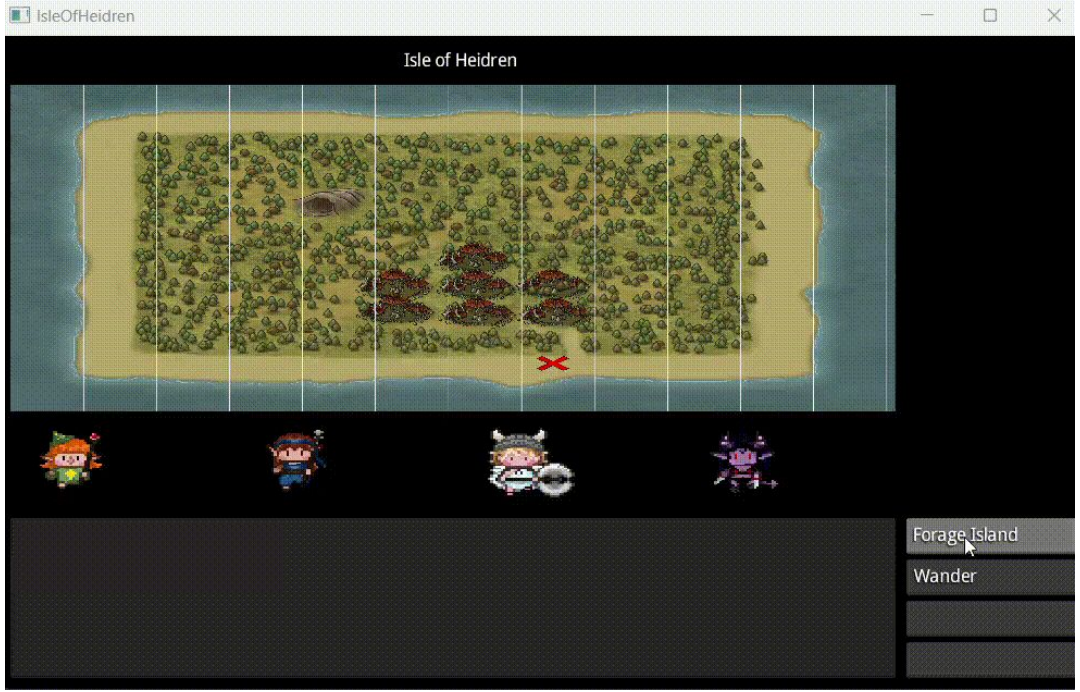

Figure 20: Gif of game responding to mouse input in the button panel

Requirement 61.1: Buttons shall be dynamic based on valid story actions, and will include character specific actions

Implementation: buttons change based on if in story or combat or on the map

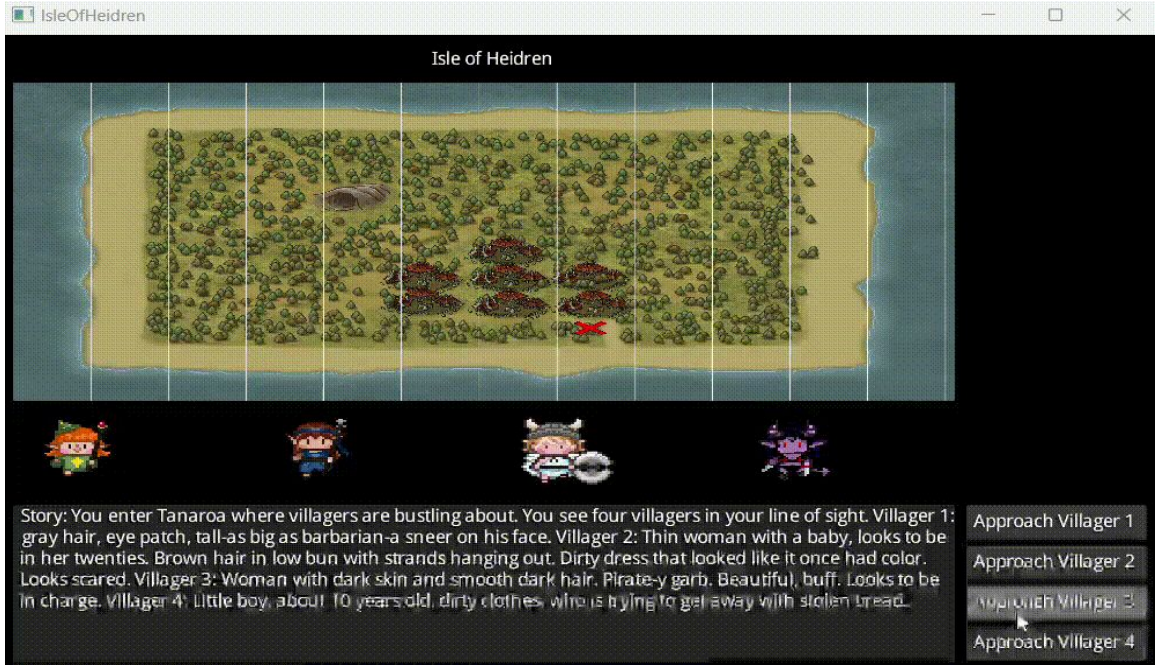

Figure 21: Gif of button options switching from story mode to combat mode

Requirement 61.2: In combat each player will take turns doing one action which will be defined in the button panel

Implementation: The combat mode shows which player is active with highlighting and the relevant attacks are on the buttons

