Project Files: https://github.com/Jayden-58/Random_Adventure_Generator

Introduction

My portfolio project "Random Adventure Generator" is a text-based adventure game. The main Idea of the Random Adventure Generator is for the player to have a completely unique adventure every time they play, In both map layout, and story.

Key features:

- Proceedurally generated map
- Proceedurally generated characters
- Player navigation with an updating map as the player explores
- Random encounters
- Fight system with critical hits and missed attacks
- Merchant
- Charisma system
- Proceedurally generated rooms

Design and Implementation

Proceeduralluy generated map:

The map is a 10 by 10 grid made of a list containing 10 lists. There are actually two maps being used for the game. The "player map", and the actual map (which I'll be referring to as "map"). The player map starts blank and fills in as the player navigates through the dungeon.

Blank player map:

```
['u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u']
['u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u']
['u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u']
['u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u']
['u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u']
['u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u']
['u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u']
['u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u']
['u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u']
['u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u']
```

Map:

To start, I created the function to generate the map. First, it creates a 2D list using nested for loops:

```
def make_map():
    map = []
    global end_position_rows
    global end_position_cols
    for rows in range(0, 10, 1):
        map.append([])
        for col in range(0, 10, 1):
```

Next, I needed to place walls, rooms, the start room, and the endroom. But, I needed them to be placed in such a way that all of the rooms are accessible. To do this, I wrote out a list of “rules” using conditional logic:

```
    for col in range(0, 10, 1):
        if rows == 0 or rows == 9:  # top and bottom of the map are made into walls
            map[rows].append("w")
        elif col == 0 or col == 9:
            map[rows].append("w")  # sides of the map are made into walls
        elif rows == 1 and col == 1:
            map[rows].append("S")  # the start will always be at (1,1)
        elif rows == 1 and col == 2:  # tile to the right of the start
            # determines if a square will be a room or a wall
            letter = ""
            wall_calculator = random.randint(0, 1)
            if wall_calculator == 1:
                map[rows].append("R")
            else:
                map[rows].append("w")
                # all the squares in the first non wall row (after the start and first square are already assigned)
        elif rows == 1:
            if map[rows][col - 1] == "R":
                letter = ""
                wall_calculator = random.randint(0, 1)
                if wall_calculator == 1:
                    map[rows].append("R")
                else:
                    map[rows].append("w")
            else:
                map[rows].append("w")

        elif rows == 2 and col == 1:
            # the tile under the start location reacts to how the tile to the right of the start location gets assigned.
            # if the tile to the right of the start is a wall, the tile under the start MUST be a room.
            if map[1][2] == "w":
                map[rows].append("R")
            else:
                letter = ""
                wall_calculator = random.randint(0, 1)
                if wall_calculator == 1:
                    map[rows].append("R")
                else:
                    map[rows].append("w")
        else:
            if map[rows - 1][col] == "R" and map[rows][col - 1] == "R":
                letter = ""
                wall_calculator = random.randint(0, 1)
                if wall_calculator == 1:
                    map[rows].append("R")
                else:
                    map[rows].append("w")
            elif map[rows - 1][col] == "R" or map[rows][col - 1] == "R":
                map[rows].append("R")
            else:
                map[rows].append("w")
while True:  # finds a random end point for the dungeon
    rows_index = random.randint(0, rows - 1)
    col_index = random.randint(0, col - 1)
    if map[rows_index][col_index] == "R":
        map[rows_index][col_index] = "E"
        end_position_rows = rows_index
        end_position_cols = col_index
        break
    else:
        pass
```

I also created a function to generate the player map. This function creates a 2D list, and fills it with the letter “u” for unknown :

```
# the map the player will see
def make_player_map():
    player_map = []
    for rows in range(0, 10, 1):
        player_map.append([])
        for col in range(0, 10, 1):
            player_map[rows].append("u")
    return player_map
```

Finally, I created the map class and gave it functions to print itself, and to return the coordinates of the end space.

```
class Map:
    def __init__(self, array):
        self.array = array
    def print_map(self):
        for row in range(0, 10, 1):
            print(self.array[row])
        print("  u: unknown")
        print("  w: wall")
        print("  R: room")
        print("  S: start")
        print("  E: exit")
        print("  P: player")
    def get_end_rows(self):
        return end_position_rows
    def get_end_cols(self):
        return end_position_cols
```

