Algorithm and Programming Final Project

Project Name: “Untitled dungeon game”

Name: Osten Antonio

Student ID: 2802546115

Class: L1CC

Table of contents

A. Introduction

For my final project, I wanted to create something that I have not learned before, a game. I have always wanted to make an RPG game for a very long time, and I thought that this would be the perfect opportunity to learn and create it. The main inspiration behind this game is Soul Knight, a rogue-like game where you traverse through procedurally generated dungeons, with each level ending with a boss. However, while I initially planned to make a boss or a similar system to Soul Knight, I ended up making the dungeon generate infinitely, due to the time constraints. However, this allowed me to make a scoring and leaderboard system. I would end up naming the game “Untitled dungeon game” because I can’t think of any other name, since the game also ended up kind of generic.

The main gameplay of the game is to traverse between rooms, where when you enter a room, it will block all the entries and exit to another room until you kill all of the enemies in your current room. Additionally, there are two different classes of monsters, ranged and melee, with each class having different types of monsters with different characteristics (eg, one having a faster fire rate, lower damage and another with very slow fire rate and very high damage).

As for the player, they also have different classes with each having different perks (listed on the down, and also on the github page). These classes determine the playstyle that the player may take in when playing the game. The player can level up, which randomly upgrades their stats based on their difficulty (the harder, the higher the stats upgrade multiplier will be). In addition to the player’s melee attack, they also can shoot which has a cooldown.

Once completing a run of the game, the player will be rewarded with points, which can be used to upgrade their base stats in the shop. These points also determine the player’s position in the leaderboard.

A.1 Project link

https://github.com/osten-antonio/Algoprog-Final-Project

B. Design

Diagrams can be seen in the repository for a clearer picture.

B.1 Pseudocodes

The following are the pseudocodes for **the important** algorithms in the game:

B.1.1 Room generation

Main game loop

```
WHILE running
    # Moving rooms logic
    IF  player.hitbox overlaps current_room.border THEN
        IF player.hitbox is outside the left border THEN
            current_room.x = current_room.x - 1
        ELSE IF player.hitbox is outside the right border THEN
            current_room.x = current_room.x + 1
        ELSE IF player.hitbox is outside the top border THEN
            current_room.y = current_room.y - 1
        ELSE IF player.hitbox is outside the bottom border THEN
            current_room.y = current_room.y + 1
        END IF
    END IF

    # Room generation logic
    IF current_room is not in visited_rooms THEN
        generate_room(current_room)
    END IF
END WHILE
```

Generate room function

```
FUNCTION generate_room(current_room)
    adjacent_offsets = [
        (0, 1),   # Down
        (1, 0),   # Right
        (0, -1),  # Up
        (-1, 0),  # Left
    ]
    IF current_room is not in visited_room THEN
        visited_room.append(current_room)
    END IF



    FOR EACH offset IN adjacent_offsets
        adjacent_room = (current_room[0] + offset[0], current_room[1] + offset[1])
        IF adjacent_room is not in generated_room THEN
            calculated_offset = current_room.topleft + offset
            generate_room(calculated_offset)
        END IF
    NEXT offset
END FUNCTION
```

B.1.2 Drawing tilemap

```
FOR EACH layer IN tilemap_file
    FOR EACH pos, image IN LAYER
        drawn_layers["layer_name"].append(draw(image,pos))
    NEXT pos, image
NEXT layer
```

B.1.3 Enemy spawning

Main game loop

```
WHILE running
    IF current_room is not in visited_rooms AND current_room is not (0,0) THEN
        spawn_enemies(current_room)
    END IF
END WHILE
```

Spawn enemies function

```
FUNCTION spawn_enemies(current_room)
    max_enemies = 10
    FOR i=0 TO MAX_ENEMIES
        valid_position = False
        WHILE NOT valid_position
            chosen_enemy = random.choice(ENEMIES)
            topleft = current_room.topleft
            bottomright = current_room.bottomright
            pos = RANDOM(start = topleft - offset,end = bottomright - offset)
            temp = RECTANGLE(pos, TILESIZE)
            IF NOT temp overlaps walls THEN
                valid_position = True
            END IF
        END WHILE
        IF valid_position == True THEN
            spawned_enemy.append(chosen_enemy)
        END IF
    NEXT i
END FUNCTION
```

B.1.4 Main menu buttons and main menu

```
buttons = [play_button,store_button,class_button,quit_button]

# Note: The linking here has different implementation compared to actual program, but
logic remains the same

FOR i = 0 to 3
    IF i != 3 THEN
        buttons[i].next = buttons[i+1]
    ELSE
        buttons[3].next = buttons[0]
    END IF
NEXT i

FOR i = 3 to 0
    IF i != 0 THEN
        buttons[i].prev = buttons[i-1]
    ELSE
        buttons[0].prev = buttons[3]
    END IF
NEXT i

WHILE running
    IF key.pressed == TAB OR key.pressed == DOWN THEN
        current_button = current_button.next
    ELSE IF key.pressed == UP THEN
        current_button = current_button.prev
    ELSE IF key.pressed == MOUSE1  OR key.pressed == ENTER THEN
        current_button.action()
    END IF
    FOR EACH button IN buttons
        IF button.checkForHover(MOUSE_POS) THEN
            button.highlightColor(True)
            current_button = button
        ELSE
            button.highlightColor(False)
        END IF
    NEXT button
END WHILE
```

B.1.5 Leaderboard functions

Read leaderboard function

```
FUNCTION read_leaderboard()
    leaderboard = read(leaderboard.csv)
    leaderboard = sort(leaderboard)
    RETURN leaderboard.subArr(start = 0, end = 9)
END FUNCTION
```

Write score function

```
FUNCTION write_leaderboard(player_name)
    leaderboard = read_leaderboard()
    IF leaderboard.length < 10 OR player_score > leaderboard.score[9]
        leaderboard.add(player_data)
        leaderboard = sort(leaderboard)
        leaderboard.save()
    END IF
END FUNCTION
```

Display leaderboard function

```
FUNCTION draw_leaderboard()
    leaderboard = read_leaderboard()
    leaderboard = sort(leaderboard)
    place=0
    FOR EACH entry IN leaderboard
        screen.display_text(entry, x=screen.mid,y=place+offset)
        place=place+1
    NEXT entry
END FUNCTION
```

B.1.6 Enemy movement logic

Ranged enemy

```
movement_vector =(0,0)
IF timer_counter >= change_direction_timer THEN
    movement_vector += vector(random(-20,20),random(-20,20))
    timer_counter = 0
```

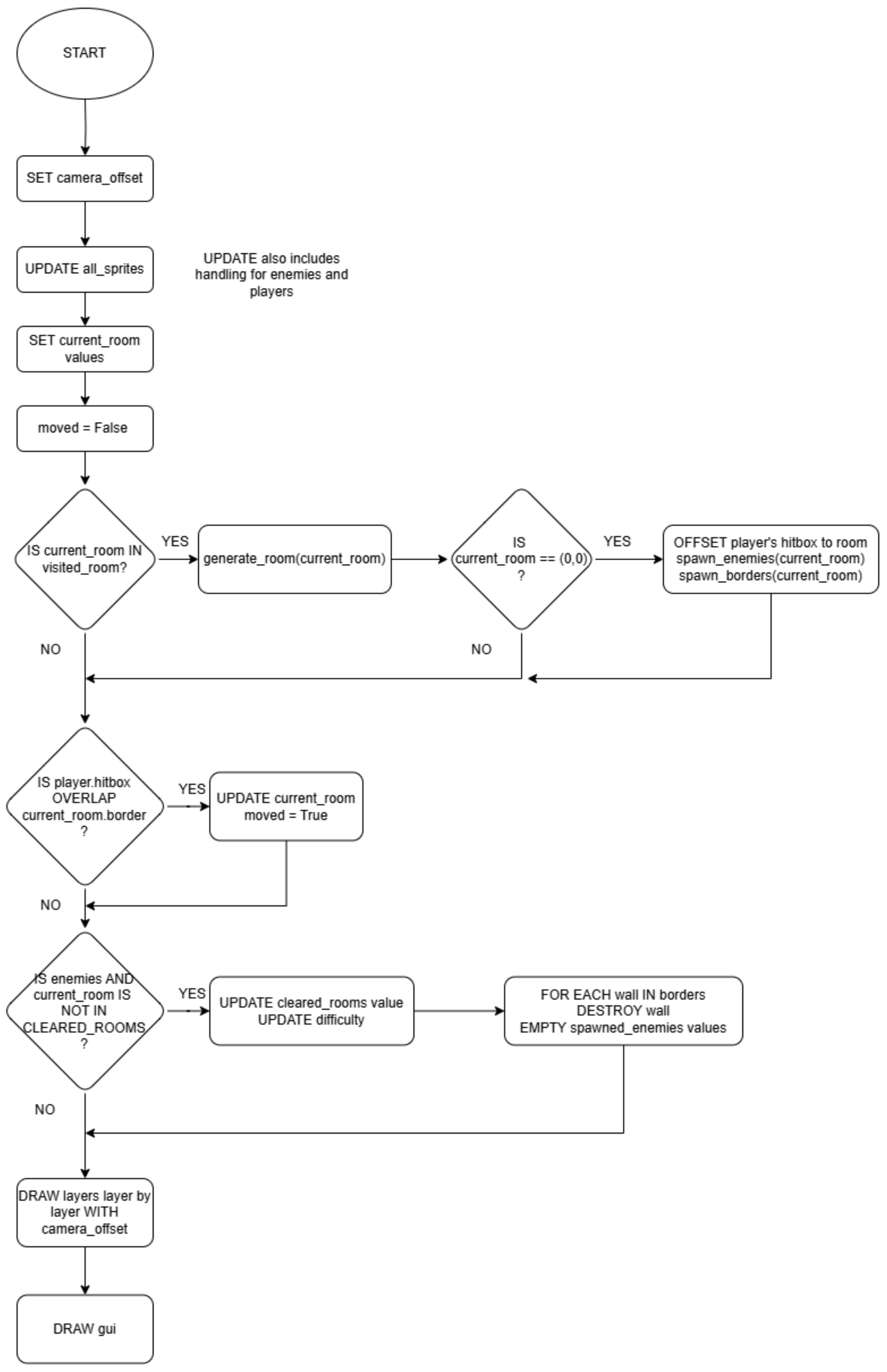
```
END IF
original_pos = hitbox_pos
hitbox.pos += movement_vector * speed
IF check_collision() THEN
    movement_vector = vector(random([-100, 100]), random([-100, 100]))
END IF
velocity =hitbox.pos - original_pos
ALIGN hitbox WITH sprite rectangle
```