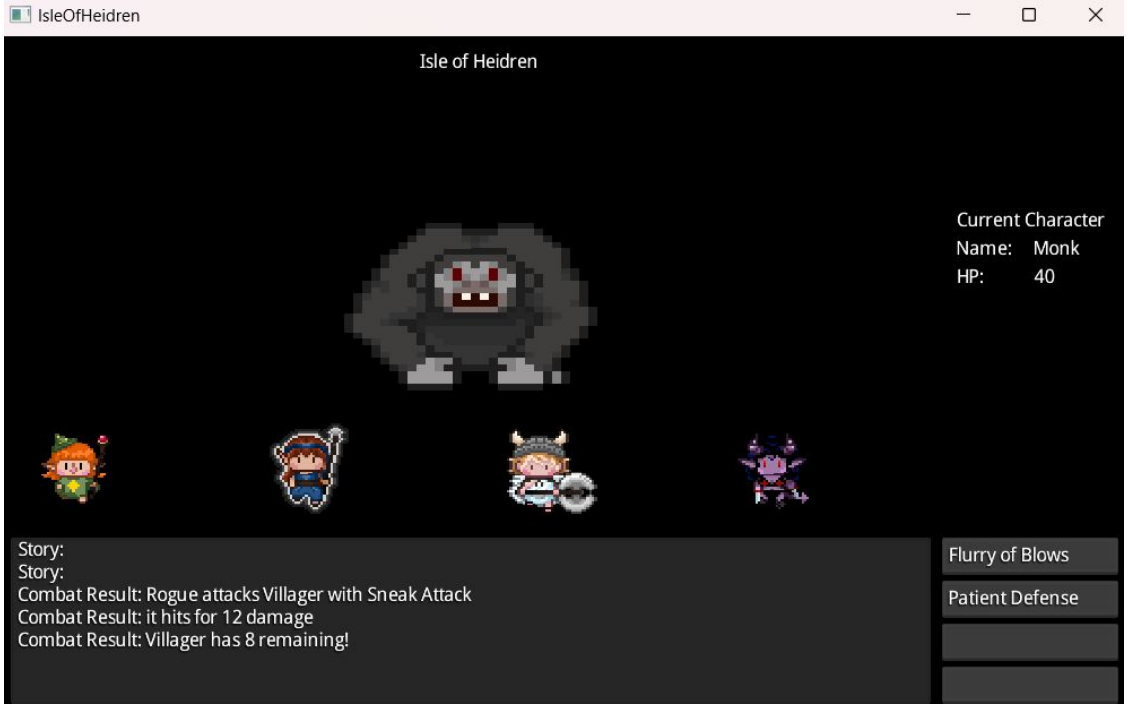

Figure 22: The game showing highlighted monk character

Design: Buttons - requirements 60, 61

Buttons will be implemented using two button components defined by LibGDX:

- TextButton
- ImageButton

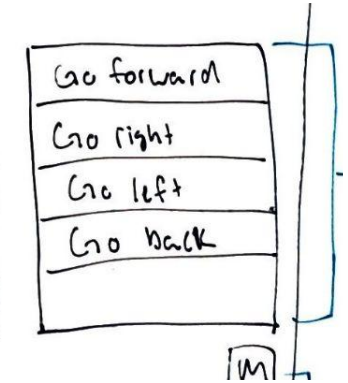

Figure 23: Our button panel mock

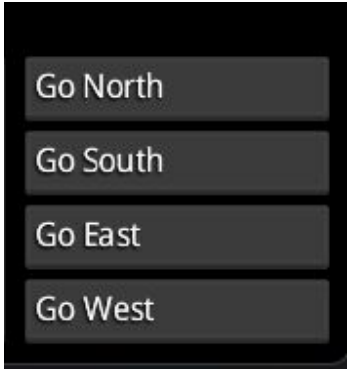

Figure 24: Our final button panel

Requirement 62.0: Shall have a primary image container showing environmental PNGs that are relevant to the current game state

Evaluation Failed

Implementation: We ending up focusing on the combat and map images and did not have enough time to create images for the various scenes.

Requirement 62.1: During combat or NPC interactions, the image container will display relevant NPCs and monsters

Implementation: The game either shows a image of a monster you are fighting or where you are on the map

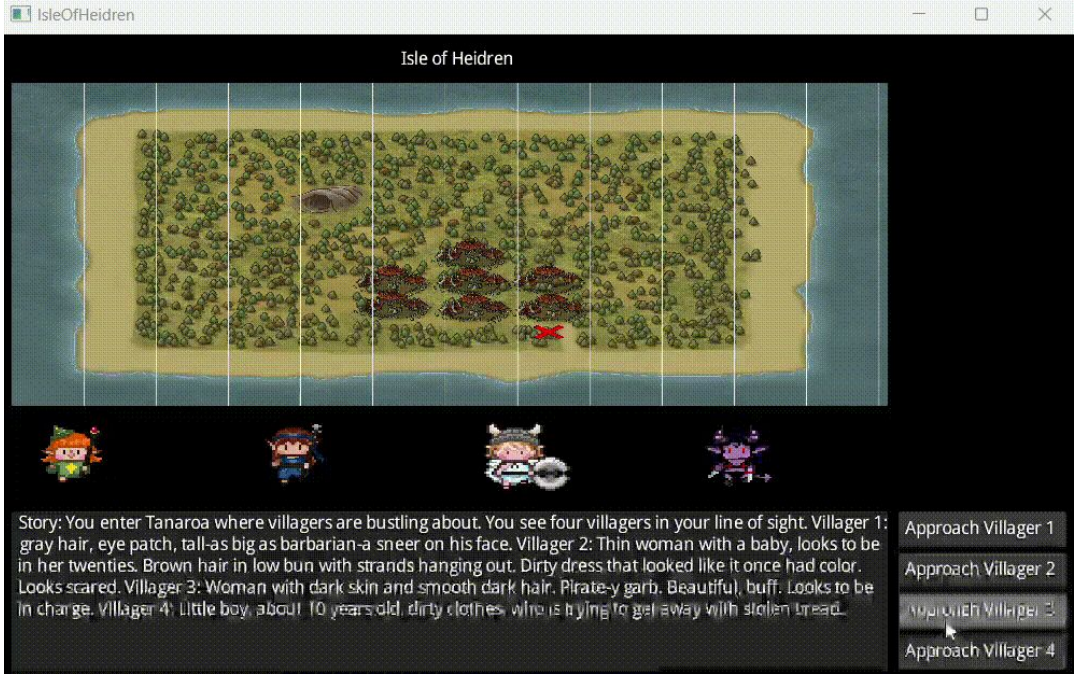

Figure 25: Gif of the contents of the primary image display changing based on the game state

Design: Primary Image Display - requirements 60, 62

- Image container shall be 600 x 400 pixels
- This component will load the story data from the event object and load the relevant image.