Procedurally generated characters:

First, I created the character class. It has two parameters, health, and motivation. The health will be used to check if the player is alive before every turn starts. The player's motivation is part of the randomly generated story. Within the player class, I defined 3 functions. The first

function is to check if the player is alive by making sure the health of the player is greater than zero. Next, I have the function to introduce the player at the beginning of the game, by displaying their motivation, health, and weapon.

```
class Player:
    def __init__(self, health, motivation):
        self.health = health
        self.motivation = motivation

    def check_alive(self):
        if self.health > 0:
            return True
        else:
            return False

    def introduce_player(self,  weapon):
        print("You entered the dungeon seeking " + str(self.motivation) + ".")
        print("your health is: " + str(self.health) + ".")
        print("you brought your trusty " + weapon + ".")

    def print_health(self):
        print("your health is: " + str(self.health))
```

In order to get the parameters to make the player, I created functions to generate the player's health, and to pick a motivation for the player.

```
def generate_player_motivation():
    player_motivations = ("glory", "gold", "revenge", "power", "wisdom", "...nothing. You are he
                          "to destroy anything in your path", "to learn about acient beings who
    player_motivation_index = random.randint(0, len(player_motivations) - 1)
    new_player_motivation = player_motivations[player_motivation_index]
    return new_player_motivation


def generate_player_health():
    new_player_health = random.randint(100, 200)
    return new_player_health
```

Another type of character I needed to create was the monsters the character would fight. The monster is a much simpler character and did not require a class, So I created a function to return a string with the monster's name.

```
def monster_generator():
    monsters = ("warewolf", "skeleton", "gobllin", "centaur", "gargoyle
                "vampire", "zombie", "ghost", "elf", "bandit", "theif")
    descriptions = ("tall",  "short", "huge", "tiny", "mystrious",
                    "wild", "brave", "angry", "hungry", "blood thirsty"
    monster_index_number = random.randint(0, len(monsters) - 1)
    description_index_number = random.randint(0, len(descriptions) - 1)
    new_monster = (str(descriptions[description_index_number]
                    ) + " " + str(monsters[monster_index_number]))
    return(new_monster)
```

Each character comes with a weapon object. Weapons will be assigned to characters at random. players, monsters, and merchants will each have one. To make the weapons, I first created the weapon class.

```
class Weapon:
    def __init__(self, weapon_type, weapon):
        self.weapon_type = weapon_type
        self.weapon = weapon

    def get_action_word(self, weapon_type):
        if weapon_type == "projectile":
            return "shoot"
        elif weapon_type == "swingable":
            return "swing"
```

To generate weapons, I needed to create 2 functions. The first one determines weather the weapon will be projectile based, or swinging based.

```
def make_weapon_type():
    weapon_types = ("projectile", "swingable")
    weapon_type_index = random.randint(0, len(weapon_types)-1)
    weapon_type = weapon_types[weapon_type_index]
    return weapon_type
```

Once the weapon's type is determined, The next function's purpose is to get a name for our new weapon object.

```
def make_weapon(weapon_type):
    if weapon_type == "projectile":
        weapons = ("bow", "flintlock pistol", "crossbow",
                   "double barreled shotgun", "revolver", "slingshot", "rocket launcher", "musket",
    elif weapon_type == "swingable":
        weapons = ("sword", "great sword", "knife", "axe",
                   "battle axe", "war hammer", "dagger", "baseball bat", "golf club", "nunchucks",
    weapon_index_number = random.randint(0, len(weapons) - 1)
    new_weapon = weapons[weapon_index_number]
    return new_weapon
```

Player Navigation:

The next thing to create was a way to have the player navigate the map. First, I created a function that tells the player the directions that they can travel. (The player is not able to travel through walls). After the player inputs their direction choice, the function makes sure the choice is valid, and calls a function to update the player's position and also stores the player's previous location on the map, so that they can go back if they choose.

```
def player_navigation(map):
    can_north = False
    can_south = False
    can_east = False
    can_west = False
    global player_back_direction
    #player_position = map[player_rows_position][player_cols_position]
    if map[player_rows_position][player_cols_position + 1] == "w":  # east of the player
        pass
    else:
        if player_back_direction == "east":
            print("player can go east (back)")
        else:
            print("player can go east")
        can_east = True
    if map[player_rows_position][player_cols_position - 1] == "w":  # west of the player
        pass
    else:
        if player_back_direction == "west":
            print("player can go west (back)")
        else:
            print("player can go west")
        can_west = True
    if map[player_rows_position + 1][player_cols_position] == "w":  # south of the player
        pass
    else:
        if player_back_direction == "south":
            print("player can go south (back)")
        else:
            print("player can go south")
        can_south = True
    if map[player_rows_position - 1][player_cols_position] == "w":  # north of the player
        pass
    else:
        if player_back_direction == "north":
            print("player can go north (back)")
        else:
            print("player can go north")
        can_north = True
    print(" ")
    valid_choice = False

    while valid_choice == False:
        direction_choice = input("Enter the direction You'd like to go (n, s, e, w): ")
        if direction_choice == "n" and can_north == True:
            player_back_direction = "south"
            update_player_position("north")
            valid_choice = True
        elif direction_choice == "s" and can_south == True:
            player_back_direction = "north"
            update_player_position("south")
            valid_choice = True
        elif direction_choice == "e" and can_east == True:
            player_back_direction = "west"
            update_player_position("east")
            valid_choice = True
        elif direction_choice == "w" and can_west == True:
            player_back_direction = "east"
            update_player_position("west")
            valid_choice = True
        else:
            print("Invalid Choice")
```

Next, I wrote the function to update the player's position.

```
def update_player_position(direction):
    global player_cols_position, player_previous_cols_position, player_rows_position, player_previous_rows_position
    player_previous_rows_position = player_rows_position
    player_previous_cols_position = player_cols_position
    if direction == "north":
        player_rows_position -= 1
    elif direction == "south":
        player_rows_position += 1
    elif direction == "west":
        player_cols_position -= 1
    else:
        player_cols_position += 1
```

When the player's position is updated, the map also needs to be updated. On the map, "P" represents the player, so we move the "P" to the new position. Also, we "discover" the adjacent tiles if they have yet to be discovered.

```
def update_player_map(map, player_map):
    player_map[player_rows_position][player_cols_position] =  "P"  #map[player_rows_position][player_cols_position]
    player_map[player_rows_position - 1][player_cols_position] = map[player_rows_position - 1][player_cols_position]
    player_map[player_rows_position + 1][player_cols_position] = map[player_rows_position + 1][player_cols_position]
    player_map[player_rows_position][player_cols_position - 1] = map[player_rows_position][player_cols_position - 1]
    player_map[player_rows_position][player_cols_position + 1] = map[player_rows_position][player_cols_position + 1]
```

The final part of the player navigation is to check if the player landed on the "ending" square.

```
def check_ending(end_rows, end_cols):
    if player_rows_position == end_rows and player_cols_position == end_cols:
        print("You have successfully reached the end of the dungeon. Along the way you f
        print("game over.")
        exit()
```

<u>Random encounters</u>:

The first function I needed to create random encounters is a calculator to determine if an encounter will happen. I decided to make random encounters a 15% chance every time the player moves to a room.

```
def encounter_calculator():
    will_fight = False
    number = random.randint(1, 100)
    if number <= 15:
        will_fight = True
    else:
        will_fight = False
    return will_fight
```

Next I created a function to build and print the encounter. It selects a player entrance, and an entrance for the monster. Then, it prints the encounter.

```
def encounter_builder(monster, weapon):
    player_entrances = ("as you look around the room, a ", "you charge into the room, but a
                        "you stumble clumsily into the room, a ", "you crawl into the room
    player_entrances_index = random.randint(0, len(player_entrances) - 1)
    new_player_entrance = player_entrances[player_entrances_index]

    monster_entrances = (" appeared!", " is standing in the middle of the room, ready to ba
                         " turns around slowly.....", " came from behind you!", " jumps dow
    entrance_index_number = random.randint(0, len(monster_entrances) - 1)
    new_entrance = monster_entrances[entrance_index_number]

    print(new_player_entrance + str(monster) +
          " holding a " + weapon + new_entrance)
```

Fight system:

To start creating the fight system I created the main battle loop function. The loop will run until either the monster dies, the player dies, or the player talks the monster out of fighting (more on that later).

```
def battle_loop(player_health, monster_health, player_action_word, monster_action_word, monster, player_weapon, monster_weapon):
    turn = "p"
    enemy_alive = True
    player_alive = True
    while enemy_alive == True and player_alive == True:
        if turn == "p":  # players turn
            damage = player_move(player_action_word, player_weapon)
            if damage == -1: #charisma option makes the damage -1
                break
            else:
                monster_health -= damage
                if monster_health <= 0:
                    enemy_alive = False
                    print("you won the battle")
                else:
                    turn = "e"

        else:
            player_health -= monster_move(monster_action_word,
                                          monster, monster_weapon)
            if player_health <= 0:
                player_alive = False
                print("You died.")
                exit()
            else:
                turn = "p"
    return player_health
```

Next, I created functions for both the player's turn, and the monster's turn. They are very similar, but the odds have been put in the favor of the player (The player typically does more damage). This also calculates if the attack will be a "critical hit" or a "miss". Critical hits have a 5% chance of happening, and misses have a 10% chance.

Player's move function:

```
def player_move(action_word, weapon):
    choice = " "
    while choice == " ":
        new_damage = 0
        choice = input("Select an option: fight (f), talk (t)")
        if choice == "fight" or choice.lower() == "f":
            damage_calculator = random.randint(0, 100)
            if damage_calculator < 10:
                new_damage = 0
                print("you " + action_word + " your " + weapon + ".")
                print("The attack missed: 0 damage.")
            elif damage_calculator >= 95:
                new_damage = 45
                print("you " + action_word + " your " + weapon + ".")
                print("The attack was a critical hit: " +
                      str(new_damage) + " damage!")
            else:
                # generates a random number between 1 and 5
                new_damage = random.randint(1, 30)
                print("you " + action_word + " your " + weapon + ".")
                print("The attack hit: " + str(new_damage) + " damage.")
        elif choice == "talk" or choice.lower() == "t":
            fight_over_check = charisma()
            print(" ")
            if fight_over_check == True:
                return -1
            else:
                pass
            pass  # remove later
        else:  # makes loop restart if an invalid choice is chosen
            choice = " "
    return(new_damage)
```

Monster’s move function:

```
def monster_move(action_word, monster, weapon):
    new_damage = 0
    damage_calculator = random.randint(0, 100)
    if damage_calculator < 10:
        new_damage = 0
        print("The " + monster + " " + action_word + "s their " + weapon + ".")
        print("The attack missed: 0 damage.")
    elif damage_calculator >= 95:
        new_damage = 15
        print("The " + monster + " " + action_word + "s their " + weapon + ".")
        print("The attack was a critical hit: " + str(new_damage) + " damage!")
    else:
        # generates a random number between 1 and 5
        new_damage = random.randint(1, 10)
        print("The " + monster + " " + action_word + "s their " + weapon + ".")
        print("The attack hit: " + str(new_damage) + " damage.")
    return(new_damage)
```

Merchant:

A second type of random encounter is the merchant. If an enemy does not spawn, there is a 5% chance that a merchant will. First, similar to the monster's encounter calculator, I created a calculator with a 5% chance to return that the merchant will appear.

```
def merchant_encounter_calculator():
    will_appear = False
    number = random.randint(0, 100)
    if number <= 5:
        will_appear = True
    else:
        will_appear = False
    return will_appear
```

Next, I created a function to ask the player If they would like to trade weapons with the merchant. If the player chooses yes, They will no longer have their current weapon, but instead, the weapon that the merchant gave them. If the player chooses no, they keep their current weapon.

```
def merchant(player_weapon, merchant_weapon):
    print("An old lady is sitting on the floor.")
    print("Hello, I am the dungeon merchant. Lets make a deal....")
    print("I'll trade you this " + merchant_weapon +
          " for your " + player_weapon + ".")
    print("Sounds like a good deal right?")
    print(" ")
    player_answer = input("enter your answer: (yes or no)")
    if player_answer.lower() == "yes" or player_answer.lower() == "y":
        print("hehehe")
        print(" ")
        print("You traded your " + player_weapon +
              " for a " + merchant_weapon + ".")
        return True
    else:
        print("maybe next time then hehehehe")
        print(" ")
        return False
```

Charisma system:

When a monster appears, the player will have the option to try to talk to the monster. If successful, the battle will stop and the monster will let you go. There are three types of conversations, each with a 33.33…% chance of happening. “Agreement” will have the monster agree to stop the fight, “Disagreement” will have the monster disagree that the fight should end. “Question” will have the monster ask the player a question, and generate a correct answer. The questions are yes or no based. If the player answer’s the question in a way that the monster agrees with, they will be allowed to pass peacefully. If the monster disagrees with the player’s answer, the fight will continue.

```
def charisma():
    possible_type_of_conversation = ("question", "agreement", "disagreement")
    type_of_conversation_index = random.randint(
        0, len(possible_type_of_conversation) - 1)
    conversation_type = possible_type_of_conversation[type_of_conversation_index]
    if conversation_type == "question":
        possible_question_intro = ("...fine, Maybe we can see if we find common ground. If we do, I'll let you walk out of here alive.",
                                   "I'll ask you once...", "this issue has been on my mind for a long time. If we agree I'll let you go
        possible_queston_intro_index = random.randint(
            0, len(possible_question_intro) - 1)
        question_intro = possible_question_intro[possible_queston_intro_index]
        print(question_intro)
        possible_questions = ("do you like the king's new policies?",
                              "Do you like pizza with pineapple on it?", "is water wet?", "Do you like the beach more than the mountains
        possible_questions_index = random.randint(
            0, len(possible_questions) - 1)
        question = possible_questions[possible_questions_index]
        print(question)
        possible_answers = ("yes", "no")
        possible_answer_index = random.randint(0, len(possible_answers) - 1)
        answer = possible_answers[possible_answer_index]
        player_answer = input("Enter your answer. (yes or no): ")
        if player_answer.lower() == "y":
            player_answer = "yes"
        elif player_answer == "n":
            player_answer = "no"
        if str(player_answer) == answer:
            possible_good_results = ("I'm glad we are on the same page, go on ahead.",
                                     "you and I are more alike than I thought. I apopogize for almost killing you")
            possible_good_result_index = random.randint(
                0, len(possible_good_results) - 1)
            good_result = possible_good_results[possible_good_result_index]
            print(good_result)
            return True
        else:
            possible_bad_results = ("...Then I have no choice...", "In that case, prepare to die!", "You monster.", ".....", "I'll n
                                    "you have disrespected me. Prepare to suffer the consequences.", "I thought maybe we could come to a
            possible_bad_result_index = random.randint(
                0, len(possible_bad_results) - 1)
            bad_result = possible_bad_results[possible_bad_result_index]
            print(bad_result)
            return False
    elif conversation_type == "agreement":
        possible_agreements = ("I didn't really want to fight you anyways.", "I'm more scared of you than you are of me, please, let me
                               "Wow...you actually tried to talk to me instead of trying to fight me. You have my respect, and I'll stay
        possible_agreement_index = random.randint(
            0, len(possible_agreements) - 1)
        agreement = possible_agreements[possible_agreement_index]
        print(agreement)
        return True
    else:
        possible_disagreements = ("I never back down from a fight!", "Sorry pal, but this is a fight.", "I'll make you sorry you ever ca
                                  "I cannot let you pass.", "don't talk me out of killing you!", "I remember what you did 3 years ago...
        possible_disagreement_index = random.randint(
            0, len(possible_disagreements) - 1)
        disagreement = possible_disagreements[possible_disagreement_index]
        print(disagreement)
        return False
```

Procedurally generated rooms:

The rooms have a description when the player enters them which is generated procedurally. I have a random number select the type of walls, then the type of floors, and finally a “decoration” for the room.

```
def room_generator():
    #floors and walls
    walls = ("brick", "stone", "marble", "ice",
             "stone bricks", "volcanic rock", "crystal")
    wall_index_number = random.randint(0, len(walls) - 1)
    new_wall = walls[wall_index_number]
    if new_wall == "brick" or new_wall == "stone bricks":
        floors = ("tile", "stone", "wood", "stone bricks",
                  "cracked stone bricks", "colorful tile", "cracked tile"
    elif new_wall == "stone":
        floors = ("stone", "sand", "rocks", "shallow water",
                  "stone, with stalagmites scattered around")
    elif new_wall == "marble":
        floors = ("marble", "cracked marble")
    elif new_wall == "volcanic rock":
        floors = ("volcanic rock", "burnt stone bricks")

    else:
        floors = ("ice", "cold shallow water", "stone", "crystal")
    floor_index_number = random.randint(0, len(floors) - 1)
    new_floor = floors[floor_index_number]
    # decoration
    decorations = ("there is a table in the middle of the room", "the floo
                   "there is writing in acient text on the walls", "the ro
    decoration_index_number = random.randint(0, len(decorations) - 1)
    new_decoration = decorations[decoration_index_number]
    # putting together the pieces
    print("the walls are " + new_wall + " the floor is " +
          new_floor + ". " + new_decoration)
    print(" ")
```