Melee enemy

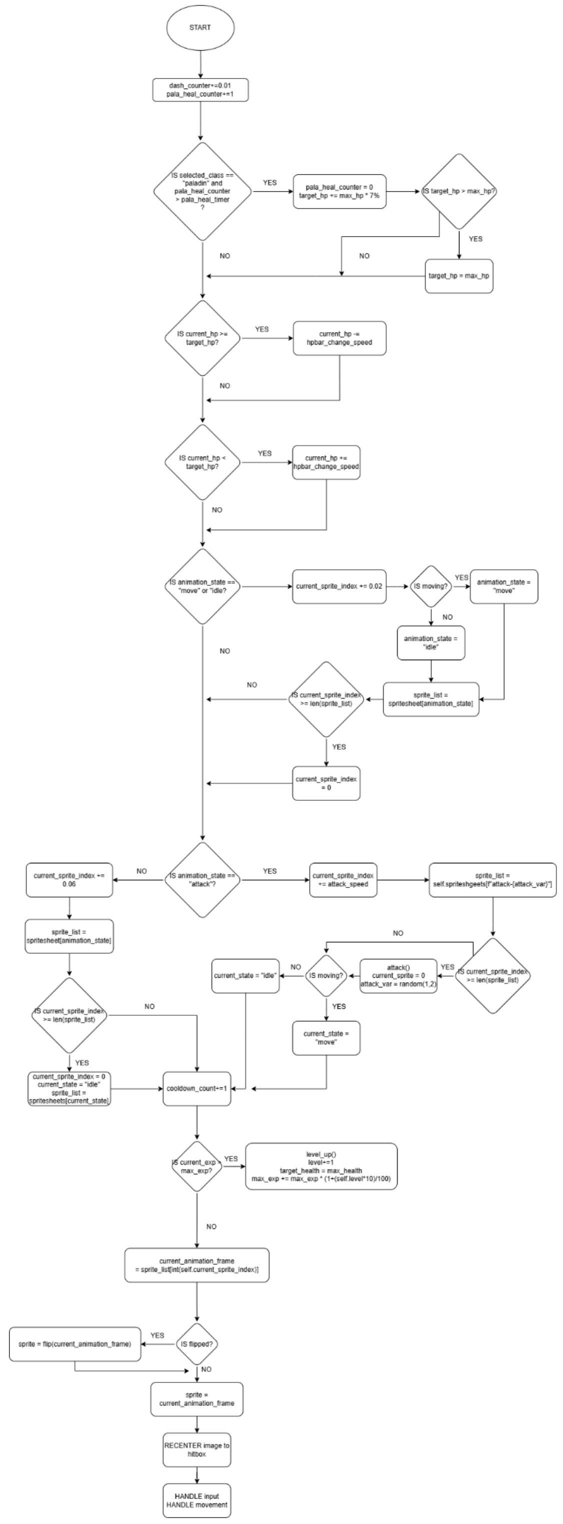
```
IF player.hitbox overlaps notice_range THEN
    distance = sqrt((player.hitbox.pos - hitbox.hitbox.pos)**2)
    IF distance > 0 THEN
        direction_vector = vector((player.hitbox.pos - hitbox.hitbox.pos)/distance)
    END IF
    hitbox.pos += direction_vector* speed
    check_collision()
    ALIGN hitbox WITH sprite rectangle
ELSE
    timer_counter += 1
    movement_vector =(0,0)
    IF timer_counter > change_direction_timer THEN
        movement_vector += vector(random(-0.1,0.1),random(-0.1,0.1))
        timer_counter = 0
    END IF
    original_pos = hitbox_pos
    hitbox.pos += movement_vector * speed
    check_collision()
    velocity =hitbox.pos - original_pos
    ALIGN hitbox WITH sprite rectangle
END IF
```

B.2 Flowcharts

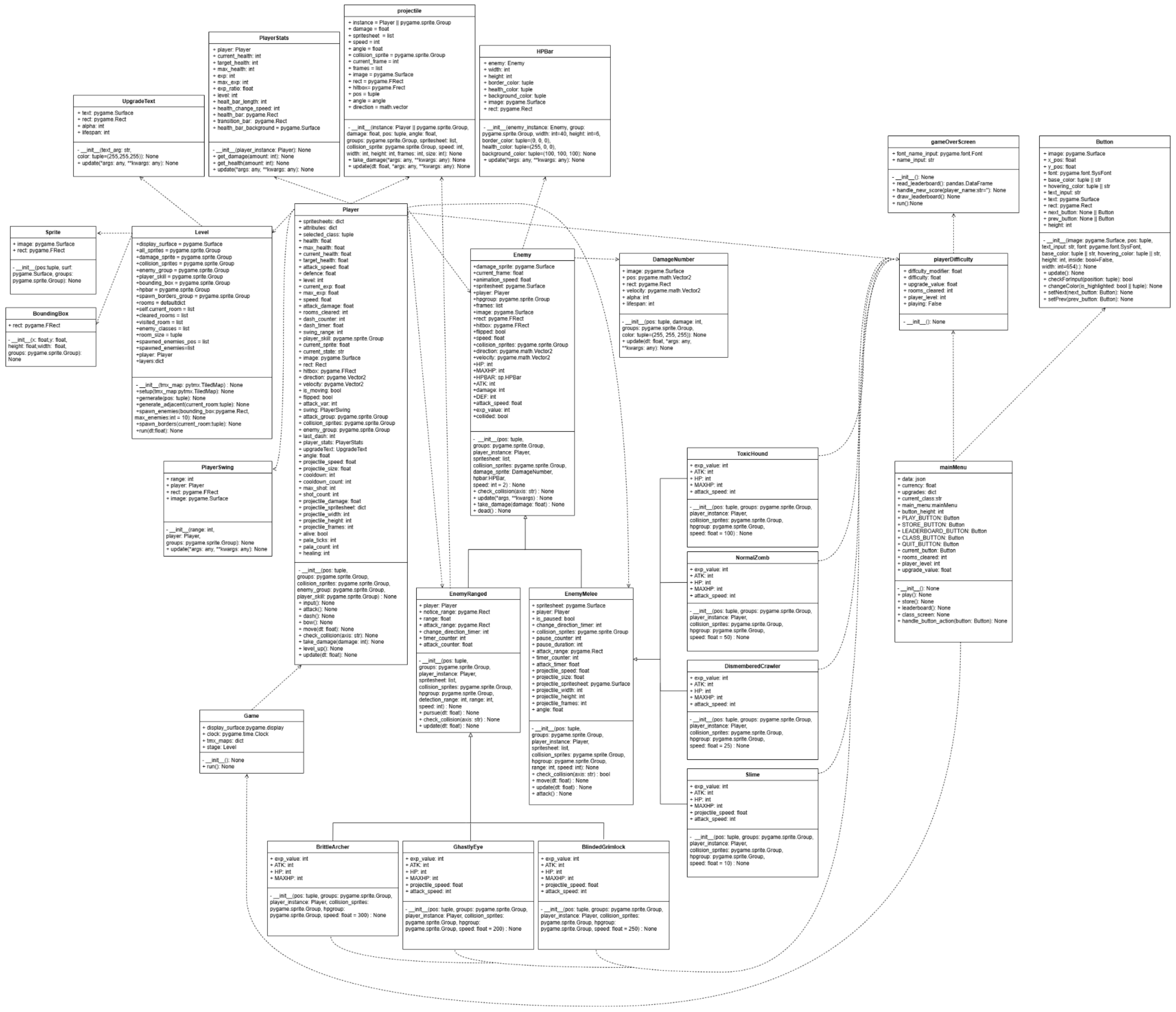
B.2.1 Main game loop


START
SET camera_offset
UPDATE all_sprites
UPDATE also includes handling for enemies and players
SET current_room values
moved = False
IS current_room IN visited_room?
YES
generate_room(current_room)
IS current_room == (0,0) ?
YES
OFFSET player's hitbox to room
spawn_enemies(current_room)
spawn_borders(current_room)
NO
NO
IS player.hitbox OVERLAP current_room.border ?
YES
UPDATE current_room
moved = True
NO
IS enemies AND current_room IS NOT IN CLEARED_ROOMS ?
YES
UPDATE cleared_rooms value
UPDATE difficulty
FOR EACH wall IN borders
DESTROY wall
EMPTY spawned_enemies values
NO
DRAW layers layer by layer WITH camera_offset
DRAW gui