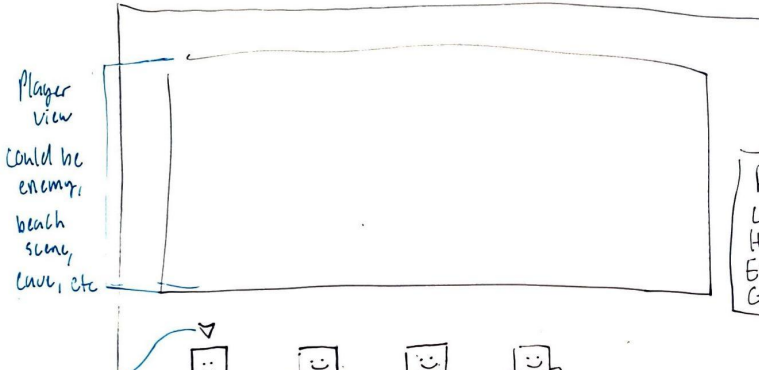

Figure 26: Our original mock of the primary image display

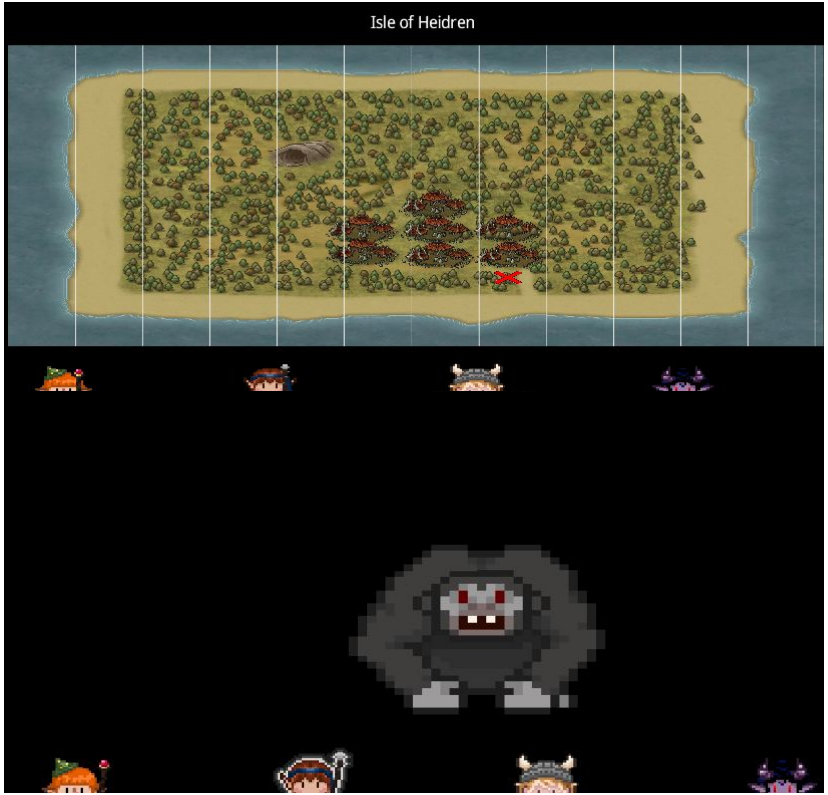

Figure 27: The primary image display showing the map and a monster

Requirement 63: Will have a console-like output for game text, battle results, and story

Implementation: The game has a console like text output box at the bottom of the screen

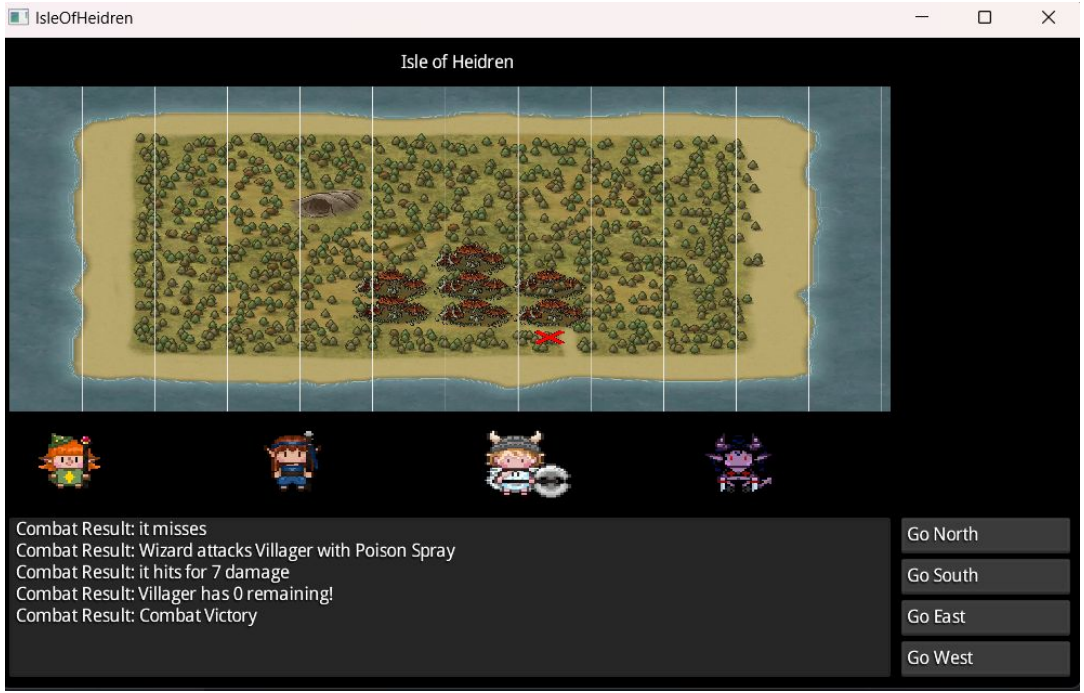

Figure 28: The text console displaying battle text

Design: Console Component - requirements 60, 63

- A console like component will be created
- The console display will be 600 x 100 pixels.
- It will be in the style of a game dialog box and allow the user to scroll up or down to view previous and the current text. The scrolling action will respond to mouse click and drag or mouse wheel input.

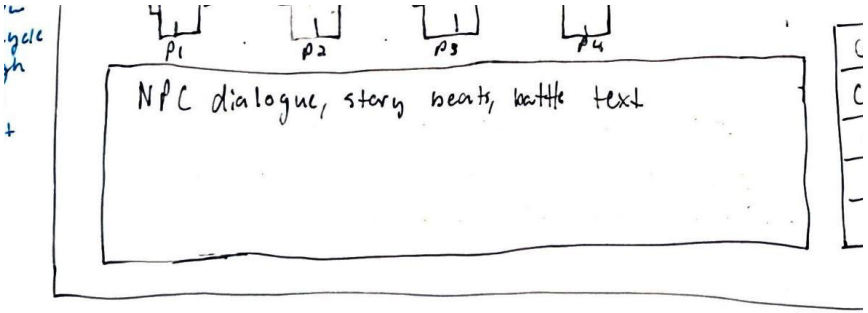

Figure 29: Our original mock of the console component

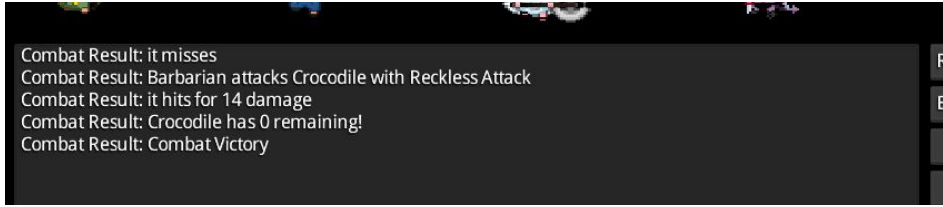

Figure 30: The final console component displaying battle text

Requirement 64.0: Each playable character shall be pictured as sprites in the main window. These sprites may be animated.