Building the game:

The next step is to use all the pieces shown above to build the game. First, I imported everything I needed from the other files.

```
from project_pkg.player_builder import Player, generate_player_motivation, generate_player_health
from project_pkg.weapon_generator import Weapon, make_weapon_type, make_weapon
from project_pkg.map_generator import Map, make_map, make_player_map
from project_pkg.encounter_generator import encounter_calculator, encounter_builder, merchant_encounter_calculator, merchant
from project_pkg.battle_loop import battle_loop
from project_pkg.monster_generator import monster_generator
from project_pkg.player_navigation import player_navigation, update_player_map, check_ending
from project_pkg.room_generator import room_generator
from project_pkg.welcome import welcome
```

Next, I created my objects.

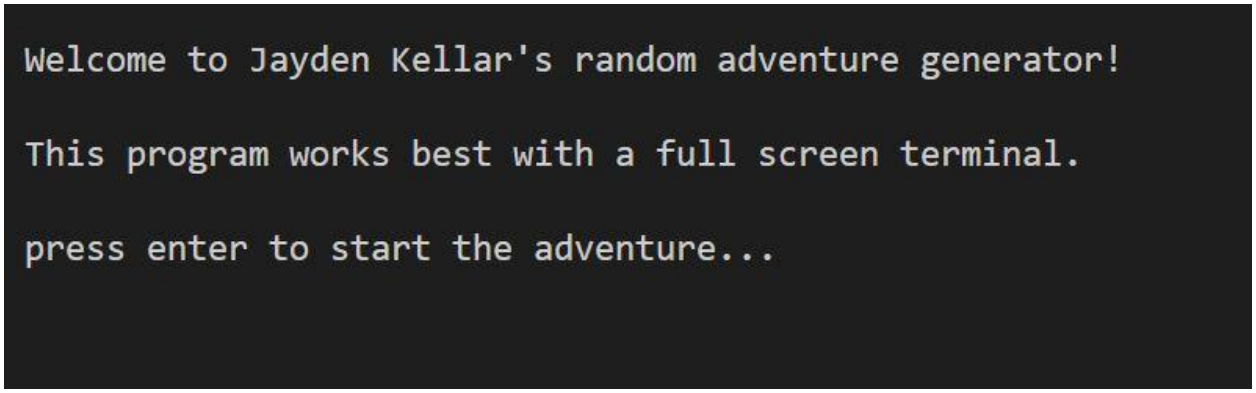
```
generated_player_health = generate_player_health()
generated_player_motivation = generate_player_motivation()
player = Player(generated_player_health, generated_player_motivation)
weapon_type = make_weapon_type()
player_weapon = Weapon(weapon_type, make_weapon(weapon_type))
map = Map(make_map())
player_map = Map(make_player_map())
in_combat = False
```

Then, I call my two functions that I made to be called before the game starts. Finally, I call the functions I created to Make the main game loop.

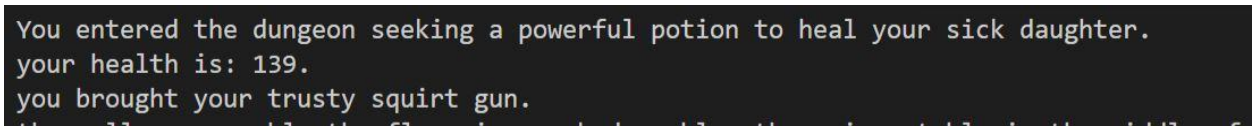
```
welcome()
player.introduce_player(player_weapon.weapon)
while player.check_alive() == True:
    room_generator()
    check_ending(map.get_end_rows(), map.get_end_cols())
    in_combat = encounter_calculator()
    if in_combat == True:
        generated_monster_weapon_type = make_weapon_type()
        weapon = Weapon(generated_monster_weapon_type,
                        make_weapon(generated_monster_weapon_type))
        monster = str(monster_generator())
        encounter_builder(monster, weapon.weapon)
        player.health = battle_loop(player.health, 100, player_weapon.get_action_word(
            player_weapon.weapon_type), weapon.get_action_word(weapon.weapon_type), monster, player_weapon.weapon, weapon.weapon)
    else:
        merchant_spawn = merchant_encounter_calculator()
        if merchant_spawn == True:
            merchant_weapon_type = make_weapon_type()
            merchant_weapon = Weapon(
                merchant_weapon_type, make_weapon(merchant_weapon_type))
            if merchant(player_weapon.weapon, merchant_weapon.weapon) == True:
                player_weapon = merchant_weapon
    update_player_map(map.array, player_map.array)
    player_map.print_map()
    player_navigation(map.array)
    player.print_health()
```