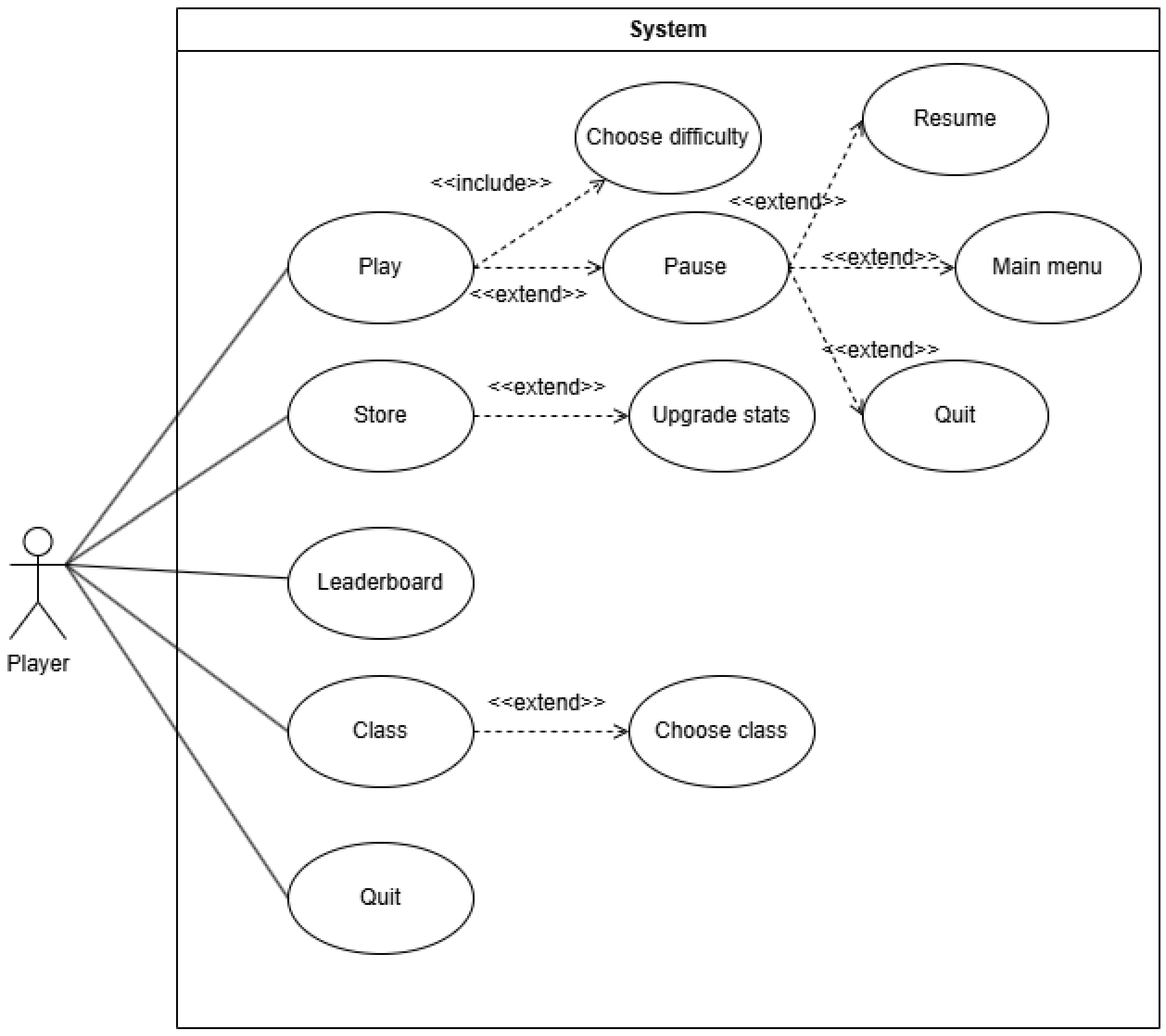
B.2.2 Player main update loop


START
dash_counter+=0.01
pala_heal_counter+=1
IS selected_class == "paladin" and pala_heal_counter > pala_heal_timer ?
YES
pala_heal_counter = 0
target_hp += max_hp * 7%
IS target_hp > max_hp?
YES
target_hp = max_hp
NO
NO
IS current_hp >= target_hp?
YES
current_hp -= hpbar_change_speed
NO
IS current_hp < target_hp?
YES
current_hp += hpbar_change_speed
NO
IS animation_state == "move" or "idle?
current_sprite_index += 0.02
IS moving?
YES
animation_state = "move"
NO
animation_state = "idle"
sprite_list = spritesheet[animation_state]
IS current_sprite_index >= len(sprite_list)
NO
YES
current_sprite_index = 0
NO
IS animation_state == "attack"?
YES
current_sprite_index += attack_speed
sprite_list = self.spriteshgeets[f"attack-{attack_var}"]
IS current_sprite_index >= len(sprite_list)
YES
attack()
current_sprite = 0
attack_var = random(1,2)
NO
IS moving?
NO
current_state = "idle"
YES
current_state = "move"
NO
current_sprite_index += 0.06
sprite_list = spritesheet[animation_state]
IS current_sprite_index >= len(sprite_list)
NO
YES
current_sprite_index = 0
current_state = "idle"
sprite_list = spritesheets[current_state]
cooldown_count+=1
IS current_exp > max_exp?
YES
level_up()
level+=1
target_health = max_health
max_exp += max_exp * (1+(self.level*10)/100)
NO
current_animation_frame = sprite_list[int(self.current_sprite_index)]
IS flipped?
YES
sprite = flip(current_animation_frame)
NO
sprite = current_animation_frame
RECENTER image to hitbox
HANDLE input
HANDLE movement