Implementation: Animated sprites are on the screen for all the characters

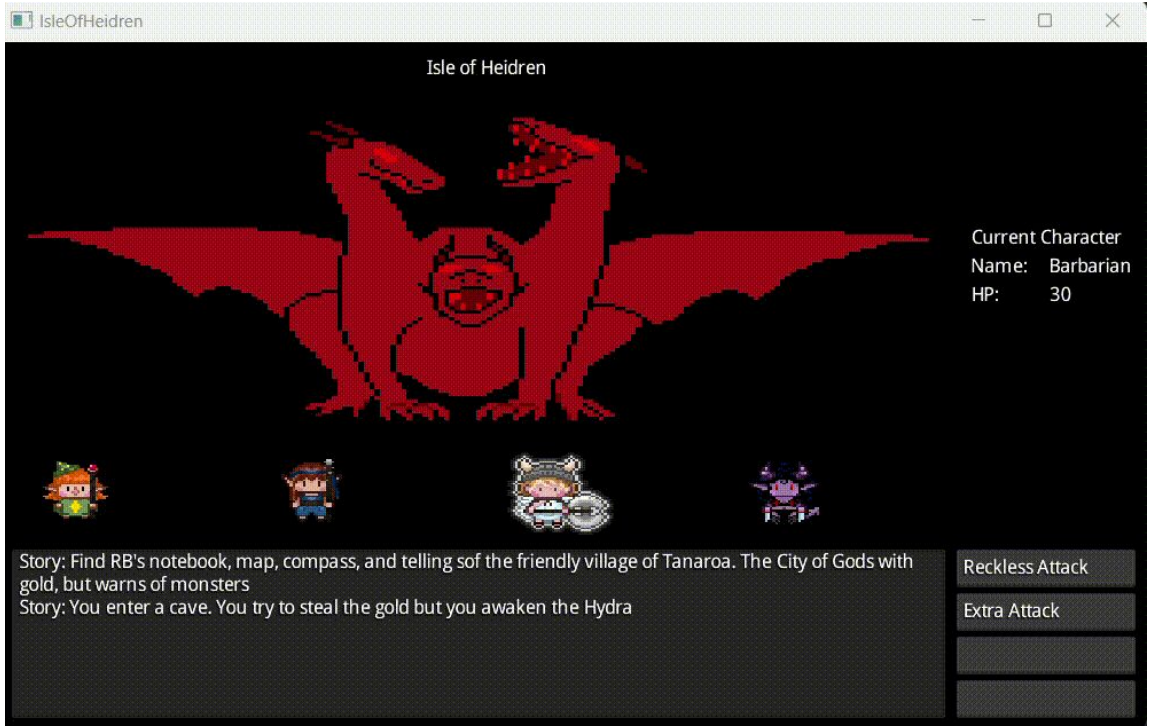

Figure 31: The game displaying the character sprites and the hydra in battle

Requirement 64.1: UI will indicate which character is active (turn) in combat by highlighting the character's sprite. This includes monster and NPC sprites

Evaluation Partially Passed

Implementation: character sprites are highlighted during their turn in combat, but NPCs and monsters are not

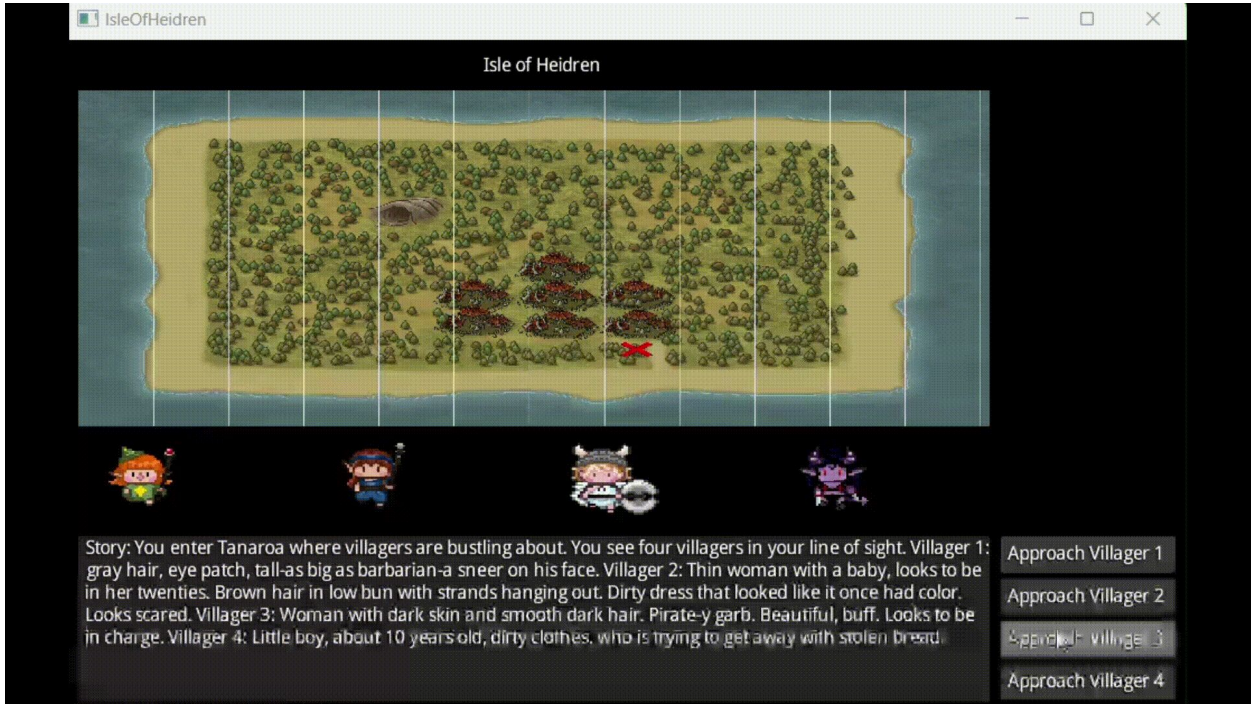

Figure 32: The game showing the highlighted character changing based on whose turn it is in battle

Design: Character Sprites - requirement 64

- Root table will display the four sprites side by side in 100 x 100 pixel blocks above the terminal and below the image display
- The active fighter will be denoted by a white highlight that will loop through each character in the proper fighting order, including enemies

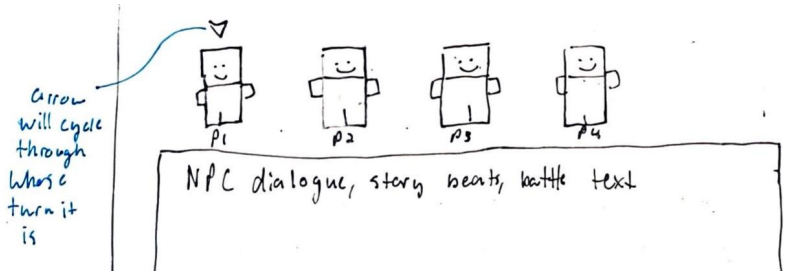

Figure 33: Original mock of the sprite lineup and the active fighter denotation

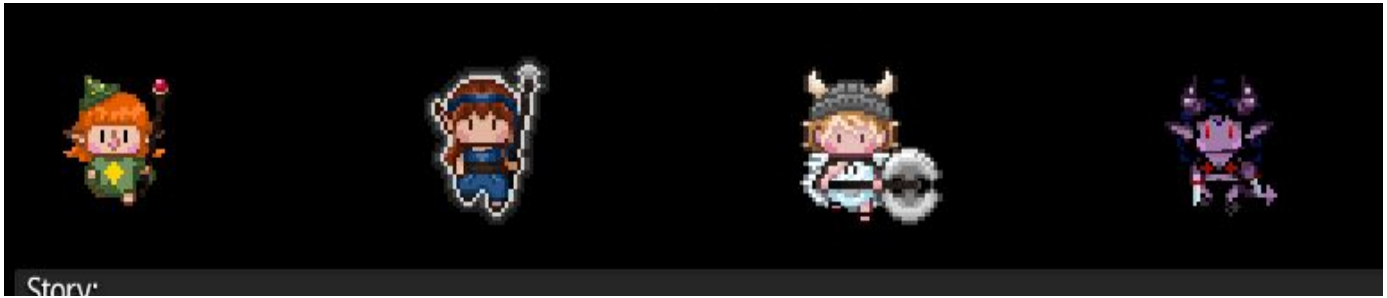

Figure 34: Final character sprite lineup with active fighter denotation

Requirement 65: In combat, the currently selected character's stats shall be pictured next to the image container. Outside of battle, the panel shall be blank