Gameplay

First, the player is welcomed and informed that the game is best played with a full screen terminal:

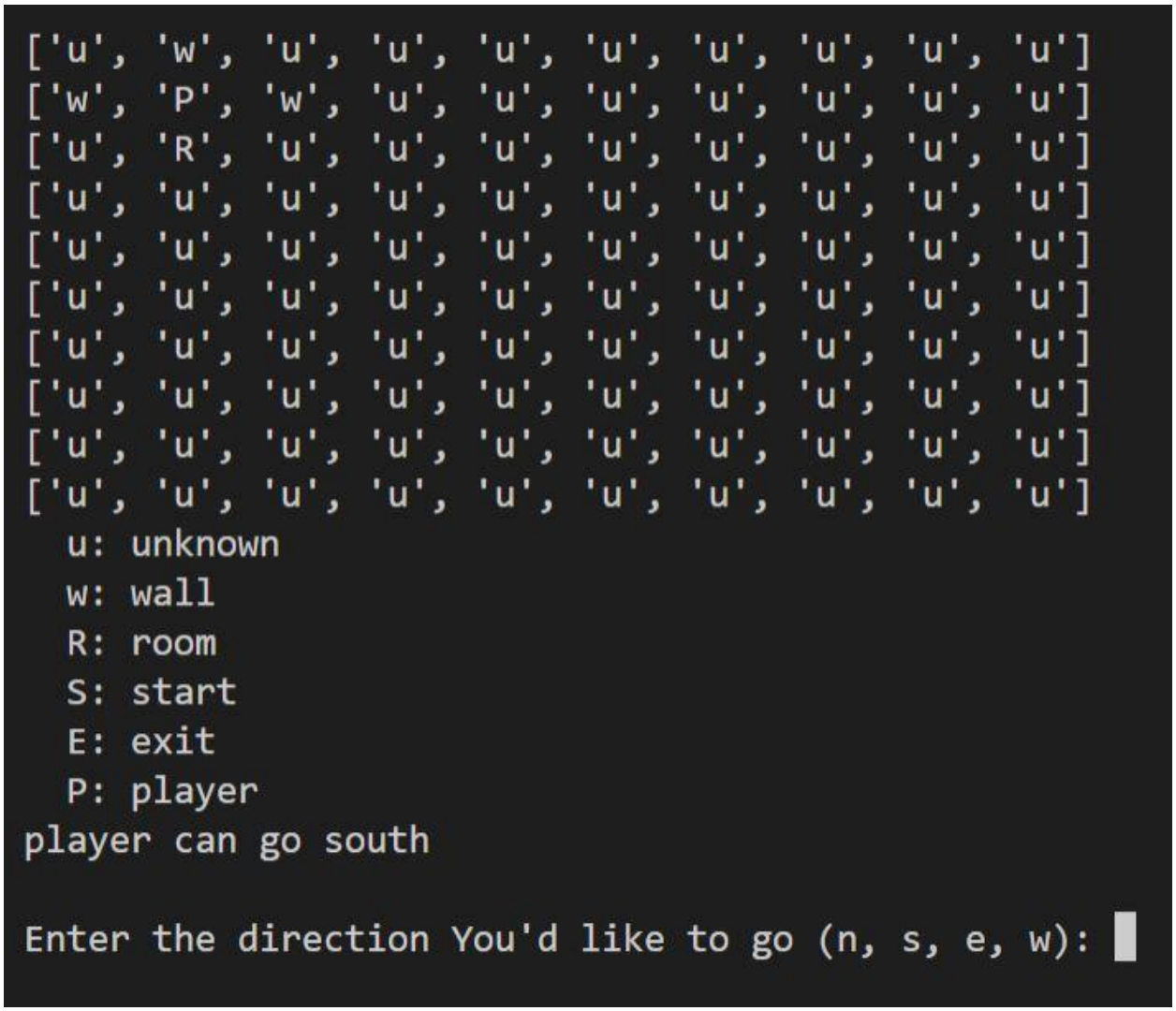


Next, the newly generated player character is introduced:



Next the first room is generated and printed:

Next, (if the first room is not an encounter) the player's map is displayed and the player is asked to pick an available option to navigate:



As the player navigates, new room dercriptions are generated, and the player's map updates as they explore:

```
Enter the direction You'd like to go (n, s, e, w): s
your health is: 139
the walls are stone bricks the floor is cracked stone bricks. this air in room is filled with bubbles

['u', 'w', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u']
['w', 'S', 'w', 'u', 'u', 'u', 'u', 'u', 'u', 'u']
['w', 'P', 'R', 'u', 'u', 'u', 'u', 'u', 'u', 'u']
['u', 'R', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u']
['u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u']
['u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u']
['u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u']
['u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u']
['u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u']
['u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u', 'u']
  u: unknown
  w: wall
  R: room
  S: start
  E: exit
  P: player
player can go east
player can go south
player can go north (back)

Enter the direction You'd like to go (n, s, e, w):
```

If the encounter calculator determines that an encounter will occur:

```
your health is: 139
the walls are ice the floor is stone. the floor has acient symbols, seemingly hand-carved centuries ago

you crawl into the room on your hands and knees, trying your best to be quiet, but a hungry vampire holding
 a flare gun appears out of thin air!
Select an option: fight (f), talk (t)
```

Fighting an enemy:

```
Select an option: fight (f), talk (t)f
you shoot your squirt gun.
The attack missed: 0 damage.
The hungry vampire shoots their flare gun.
The attack hit: 8 damage.
Select an option: fight (f), talk (t)
```

Talking to an enemy (shown type of conversation is “question”):

```
Select an option: fight (f), talk (t)t
...fine, Maybe we can see if we find common ground. If we do, I'll let you walk out of here alive.
do you like the king's new policies?
Enter your answer. (yes or no): no
I thought maybe we could come to an agreement....I was incorrect

The hungry vampire shoots their flare gun.
The attack hit: 10 damage.
Select an option: fight (f), talk (t)
```

The merchant encounter:

```
An old lady is sitting on the floor.
Hello, I am the dungeon merchant. Lets make a deal....
I'll trade you this great sword for your squirt gun.
Sounds like a good deal right?

enter your answer: (yes or no)yes
hehehe
```

Player death:

```
Select an option: fight (f), talk (t)f
you swing your great sword.
The attack hit: 11 damage.
The angry scarecrow swings their golf club.
The attack hit: 1 damage.
You died.
```

Reaching the end of the dungeon:

```
You have successfully reached the end of the dungeon. Along the way you found what you were seeking, but
you were lucky to escape with your life
game over.
```

Conclusion

<u>What I learned</u>:

This project was a fantastic learning experience! I learned a lot about using conditional logic to create rules for procedural generation. I also learned about 2D lists and how to use

them as a grid for a map. As an extension of this, I learned how to get the “coordinates” of the list, to use for the navigation of the player. Another thing I learned about was converting my program to object oriented programming, this proved to be challenging at first, but as time went on, I became more comfortable with how classes are made and I learned how useful they can be. Also, I learned about how to make a package to easily organize and access files.

Plans for the future:

I feel like this is the type of project that can be expanded endlessly. If I had more time/ If I come back to this project, I would add the following:

- A gold system (enemies would drop gold, and gold can be used at the merchant to acquire weapons instead of trading them.
- I would use a queue to save the descriptions of previous rooms, (currently when you enter a new room, even if the room is one that has already been visited, a new description for the room is generated).
- A healer (this would be a random encounter similar to the merchant).
- A better battle system with multiple types of moves.
- A better weapon generation system with different “tiers” of weapons. (also, I’d like to make the weapons responsible for the damage delt, as opposed to random number generation like how it currently is)