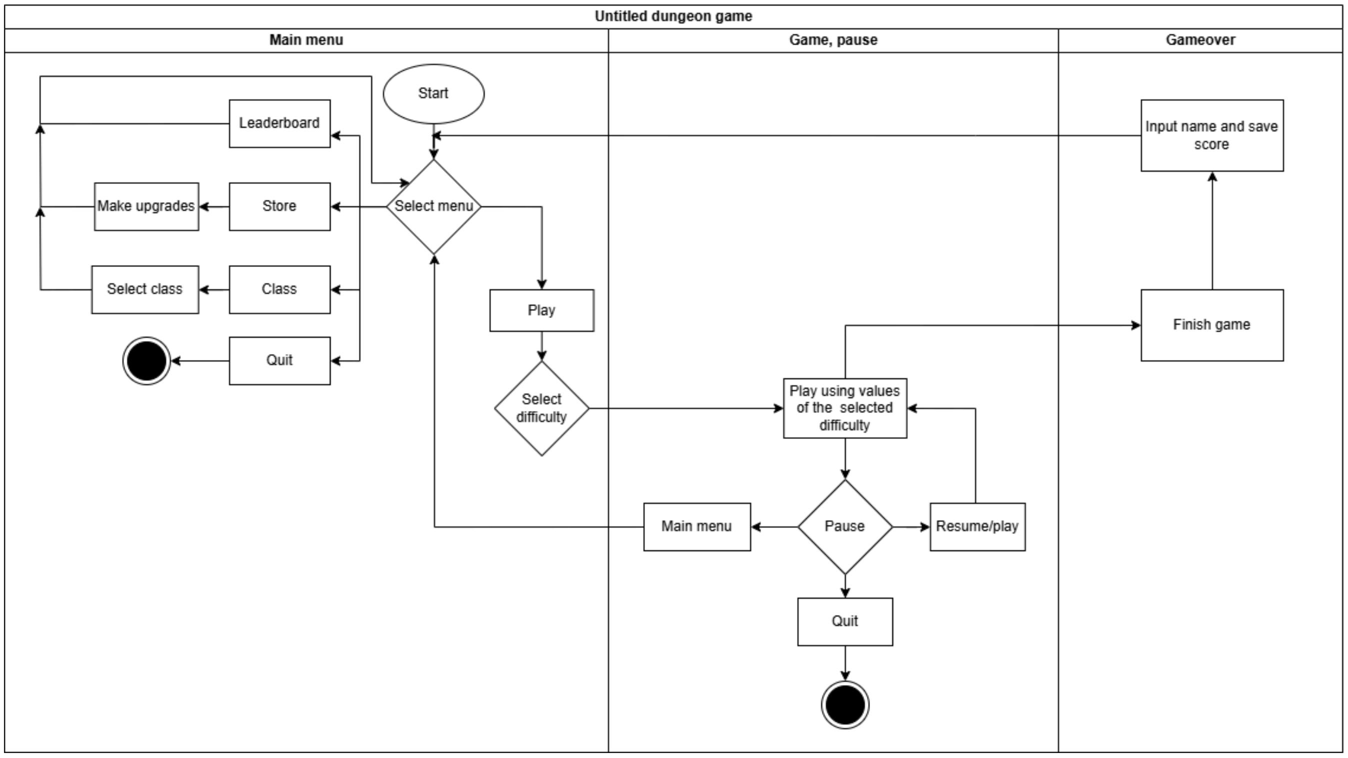
B.3 Class diagram


UpgradeText
+ text: pygame.Surface
+ rect: pygame.Rect
+ alpha: int
+ lifespan: int
- __init__(text_arg: str, color: tuple=(255,255,255)): None
+ update(*args: any, **kwargs: any): None
PlayerStats
+ player: Player
+ current_health: int
+ target_health: int
+ max_health: int
+ exp: int
+ max_exp: int
+ exp_ratio: float
+ level: int
+ healt_bar_length: int
+ health_change_speed: int
+ health_bar: pygame.Rect
+ transition_bar: pygame.Rect
+ health_bar_background = pygame.Surface
- __init__(player_instance: Player): None
+ get_damage(amount: int): None
+ get_health(amount: int): None
+ update(*args: any, **kwargs: any): None
projectile
+ instance = Player || pygame.sprite.Group
+ damage = float
+ spritesheet = list
+ speed = int
+ angle = float
+ collision_sprite = pygame.sprite.Group
+ current_frame = int
+ frames = list
+ image = pygame.Surface
+ rect = pygame.FRect
+ hitbox= pygame.Frect
+ pos = tuple
+ angle = angle
+ direction = math.vector
- __init__(instance: Player || pygame.sprite.Group, damage: float, pos: tuple, angle: float, groups: pygame.sprite.Group, spritesheet: list, collision_sprite: pygame.sprite.Group, speed: int, width: int, height: int, frames: int, size: int): None
+ take_damage(*args: any, **kwargs: any): None
+ update(dt: float, *args: any, **kwargs: any): None
HPBar
+ enemy: Enemy
+ width: int
+ height: int
+ border_color: tuple
+ health_color: tuple
+ background_color: tuple
+ image: pygame.Surface
+ rect: pygame.Rect
- __init__(enemy_instance: Enemy, group: pygame.sprite.Group, width: int=40, height: int=6, border_color: tuple=(0, 0, 0), health_color: tuple=(255, 0, 0), background_color: tuple=(100, 100, 100): None
+ update(*args: any, **kwargs: any): None
Sprite
+ image: pygame.Surface
+ rect: pygame.FRect
- __init__(pos:tuple, surf: pygame.Surface, groups: pygame.sprite.Group): None
BoundingBox
+ rect: pygame.FRect
- __init__(x: float,y: float, height: float,width: float, groups: pygame.sprite.Group): None
Level
+display_surface = pygame.Surface
+all_sprites = pygame.sprite.Group
+damage_sprite = pygame.sprite.Group
+collision_sprites = pygame.sprite.Group
+enemy_group = pygame.sprite.Group
+player_skill = pygame.sprite.Group
+bounding_box = pygame.sprite.Group
+hpbar = pygame.sprite.Group
+spawn_borders_group = pygame.sprite.Group
+rooms = defaultdict
+self.current_room = list
+cleared_rooms = list
+visited_room = list
+enemy_classes = list
+room_size = tuple
+spawned_enemies_pos = list
+spawned_enemies=list
+player: Player
+layers:dict
- __init__(tmx_map: pytmx.TiledMap) : None
+setup(tmx_map:pytmx.TiledMap): None
+gernerate(pos: tuple): None
+generate_adjacent(current_room:tuple): None
+spawn_enemies(bounding_box:pygame.Rect, max_enemies:int = 10): None
+spawn_borders(current_room:tuple): None
+run(dt:float): None
PlayerSwing
+ range: int
+ player: Player
+ rect: pygame.FRect
+ image: pygame.Surface
- __init__(range: int, player: Player, groups: pygame.sprite.Group): None
+ update(*args: any, **kwargs: any): None
Player
+ spritesheets: dict
+ attributes: dict
+ selected_class: tuple
+ health: float
+ max_health: float
+ current_health: float
+ target_health: float
+ attack_speed: float
+ defence: float
+ level: int
+ current_exp: float
+ max_exp: float
+ speed: float
+ attack_damage: float
+ rooms_cleared: int
+ dash_counter: int
+ dash_timer: float
+ swing_range: int
+ player_skill: pygame.sprite.Group
+ current_sprite: float
+ current_state: str
+ image: pygame.Surface
+ rect: Rect
+ hitbox: pygame.FRect
+ direction: pygame.Vector2
+ velocity: pygame.Vector2
+ is_moving: bool
+ flipped: bool
+ attack_var: int
+ swing: PlayerSwing
+ attack_group: pygame.sprite.Group
+ collision_sprites: pygame.sprite.Group
+ enemy_group: pygame.sprite.Group
+ last_dash: int
+ player_stats: PlayerStats
+ upgradeText: UpgradeText
+ angle: float
+ projectile_speed: float
+ projectile_size: float
+ cooldown: int
+ cooldown_count: int
+ max_shot: int
+ shot_count: int
+ projectile_damage: float
+ projectile_spritesheet: dict
+ projectile_width: int
+ projectile_height: int
+ projectile_frames: int
+ alive: bool
+ pala_ticks: int
+ pala_count: int
+ healing: int
- __init__(pos: tuple, groups: pygame.sprite.Group, collision_sprites: pygame.sprite.Group, enemy_group: pygame.sprite.Group, player_skill: pygame.sprite.Group) : None
+ input(): None
+ attack(): None
+ dash(): None
+ bow(): None
+ move(dt: float): None
+ check_collision(axis: str): None
+ take_damage(damage: int): None
+ level_up(): None
+ update(dt: float): None
Enemy
+damage_sprite: pygame.Surface
+current_frame: float
+animation_speed: float
+spritesheet: pygame.Surface
+player: Player
+hpgroup: pygame.sprite.Group
+frames: list
+image: pygame.Surface
+rect: pygame.FRect
+hitbox: pygame.FRect
+flipped: bool
+speed: float
+collision_sprites: pygame.sprite.Group
+direction: pygame.math.Vector2
+velocity: pygame.math.Vector2
+HP: int
+MAXHP: int
+HPBAR: sp.HPBar
+ATK: int
+damage: int
+DEF: int
+attack_speed: float
+exp_value: int
+collided: bool
- __init__(pos: tuple, groups: pygame.sprite.Group, player_instance: Player, spritesheet: list, collision_sprites: pygame.sprite.Group, damage_sprite: DamageNumber, hpbar:HPBar, speed: int = 2) : None
+ check_collision(axis: str) : None
+ update(*args, **kwargs) : None
+ take_damage(damage: float) : None
+ dead() : None
DamageNumber
+ image: pygame.Surface
+ pos: pygame.math.Vector2
+ rect: pygame.Rect
+ velocity: pygame.math.Vector2
+ alpha: int
+ lifespan: int
- __init__(pos: tuple, damage: int, groups: pygame.sprite.Group, color: tuple=(255, 255, 255)): None
+ update(dt: float, *args: any, **kwargs: any): None
gameOverScreen
+ font_name_input: pygame.font.Font
+ name_input: str
- __init__(): None
+ read_leaderboard(): pandas.DataFrame
+ handle_new_score(player_name:str=''): None
+ draw_leaderboard(): None
+ run():None
Button
+ image: pygame.Surface
+ x_pos: float
+ y_pos: float
+ font: pygame.font.SysFont
+ base_color: tuple || str
+ hovering_color: tuple || str
+ text_input: str
+ text: pygame.Surface
+ rect: pygame.Rect
+ next_button: None || Button
+ prev_button: None || Button
+ height: int
- __init__(image: pygame.Surface, pos: tuple, text_input: str, font: pygame.font.SysFont, base_color: tuple || str, hovering_color: tuple || str, height: int, inside: bool=False, width: int=654):): None
+ update(): None
+ checkForInput(position: tuple): bool
+ changeColor(is_highlighted: bool || tuple): None
+ setNext(next_button: Button): None
+ setPrev(prev_button: Button): None
playerDifficulty
+ diffculty_modifier: float
+ difficulty: float
+ upgrade_value: float
+ rooms_cleared: int
+ player_level: int
+ playing: False
- __init__(): None
ToxicHound
+ exp_value: int
+ ATK: int
+ HP: int
+ MAXHP: int
+ attack_speed: int
- __init__(pos: tuple, groups: pygame.sprite.Group, player_instance: Player, collision_sprites: pygame.sprite.Group, hpgroup: pygame.sprite.Group, speed: float = 100) : None
NormalZomb
+ exp_value: int
+ ATK: int
+ HP: int
+ MAXHP: int
+ attack_speed: int
- __init__(pos: tuple, groups: pygame.sprite.Group, player_instance: Player, collision_sprites: pygame.sprite.Group, hpgroup: pygame.sprite.Group, speed: float = 50) : None
DismemberedCrawler
+ exp_value: int
+ ATK: int
+ HP: int
+ MAXHP: int
+ attack_speed: int
- __init__(pos: tuple, groups: pygame.sprite.Group, player_instance: Player, collision_sprites: pygame.sprite.Group, hpgroup: pygame.sprite.Group, speed: float = 25) : None
Slime
+ exp_value: int
+ ATK: int
+ HP: int
+ MAXHP: int
+ projectile_speed: float
+ attack_speed: int
- __init__(pos: tuple, groups: pygame.sprite.Group, player_instance: Player, collision_sprites: pygame.sprite.Group, hpgroup: pygame.sprite.Group, speed: float = 10) : None
mainMenu
+ data: json
+ currency: float
+ upgrades: dict
+ current_class:str
+ main_menu:mainMenu
+ button_height: int
+ PLAY_BUTTON: Button
+ STORE_BUTTON: Button
+ LEADERBOARD_BUTTON: Button
+ CLASS_BUTTON: Button
+ QUIT_BUTTON: Button
+ current_button: Button
+ rooms_cleared: int
+ player_level: int
+ upgrade_value: float
- __init__(): None
+ play(): None
+ store(): None
+ leaderboard(): None
+ class_screen: None
+ handle_button_action(button: Button): None
EnemyRanged
+ player: Player
+ notice_range: pygame.Rect
+ range: float
+ attack_range: pygame.Rect
+ change_direction_timer: int
+ timer_counter: int
+ attack_counter: float
- __init__(pos: tuple, groups: pygame.sprite.Group, player_instance: Player, spritesheet: list, collision_sprites: pygame.sprite.Group, hpgroup: pygame.sprite.Group, detection_range: int, range: int, speed: int) : None
+ pursue(dt: float) : None
+ check_collision(axis: str) : None
+ update(dt: float) : None
EnemyMelee
+ spritesheet: pygame.Surface
+ player: Player
+ is_paused: bool
+ change_direction_timer: int
+ collision_sprites: pygame.sprite.Group
+ pause_counter: int
+ pause_duration: int
+ attack_range: pygame.Rect
+ timer_counter: int
+ attack_timer: float
+ projectile_speed: float
+ projectile_size: float
+ projectile_spritesheet: pygame.Surface
+ projectile_width: int
+ projectile_height: int
+ projectile_frames: int
+ angle: float
- __init__(pos: tuple, groups: pygame.sprite.Group, player_instance: Player, spritesheet: list, collision_sprites: pygame.sprite.Group, hpgroup: pygame.sprite.Group, range: int, speed: int): None
+ check_collision(axis: str) : bool
+ move(dt: float) : None
+ update(dt: float) : None
+ attack() : None
Game
+ display_surface:pygame.display
+ clock: pygame.time.Clock
+ tmx_maps: dict
+ stage: Level
- __init__(): None
+ run(): None
BrittleArcher
+ exp_value: int
+ ATK: int
+ HP: int
+ MAXHP: int
- __init__(pos: tuple, groups: pygame.sprite.Group, player_instance: Player, collision_sprites: pygame.sprite.Group, hpgroup: pygame.sprite.Group, speed: float = 300) : None
GhastlyEye
+ exp_value: int
+ ATK: int
+ HP: int
+ MAXHP: int
+ projectile_speed: float
+ attack_speed: int
- __init__(pos: tuple, groups: pygame.sprite.Group, player_instance: Player, collision_sprites: pygame.sprite.Group, hpgroup: pygame.sprite.Group, speed: float = 200) : None
BlindedGrimlock
+ exp_value: int
+ ATK: int
+ HP: int
+ MAXHP: int
+ projectile_speed: float
+ attack_speed: int
- __init__(pos: tuple, groups: pygame.sprite.Group, player_instance: Player, collision_sprites: pygame.sprite.Group, hpgroup: pygame.sprite.Group, speed: float = 250) : None