Implementation: The stat panel displays when combat is entered

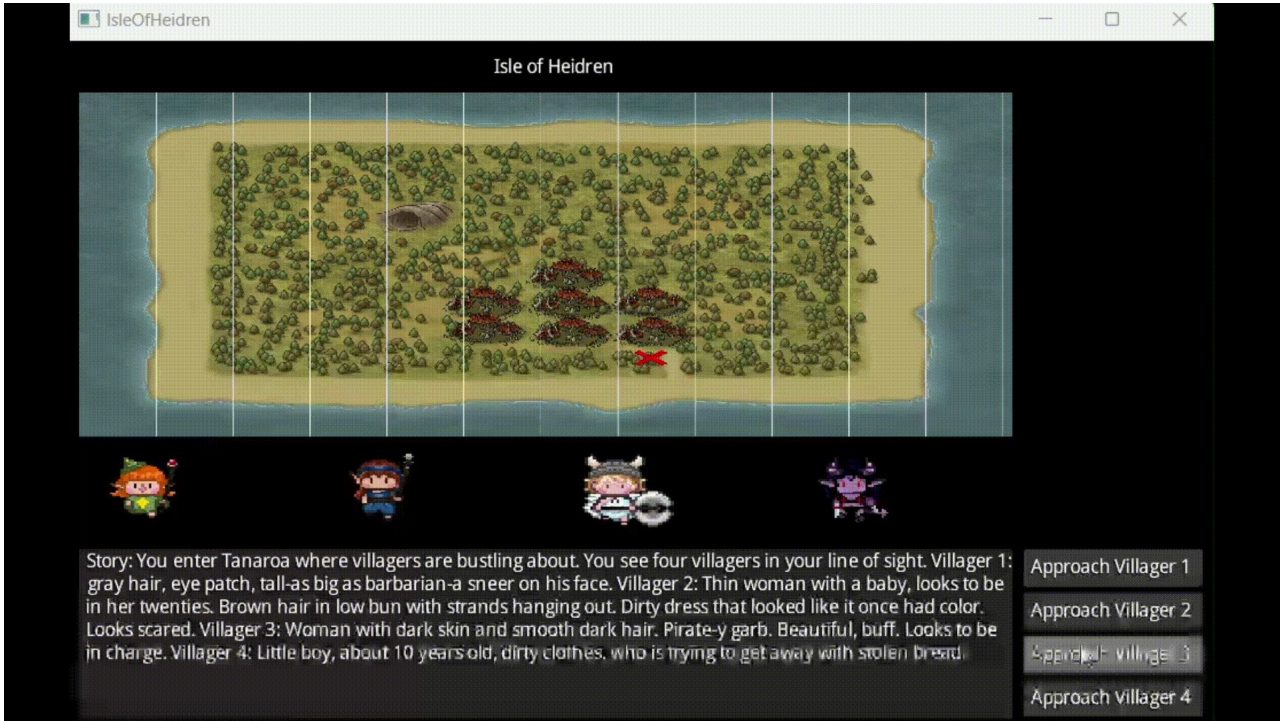

Figure 35: Gif of the stats panel changing based on the current active character

Design: Stats Panel - requirement 65

- Text Panel to the side of main image display showing currently selected character's HP value during battle.
- Blank panel outside of combat.
- Stat panel is 100x100 pixels.

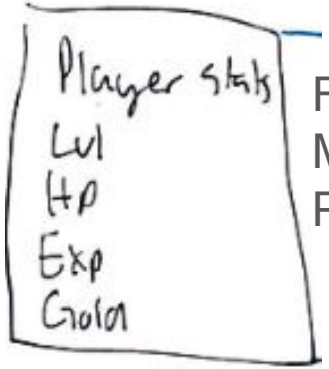

Figure 36: Mock GUI Stat Panel

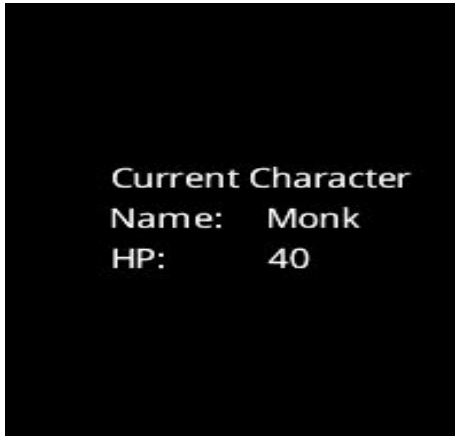

Figure 37: Final GUI Stat Panel

Requirement 70: Shall have a way to retrieve data for stats, characters, monsters, and events

Implementation: The game utilizes a repo model to prevent hard coupling of specific game logic

```
{
  "name": "Crocodile",
  "hitPoints": 30,
  "armorClass":  7,
  "attackBonus":  0,
  "attacks": ["Bite", "Tail Whip", "Stomp"]
}
```

Figure 38: Screenshot of Croc Json Definition

Design: Data Loading - requirement 70

The game will have all the necessary objects for story, monsters, players, events stored in json objects that are loaded in sequence as needed. Event objects will include references to different possible branches that can be obtained. The UI will load these then go and fetch the next object as needed.

```
{        bricklerb, 2 months ago • re
    "Name": "Start",
    "Sequence": 1,
    "Branches": [
        2,
        3,
        4
    ],
    "Branch Text": [
        "Dive into the water",
        "Talk to the fisherman",
        "Look around the outskirts"
    ],
    "Battle": false
}
```

Figure 39: Original Json Mock

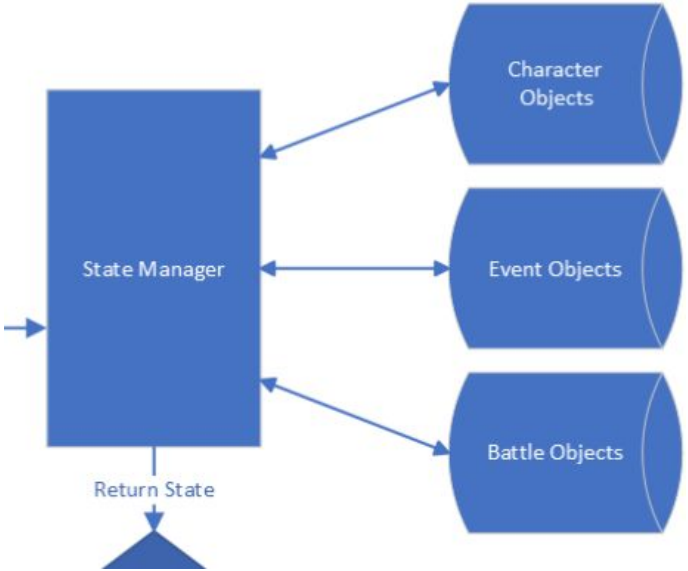

Figure 40: State Manager Repo Interaction

Requirement 80: Will have map in database that either will be shown by text or GUI component

Implementation: map displayed in image window

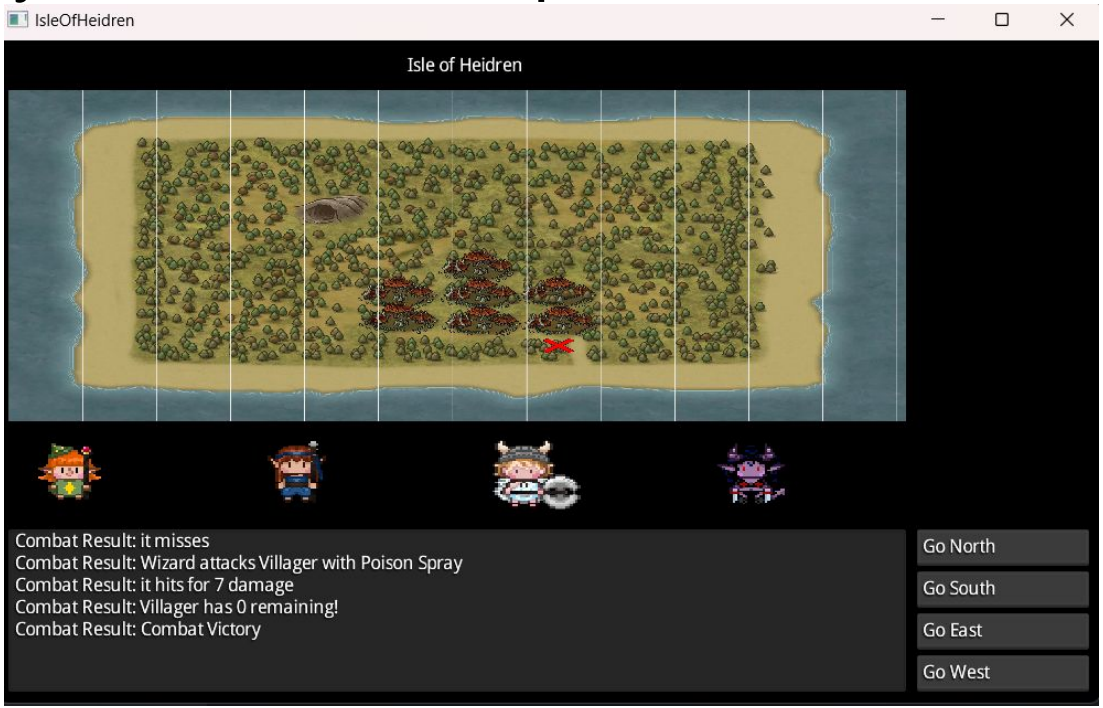

Figure 41: Screenshot of Game on Map Screen

Requirement 81: Map will have multiple locations

Implementation: The map is a 10x10 grid with events defined some of the squares. Events have locations defined within their json

```
{        20Heedje, 24 hours ago • workin
  "sequence": 22,
  "branches": [],
  "buttonsText": [],
  "consoleOutputText": "A crocodile!",
  "locX": 1,
  "LocY": 2,
  "monsters": ["Crocodile"]
}
```

Figure 42: Json File Defining an Event

Design: Map - requirement 80

Map is a 10x10 grid that allows the player to move between different squares.

Figure 43: Final Map Implementation

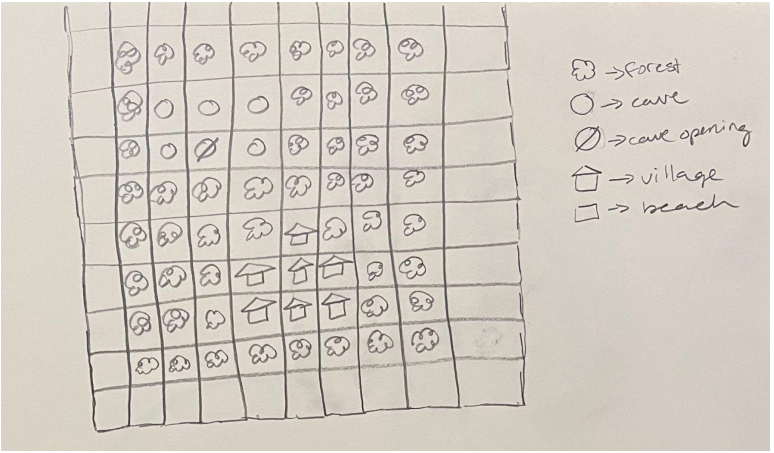

Figure 44: Original Map Mock

Requirement 90: Will have story from start to finish no matter what combination of predefined actions each character takes

Evaluation Failed

90 Implementation: The game has many story events and a final fight, but does not redirect to a game over or game victory screen when story should be over. We did not have enough time with the combat to link things together properly to achieve that