B.4 Use case diagram



B.5 Activity diagram



C. Implementation

C.1 Modules

The game is broken down into several key python files, each containing distinct functions, which are:

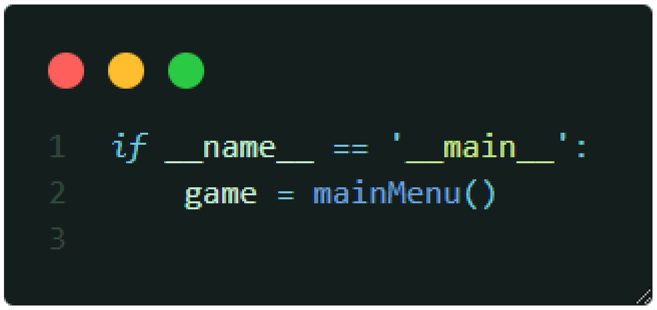
1. **settings.py**
   Contains all of the main constants used throughout the game, as well as the class which is used to store the player’s progress. This file also handles some of the imports which all of the other files use.
2. **main.py**
   Contains the game class, which initializes the game and contains the mainloop for the game.
3. **gui.py**
   Contains all of the GUI classes (Game over, main menu, pause, etc), which by extension also contains some of the main functions that the game uses to store and read data (Store, Leaderboard, Classes).
4. **entities.py**
   Contains all of the classes pertaining to the entities of the game, which are the player and the enemies. All of the player’s input is handled in the player class, as opposed to the main loop.
5. **sprites.py**
   Miscellaneous file which contains classes that manage the visual and interactive components of the game (such as the room’s bounding box, projectiles and HP bar).
6. **level.py**
   Most important file, this file contains a single class which manages the main game loop, and also main game logic.

C.1.1 Module functions and classes explanation

For more explanation of the code, please refer to the comments in the program

1. **settings.py**
    - **playerDifficulty**
      It only contains values that stores the stats regarding the player's current session, such as the difficulty modifier, and difficulty
2. **main.py**
    - **Game**
      The initialization contains the important initialization regarding the main game, including pygame's initialization, and also the level initialization.

      run: Contains the main loop regarding the main game (state.run), as well as checks if the player is alive and playing



```python
if __name__ == '__main__':
    game = mainMenu()

```

When the user runs the main.py file, it will initialize mainMenu class from gui.py, which initializes the Game class from main.py

3. **gui.py**
    - **gameOverScreen**

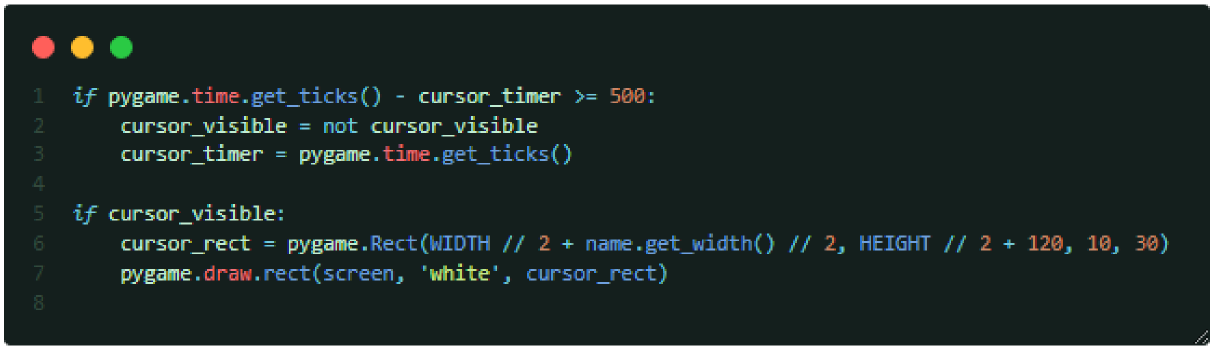
      Initializing, it reads the data of the current session, stored at the player_difficulty, which is of playerDifficulty class, stored at settings.py. Once read, it will store them to the local storage (player_data.json) to update their total_score to allow them to spend on the store.

      read_leaderboard: It reads the data from leaderboard.csv, sorts it by the ROOMS_CLEARED field, then returns the top 10 records.

      handle_new_score: It reads data from the current session, gets the leaderboard, and checks if the length of the leaderboard is less than 10, or if the current session has more score than the top 10th score, it adds a new record to the leaderboard and saves it.

      draw_leaderboard: it reads the top 10 records, using the read_leaderboard method, then iterates through every row in the record, and draws it using pygame's functions.It then displays the user's username input and returns it.

run (gameOverScreen's main loop): It displays the gui text, every loop, using draw_leaderboard to display the leaderboard.



```
if pygame.time.get_ticks() - cursor_timer >= 500:
    cursor_visible = not cursor_visible
    cursor_timer = pygame.time.get_ticks()

if cursor_visible:
    cursor_rect = pygame.Rect(WIDTH // 2 + name.get_width() // 2, HEIGHT // 2 + 120, 10, 30)
    pygame.draw.rect(screen, 'white', cursor_rect)

```

Every 500 ticks, it will display the cursor, and after another 500 ticks, it will not display it. After that, it handles the user's input such as backspace to delete, space to continue and also the keyboard input to enter their username.

- **get_font:** Returns pygame's font instance

- **Button**

  Upon initialization, it stores the arguments as instance variables, to be used in the creation of the button. If there is no image, then it will only display the button's text, and inside is used to determine whether the button is inside a container, if it is, it will display at an offset. Button uses a linked list data structure, to allow the player to navigate through buttons using their keyboards as opposed to their mouse.

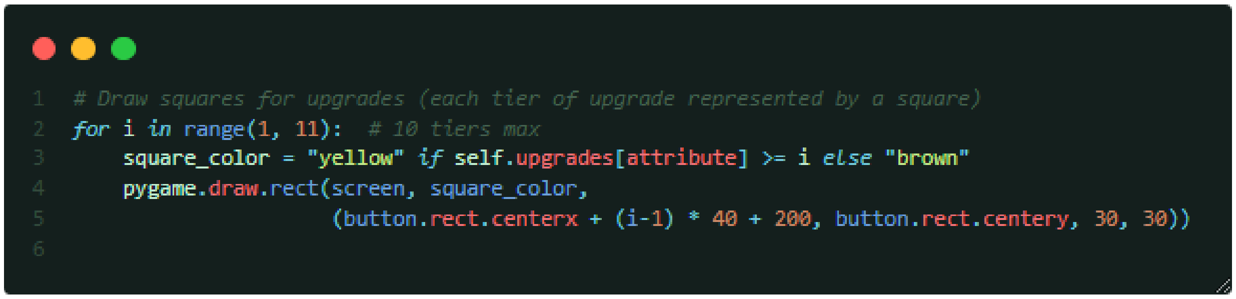
  update: Displays the button every tick
  checkForInput: returns true if the mouse position is inside the confines of the button
  changeColor: Setter function, checks whether the argument is a bool or a list, if it is a bool (meaning that the button is traversed using keyboard/linked list), it directly checks if it is true, however, if it is a list (meaning the the button is traversed using mouse), it checks if it inside the button confines. Both functions change the active button.

setNext, setPrev: Setter function used to set the next and previous button to form a circular linked list.