Design: Story - requirement 90

JSON Events created based on a story flow chart

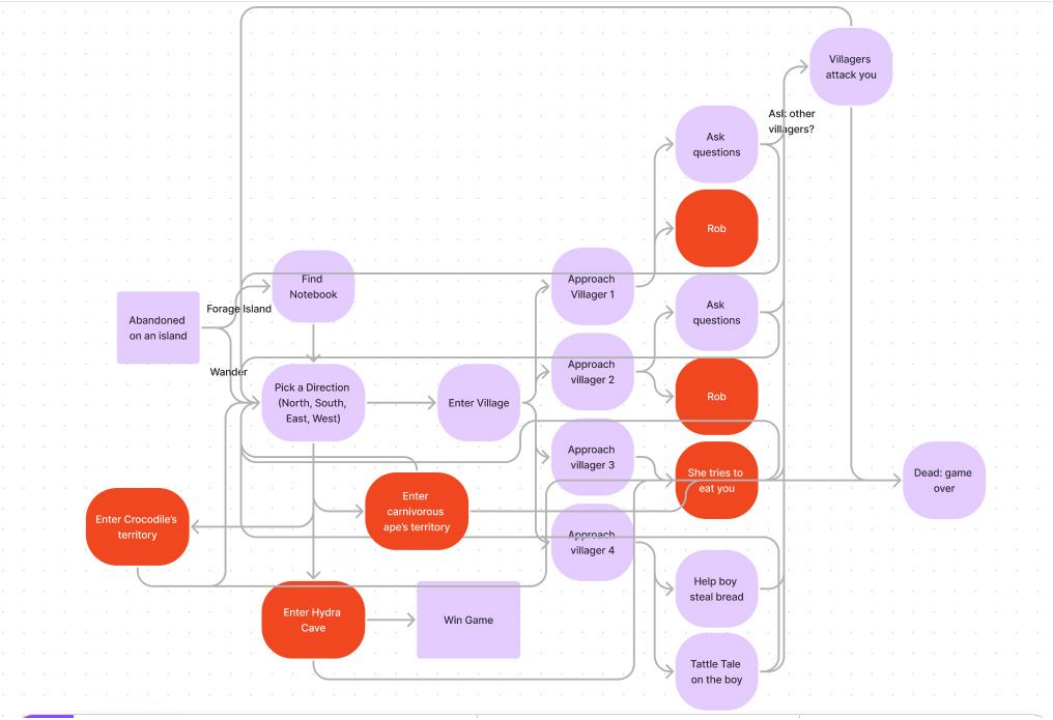

Figure 45: Story Flow Chart

Requirement 100: Shall have at least 3 types of monsters

Implementation: The game has 4 monster types: ape, villager, hydra, crocodile

```
{
  "name": "Villager",
  "hitPoints": 20,
  "armorClass":  5,
  "attackBonus":  0,
  "attacks": ["Bite", "Fist", "Stomp"]
}
```

Figure 46: Monster Json Definition

Design: Monsters and Monster Attacks - requirement 100

- Three monsters taken from DnD
- Attacks inspired by original DnD moves. For example: Fire Breath - Ranged Attack (no weapon bonus), DMG is equal to (1d12 + 5).

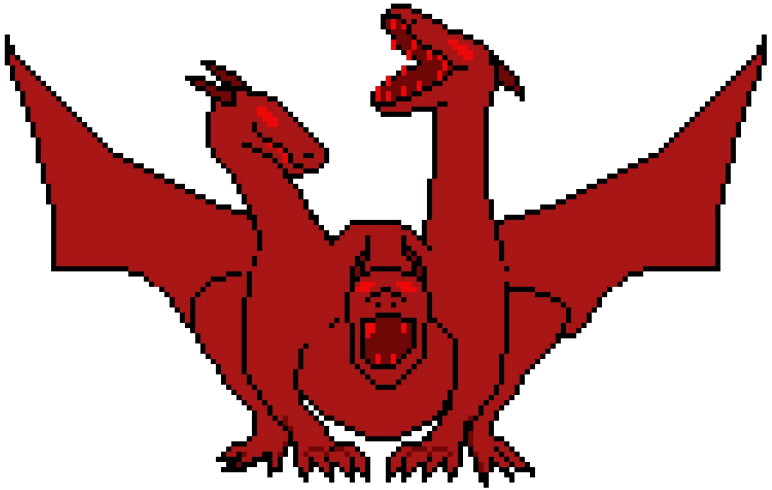

Figure 47: Hydra sprite

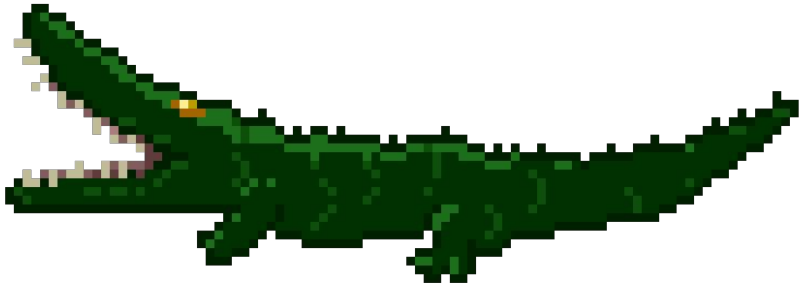

Figure 48: Crocodile sprite

Figure 49: Villager Sprite

Figure 50: Ape sprite

Requirement 105: The combat system shall follow DnD combat standards with a clear winner and damage taken to each side

Implementation: Game outputs a victory message if the combat is a win

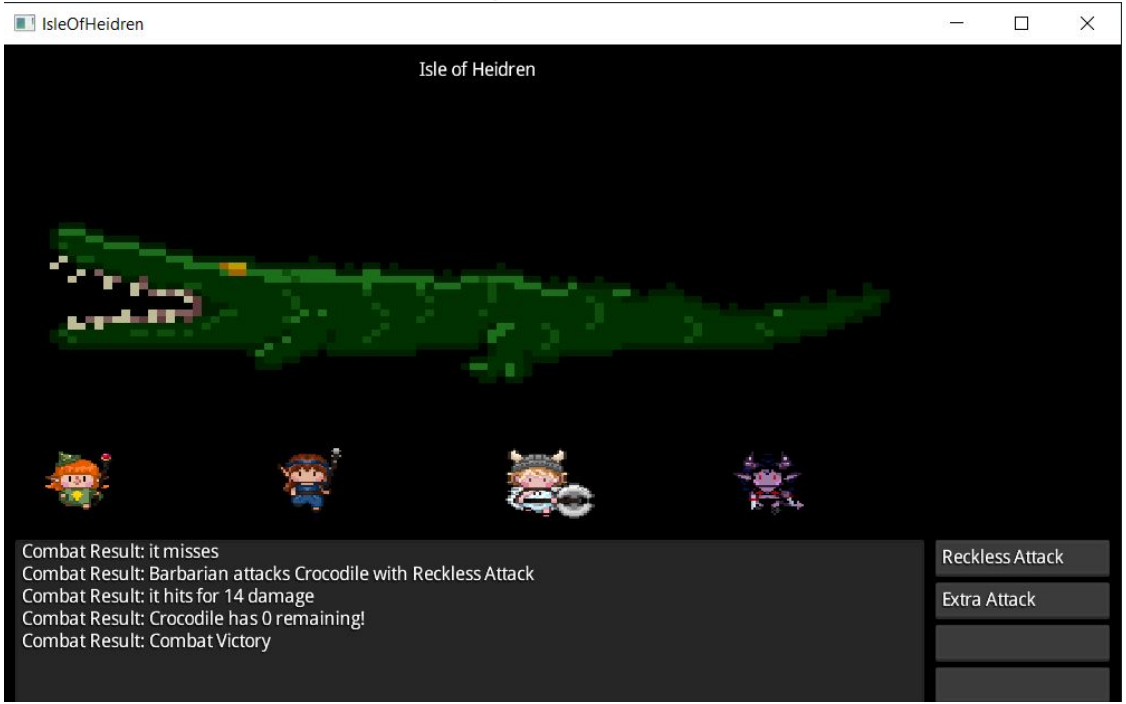

Figure 51: Screenshot of Croc Combat

Design: Combat System - requirement 105

- Turn based system of combat
- During combat, player will have option of which attack they want to use
- Attack chosen's damage is determined by dice roll

```java
public int rollDice(int sides) {
    Random random = new Random();
    return random.nextInt(sides) + 1;
}

public int tryAttack() {
    return rollDice(sides:20) + getAttackBonus();
}

public int attack(String chosenAttack) {
    int damageDealt = 0;
    switch (chosenAttack) {
        case "Sneak Attack":
            damageDealt = rollDice(sides:8) + getAttackBonus();
            break;
        case "Thieve's Cant":
            damageDealt = rollDice(sides:6) + getAttackBonus();
            break;
```