- **mainMenu**

  It initializes the main menu, handles button navigation for the menus, displays the menu's graphical interface, and handles user inputs for navigating through various options such as starting the game, accessing the store, leaderboard, and more. It first loads the player data from player_data.json to be used in other menus such as store, and class.
  It also initializes the buttons linked list, and starts the main loop for the main menu, which consists of checks and display.

  save: Used to save the player's data such as upgrades and selected class to player_data.json
  play: Displays a new menu by running a new loop for difficulty level selection, and imports Game from main to start the game using the difficulty. The value of the difficulty is saved on the current session at settings.py.
  store: Displays a new menu by running a new loop to allow the players to interact and get upgrades. It handles player's input by using button's functions, and loops between upgrade attributes displaying the square as yellow if it is upgraded to the value, and brown for the rest of the values. It allows the player to upgrade their attribute by incrementing the attribute in the local file by using save(), and also updating it in the current process variable.



```python
# Draw squares for upgrades (each tier of upgrade represented by a square)
for i in range(1, 11):  # 10 tiers max
    square_color = "yellow" if self.upgrades[attribute] >= i else "brown"
    pygame.draw.rect(screen, square_color,
                     (button.rect.centerx + (i-1) * 40 + 200, button.rect.centery, 30, 30))

```

Drawing squares for each attribute in the store, it will only draw yellow if the upgraded attribute exceeds the current index.

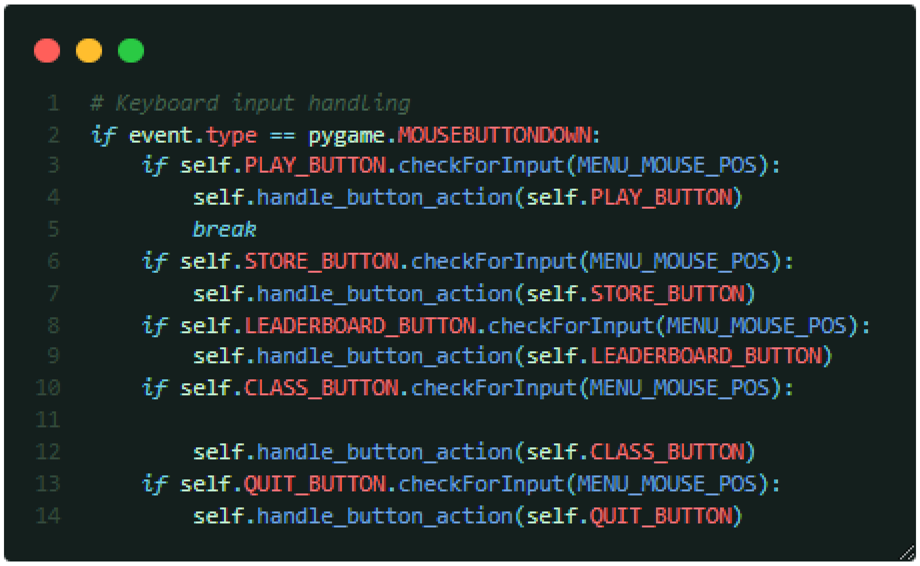
leaderboard: It uses the same logic as gameOverScreen's leaderboard display but this time, it only displays the leaderboard. It also has a separate new loop for this menu, similar to other menus.

class_screen: Displays a new menu by running a new loop to allow the players to select their class, handling the player's input by using the button's function and saving their class to the local storage using save() and also updating it in the current process variable.

handle_button_action: Name is self explanatory, it handles the button's action by checking the type of button that is passed as the argument. It is used to display other menus, or exit the program.

* **Note:** Every other menu in the main menu consists of the same button logic in their loops, which is to check for input if they click on it. If they do, they checkForInput will be called for the buttons and if it returns true, it will handle the selected buttons actions. In the main menu especially, it uses the current_button variable to check for the current_buttons highlighted (to utilize the linked list), and current_buttons is updated every time the user clicks tab, up or down. Similarly, when pressing enter it calls handle_button_action on the current button, instead of checkForInput.

Example of Note:



```python
# Keyboard input handling
if event.type == pygame.MOUSEBUTTONDOWN:
    if self.PLAY_BUTTON.checkForInput(MENU_MOUSE_POS):
        self.handle_button_action(self.PLAY_BUTTON)
        break
    if self.STORE_BUTTON.checkForInput(MENU_MOUSE_POS):
        self.handle_button_action(self.STORE_BUTTON)
    if self.LEADERBOARD_BUTTON.checkForInput(MENU_MOUSE_POS):
        self.handle_button_action(self.LEADERBOARD_BUTTON)
    if self.CLASS_BUTTON.checkForInput(MENU_MOUSE_POS):

        self.handle_button_action(self.CLASS_BUTTON)
    if self.QUIT_BUTTON.checkForInput(MENU_MOUSE_POS):
        self.handle_button_action(self.QUIT_BUTTON)
```

4. **entities.py**
    - **get_frames**

      A function with parameters to cut each sprite from a spritesheet, modify it by scaling it, and display it on screen, then return that sprite of spritesheet to be use in the mainloop, either by modifying it further (such as flipping it), and displaying it again for each tick of the mainloop.

    - **Player**

      Initializing, it inherits from the pygame's sprite class, it reads the player data from player_data.json to assign the starting value with upgraded value. Similarly, it stores the player sprites in a dictionary with all of the states of the player to be used in the program. With the starting values, it assigns the player stats as an instance variable to be used in the calculations of the program, and it will assign different starting values depending on the class (which is determined by the selected_class stored in the player_data.json). The instance variable includes the player's stats, hitbox, swing detection rectangle and also the spritesheets of the states, etc.

input: this function handles the player’s input and contains an instance of the pause menu in case they press escape or pause. This function will run every tick in the main loop, and contains all of the necessary inputs such as movement, and attacks. The player’s look angle, which is determined by their movement vector, is only updated if they are not holding alt. This angle is used to determine where the player swing is, and where the projectile should travel to.
attack: This method gets every group that collides between the player’s swing range (another class), and it checks every group that collides if it is an instance of enemy, if it is, and the swing collides with the enemies hitbox, it will run the function to allow the enemy to take damage. This damage is halved if you are an archer (since this method is for the melee attacks).
dash: This method allows the player to dash a distance according to their speed and their look direction. It first resets the last_dash value, used to determine when the player has last dashed (used to calculate the invincibility frame). The direction is determined by the angle, taking the vertical and horizontal components of the angle, and it moves the player a distance by using the same logic as the movement method, but instead of updating the player’s movement once every tick, this updates it 20 times to simulate a dash.
bow: This method is the player’s range attack. It checks if the current cooldown_count is over the cooldown, if it is over, meaning that the player has waited over the cooldown amount, it resets the cooldown_count, then it changes the player’s state to bow (to change the sprite sheet), and it will loop from 1 to 601, with 200 as the step, and this loop value will be used as the projectile’s velocity or speed. Additionally, if the player’s selected class is an archer, it loops from -1 to 1, with the step of 2, to provide for the angle for the additional 2 streams of arrows. This value will be used for the angle change (in radians), and the same loop logic will be used as the

base range attack (loop from 1 to 601, with step 200), except there will be additional angle change using the first loop’s value.

move: This method provides the main movement logic for the player. It gets the current player’s original position to calculate the velocity. Then modifies the player’s x hitbox value, checks for collision in the horizontal direction, then modifies the player’s y hitbox value, checks for collision in the vertical direction. It then updates the velocity using the original position and the current position. It also updates the player’s sprite rectangle to the player’s hitbox.

check_collision: This method takes the axis as an argument. It checks for every sprite in the collision group if it is colliding with the player’s hitbox. Then according to the axis, it uses the player’s movement vector to check whether, eg, if the player’s hitbox right side is more than the collision sprite’s left side, if it is then it sets the player’s right side to the collision sprite’s left side, effectively, limiting the player’s movement to the collision sprite. This is if the player is moving right, this direction is determined by the direction vector value. After every collision check it sets the player's velocity to 0.

take_damage: It first checks if the player has dashed in the last 300 ticks, if they haven't, then they will take damage. It first changes the player’s state to “hurt” to change the player’s sprite. Then it reduces the player’s target_health (for the health bar). It then checks if the target_health is less than 0, and sets it to 0, if it is. This is done to prevent the HP bar from going over the limit. Similarly, if the current_health is less than or equals to 0, it sets the current_state to “dead”, to animate the death animation, and and sets teh alive to false to break the main game’s loop. Then finally, it runs an instance of the game over screen.

level_up: Once this method is run, it increments the player level, then from a selected stat it randomly chooses 3 (which will be appended to a list), to be upgraded, using the setattr function to change player's instance value equivalent of the stat by 10% of its value. Then finally, it sets the upgrade text to the proper value from the upgraded list.

update: This method contains all of the necessary information to update the player's value, animating the player's sprite, incrementing the cooldown count, and performs several checks. The move and input method is also performed here. The complete process is displayed in the flow chart, **B.2.2**.

- **Enemy**

  Initializing, it inherits from the pygame's sprite class, then initializes the necessary attributes for the enemy from the arguments, such as their hp, sprite sheet, hitbox, etc similar to the player.

  check_collision: This method is exactly similar to the player's check collision method, except it updates the collision value, setting it to false when the method is called, and setting it to true when a collision is true.

  update: This method is called in the game's main loop, it calls the HPBAR's update method, sets the animation logic, and flips if sprite if it is on the right side or left side of the play.

  take_damage: This method reduces the enemy's HP by the damage, instantiates the DamageNumber class to display the damage taken by the enemy then checks if the HP is less than 0. If it is, it calls the dead method.

  dead: On a random chance, chances are 50% by default, 70% for warriors, heals the player by 7%. To heal, it adds 7% of max health to target health. If the target health is over the max health it sets it to the max health to avoid the player's health bar from exceeding the limit. Then it increments the player's current exp by the exp value of the enemy,updates HPBar, and removes itself from the main loop updates.

- **EnemyRanged**

  Inherits from Enemy class, it initializes the necessary attributes for the enemy from the arguments, except there are additional projectile values.

  check_collision: This method calls the check_collision from the Enemy class, but additionally, if it collided (defined by self.collided), it changes the movement direction value to a random vector between -100 and 100. It also sets the is_paused to true if collided.
  move: This method is similar to the player's movement method, however, it relies on random choice to determine the movement vector. If the movement timer is greater or equal to the change_direction_timer, it sets the is_paused value to true then randomly chooses the movement vector, resets the movement timer. The next steps are exactly similar to the movement logic for the player (updating the hitbox, checking collision in the axis, setting the velocity, and realigning the sprite rectangle with the hitbox).
  update: It inherits the update method from Enemy, except if is_paused is true, it waits (pause_counter >= pause_duration), then resets is_paused and collides as well as the pause counter. This pause_counter is incremented only if is_paused is true. Additionally, if the player is colliding with their attack range, it will perform the attack method, which will fire projectiles in the player's direction.
  attack: Using the distance between the player's hitbox to the enemy's center, it calculates the angle of where the projectile should be fired. It also increments the attack_timer and once it exceeds the attack speed, it is resetted and the projectile class will be called with the angle, as well as the player passed as an argument for the instance type (to check on who to reduce the HP of). Then, it recenters the attack_range to the hitbox.

- **EnemyMelee**

  Inherits from Enemy class, it initializes the necessary attributes for the enemy from the arguments, except there will be an additional value for the notice_range, which is going to be used to determine when the enemy should pursue the player.

  pursue: It first calculates the distance between the player and the enemy, if it is more than 0, then it sets the movement direction vector to be the change between the player and enemy over the distance. It then uses the same movement logic as all other entities.

  update: It inherits the update method from Enemy, but it adds an additional logic, where if the player is within the notice range, then it executes the pursue method to move the enemy to the player. If it is not, then it does the same movement logic as the ranged enemy. Additionally, it will keep incrementing the attack_counter by 0.1 every tick, and if it is equal or greater than the attack_speed, it will reset it and check if the player's hitbox is colliding with the enemy's attack range. If it is, then it will call the player's take_damage method. Then it recenters the attack_range, and notice_range to the enemy's hitbox.

- **BrittleArcher, GhastlyEye, BlindedGrimlock**

  It inherits from the EnemyRanged, then it changes the necessary values with the difficulty. This difficulty will keep on increasing every room cleared. More information can be seen in the class diagrams.
- **ToxicHound, NormalZomb, DismemberedCrawler, Slime**

  It inherits from the EnemyMelee, then it changes the necessary values with the difficulty. This difficulty will keep on increasing every room cleared. More information can be seen in the class diagrams.

5. **sprites.py**
    - **Sprite**
      This is the class to display every tiles in a tile map, it inherits from the pygame's sprite class, and sets the image to be 4* of the original tile size. From the arguments, it also sets the tile's position by getting the image's rectangle and setting its center to the position.
    - **BoundingBox**
      It inherits from the pygame's sprite class to check when the player crosses over the room boundary, and it only sets the room's boundary rectangle.
    - **PlayerSwing**
      This is the class to determine where the attack collision should be checked at, as well as to display visually where the player is looking at. It takes the player as the argument to be able to check where it is supposed to be positioned at.

      update: This method runs every tick in the mainloop, it checks if the player is holding the alt key, and only changes the angle if the player is not holding the alt key. Then it positions itself and changes the sprite image depending on that angle.
    - **DamageNumber**
      This is the class to display the damage number that the enemies take. Initializing it, it renders the damage taken at the enemy's hitbox. As Well as setting the alpha value (transparency), which is set to 100% visibility at the start.

      Update: Every tick, it will move the damage number up by the amount of velocity. Then it reduces the alpha by an amount determined by dividing the maximum amount of the alpha (255) by the lifespan. It checks if the alpha is less than or equal to 0, where if it is, it is removed from the main update loop. Finally, it sets the image to the alpha amount. This makes the method run every tick of the main loop, and gives the illusion that the damage number is moving upwards and fading out.

- **projectile**

  This is the class to create a projectile, and check if it collides with the player or the enemy. Using the arguments, it sets the necessary instance variables. The variable, instance (self.instance), can be a player or a sprite group.

  take_damage: This method performs nothing, and is only there to allow for polymorphism as this class is added to the same group as player and enemy.
  update: using the direction value, it will move the projectile until it reaches a wall or a player or an enemy. Every update loop, it increments the hitbox with the movement vector and realigns the sprite rectangle with the hitbox. Then it iterates between the collision sprites to check for collisions. If there is a collision it will remove itself from the update loop. It also checks for the instance type, where if the instance type is a group (meaning that it is an enemy), it will iterate through every enemy in the group, and check if the enemy's hitbox collides with the projectile's hitbox. If it does, it will remove itself from the update loop, and call the take_damage method for the enemy with the damage value. If it is not a group, then it is a player instance, to which it will immediately perform the same logic as the enemy without iterating.

- **HPBar**

  This class is used to draw the health bar for the enemy. Initializing it is similar to the other class (setting necessary values using the arguments).

  update: This method runs every update loop. It first recenters the entire HPbar to the enemy's mid top, but with a -10 vertical offset. It then calculates the enemy's health percentage, and checks if it is equal to or less than 0. If it is, it will remove itself from the main update loop. Then it calculates the current health_width to be displayed above the enemy. Then it draws it with a border (using two rectangles).

- **PlayerStats**

  This class is used to draw the player's stats which is displayed at the top left of the screen. Initializing it is similar to the other class (setting necessary values using the arguments), however it loads the background from a png file.

  get_damage: This method will reduce the target_health by a certain amount if it is more than 0, and set the target_health to 0 if it is less than 0.
  get_health: This method will increase the target_health by a certain amount if it is less than the max_helath, and set the target_health to the max health if it exceeds the max health.
  update: This method will run every update loop, and will only change the instance variabel to the newest value. This class is drawn on the main loop not here, unlike the other display class.

- **UpgradeText**

  This class is similar to the DamageNumber class, where it takes a string argument to be displayed. Initialising assigns the appropriate value to the appropriate instance variable. As the name suggests, it will only be instantiated when the player level ups.

  Update: Every tick, it will reduce the alpha by an amount determined by dividing the maximum amount of the alpha (255) by the lifespan. It checks if the alpha is less than or equal to 0, where if it is, it is removed from the main update loop. Finally, it sets the image to the alpha amount. THis will run every tick of the main loop, and gives the illusion that the level up text is fading out.

6. **level.py**

- **Level**

  This class contains the main loop for the game, and initializing it will combine all of the enemies from the enemy class into a list to be randomly chosen, creates all of the essential sprite group for the other class or sprites, creates all of the necessary list, and sets the current_room the player is on (which is initialized as 0,0 or the origin), as well as running the setup method.

  setup: It sets up the dictionary of all of the layers from the tile map, and with the starting room, it draws every layer tile by tile, appending it to the appropriate key at the dictionary. It also creates the bounding box for the starting room, appending it to the rooms list. THe player class is also instantiated here.
  generate: This method generates a random room from a given position. It chooses between a range of possible rooms, which is stored in a dictionary, and it then is displayed layer by layer with the same drawing step as setup.

generate_adjacent: This method is used to check whether the adjacent rooms are generated or not, it first checks if the current room is not in the visited room, if it is then it will append to the visited rooms. With the list of offsets, it generates a room using the generate method, passing the calculated position with the offsets, only if the adjacent room is not generated. This method is shown in the pseudocode for a simpler understanding.

spawn_enemies: This method is used to spawn enemies within the confines of the room. It basically tries to spawn enemies within a certain number of attempts, which every attempt will randomly choose the coordinates of the enemy within the confine of the room. If it is valid, then it will instantiate the enemy by appending the randomly chosen enemy class to the relevant enemy lists. It also adds each enemy to the enemy_group.

spawn_borders: This method is used to spawn borders to prevent the player from moving to the next room if all of the enemies in a room are not killed yet. It loads the tilemap of the borders, and appends it to the spawn_border layer in the layers dictionary, tile by tile.

run: This method is the main game loop, more info is displayed on the flow chart. In this method, it also draws everything layer by layer, performs the game logic (checking if the player has left a room, or calculating the player's current room, etc). All of the sprites are also updated here, along with the PlayerStats class.