Figure 52: Dice Roll to Show Attack Output (ex. "Sneak Attack" & "Thieve's Cant"

Requirement 110: The game will prompt the player with story options displayed in the UI utilizing the buttons, image container, and console

Implementation: The game shows relevant buttons and when clicked processes to new story elements

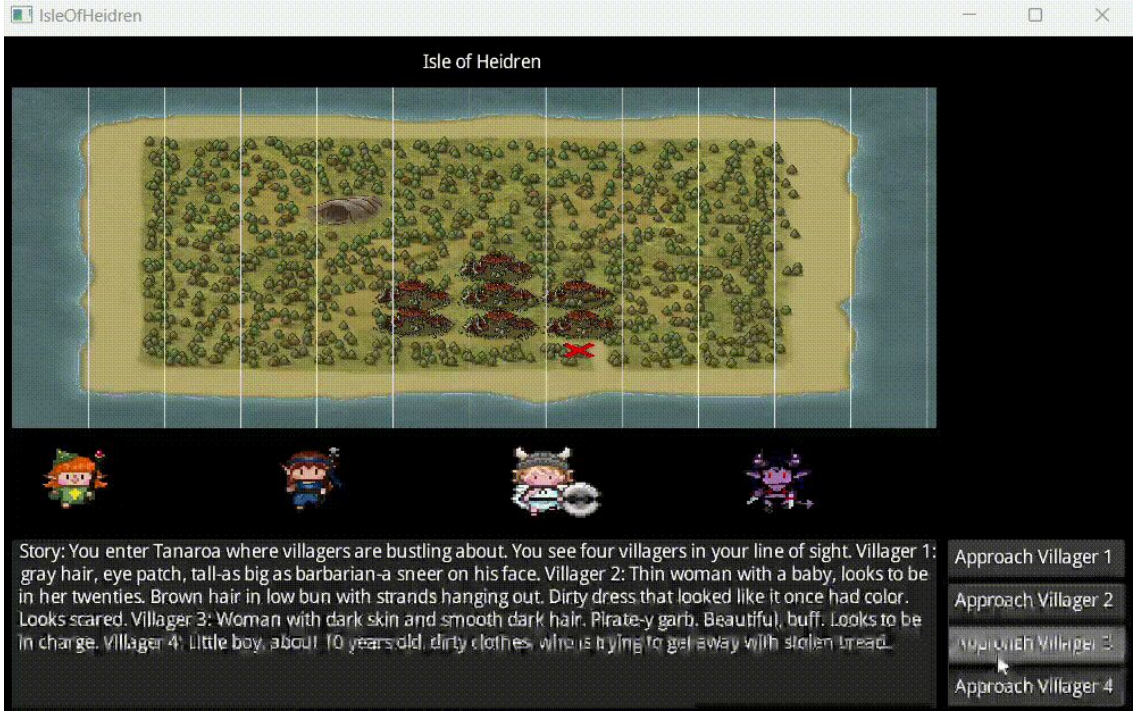

Figure 53: Gif of the GUI switching component options and displays based on user actions

Design: Story Option Prompts - requirement 110

The GUI shall change the contents of its components based on a given story event pulled from a JSON file

```
{        20Heedje, 24 hours ago • workin
  "sequence": 22,
  "branches": [],
  "buttonsText": [],
  "consoleOutputText": "A crocodile!",
  "locX": 1,
  "LocY": 2,
  "monsters": ["Crocodile"]
}
```

Figure 54: An JSON file with an event that occurs at location (1, 2)

Requirement 120: The game will have a combat mode when the story dictates it necessary

Implementation: Various events are defined as combat events and those trigger how many monsters and what monsters the player will fight

```
{        20Heedje, 24 hours ago • workin
  "sequence": 22,
  "branches": [],
  "buttonsText": [],
  "consoleOutputText": "A crocodile!",
  "locX": 1,
  "LocY": 2,
  "monsters": ["Crocodile"]
}
```

Figure 55: JSON event object of a combat event with a crocodile

Design: Story Combat Encounter Instances - requirement 120

You can approach another villager but the entire village appears wary and scared of your arrival.

Fig 1: Story text warning about villagers

- Given a certain spot within the story, combat encounters will occur.
- Could be based on quests, surprise encounters, etc.
- Approaching multiple villagers with a warning (Fig 1) results in a combat encounter with villager (Fig 2).

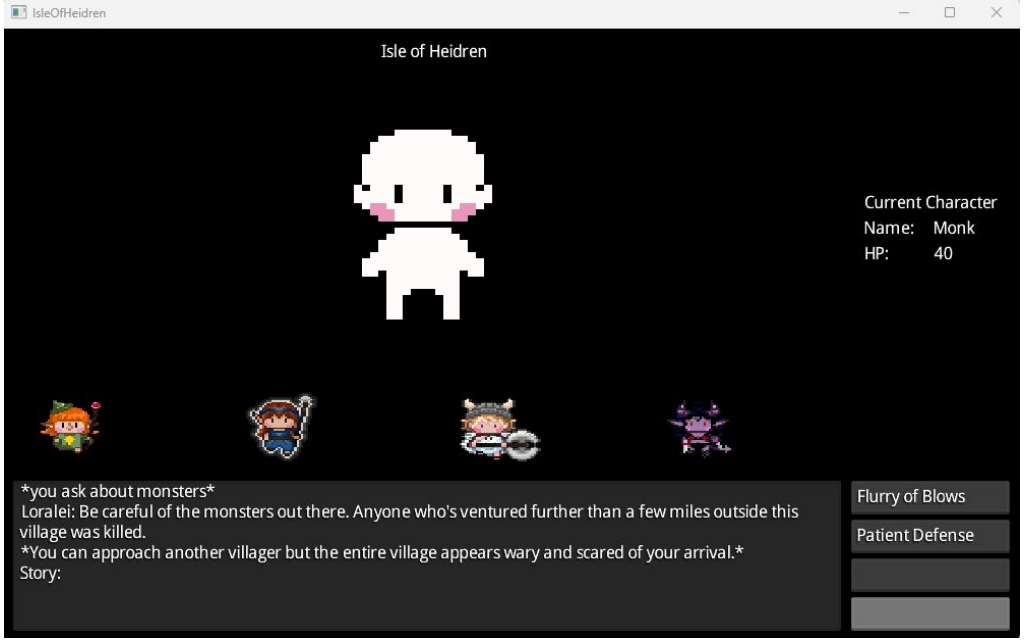

Fig 56: Combat encounter with villager

Requirement 130: Will have an end screen that displays a game over or game won text

Evaluation Failed

Implementation: We ran out of time to implement the game over screen. Early combat components took longer than expected to complete and those were required to link to a screen.

Design: Game Over Screen - requirement 130

- Game over screen was going to have text displayed saying either “Adventure Failed” or “Adventure Success”
- Button to close game

Figure 57: Screen showing text to show how adventure went with an “Exit” button to close game

Video Demonstration

https://www.youtube.com/watch?v=_SDSrBHJ73Q