* To remove from the update loops effectively stops it from being displayed, and updated.

* Additional explanation is also made in the comments.

C.2 Data storage

In addition to those, there are also separate files used to store the game data, which are:

1. **leaderboard.csv**
   Contains all of the runs that the player has done, storing their name, level and points.

   File structure:

NAME	LEVEL	ROOMS_CLEARED
________________	________________	________________

2. **player_data.json**
   Contains all of the data related to the player and game such as their currently selected class, and the upgrades.

   File structure:

   ```
   {
   "total_score": FLOAT,
    "selected_class": STR,
    "ATK": INT,
    "HP": INT,
    "DEF": INT,
    "SPEED": INT
   }
   ```

C.3 Libraries used

Pygame-ce

Pygame-ce or Pygame - Community edition is a distribution of Pygame, which offers more continuous bug fixes, and enhancements. In short, it is a more maintained version of pygame. Pygame is an open source library used to create video games in Python.

PyTMX

PyTMX is a tilemap loader used to load the tilemap from tmx files created on Tiled. It functions by reading every single tile in each layer and returning the position, and image.

Pandas

Pandas is a data analysis toolkit which is used to manipulate, and read data from many data sources. In this project, it is used to read data from csv files, and also manipulate and write on the csv file.

Json

Json is a library used to read from json files, and it is treated similar to a dictionary, but also in a separate file.

Collections

It is a built-in library used to import other data types, in this project it is used to import the defaultdict datatype, which has a function to supply missing values.

Time

It is a built-in library used to return the current time.

Random

It is a built-in library used to return a random value or choice, given a list of items.

Math

It is a built-in library used to calculate some values (particularly vectors) in the project.

C.4 Extensibility of the program

- Meaningful identifier names are used to provide more readability of the code.
- The program structure is divided into files, functioning as modules to make it easier for organizing classes and functions.
- Pseudocodes, diagrams, and data storage schema are provided to help plan the program, and how it works.
- The program is designed in a way where additional contents for the game can be easily implemented.

D. Evaluation

D.1 Does the program work properly?

Yes, the program runs perfectly if all of the libraries are installed properly. Apart from some bugs (player projectile deleting enemy projectile, and attack timing issues), which does not affect the main gameloop much, the game can be played infinitely as the main menu will be brought up everytime you lose or quit the game.

D.2 Future improvements

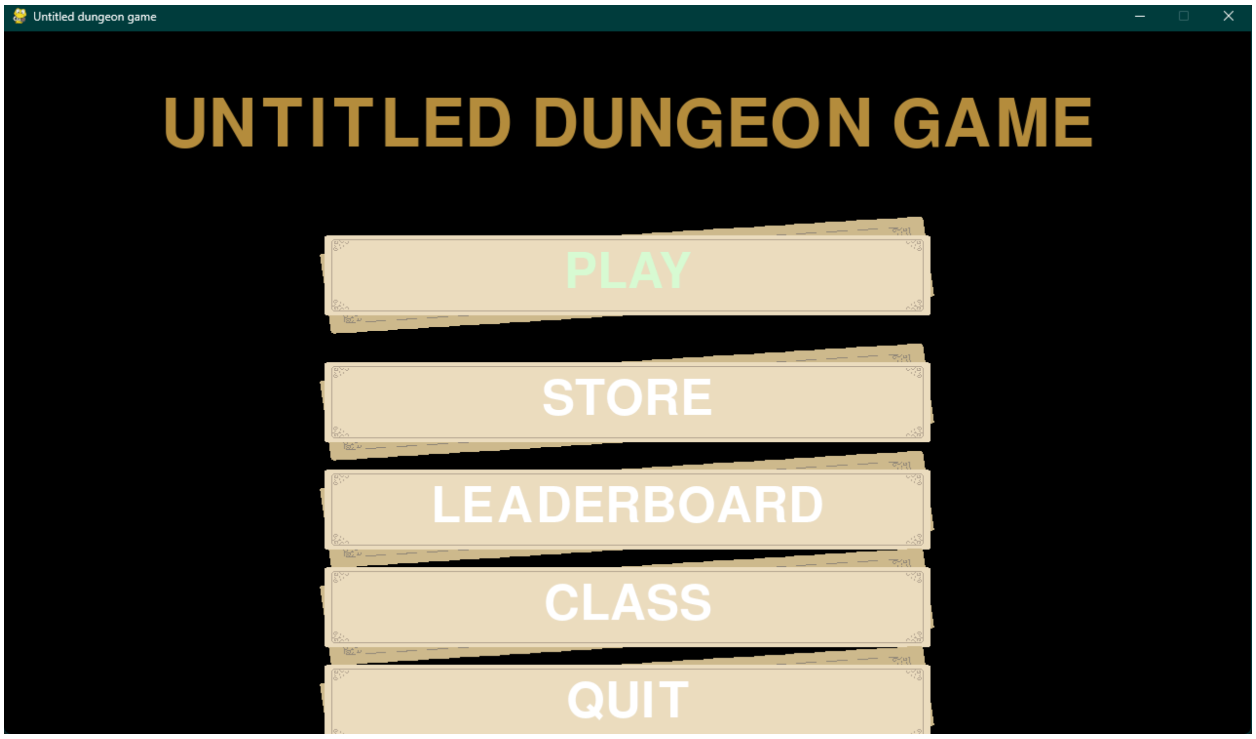
Apart from fixing the bugs, I feel like more visuals can be added, such as animating some of the tiles (the tile with fire), and having a background for the main menu. Moreover, more content can be added such as differing rooms types which will spawn when the player is above a certain threshold, and also bosses. I have planned to implement all of this, but due to the limited amount of time, I couldn’t.

D.3 Lessons learned

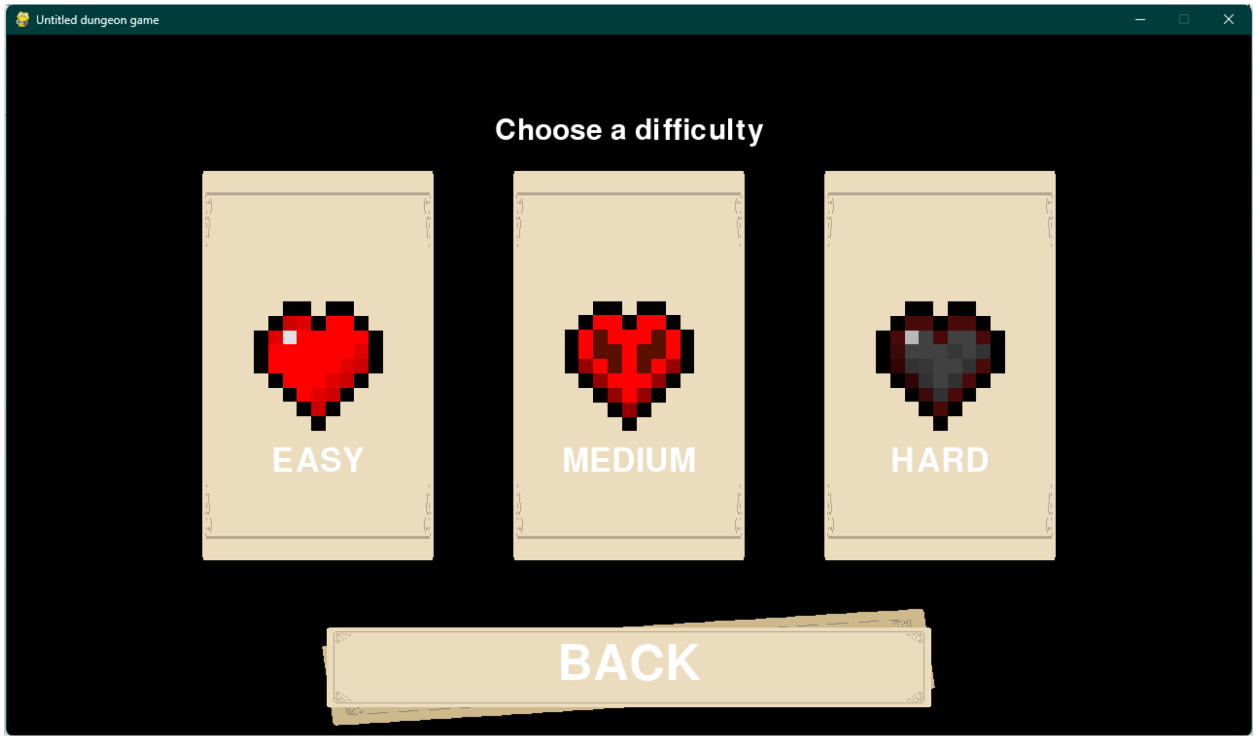
During the final project, I have learned a lot, especially since this is my first time creating a game. In addition to the lessons learned in class, I also managed to implement other programing techniques, or data structures that are not taught in class, but are valuable. For example, for the buttons I implemented a linked list structure. In addition, since pygame is more of a graphics library, other than the often syntax error that I would get, it also forces me to create my own way of debugging such as drawing the bounding boxes, printing some variables/states, etc.

E. Evidence of the working program

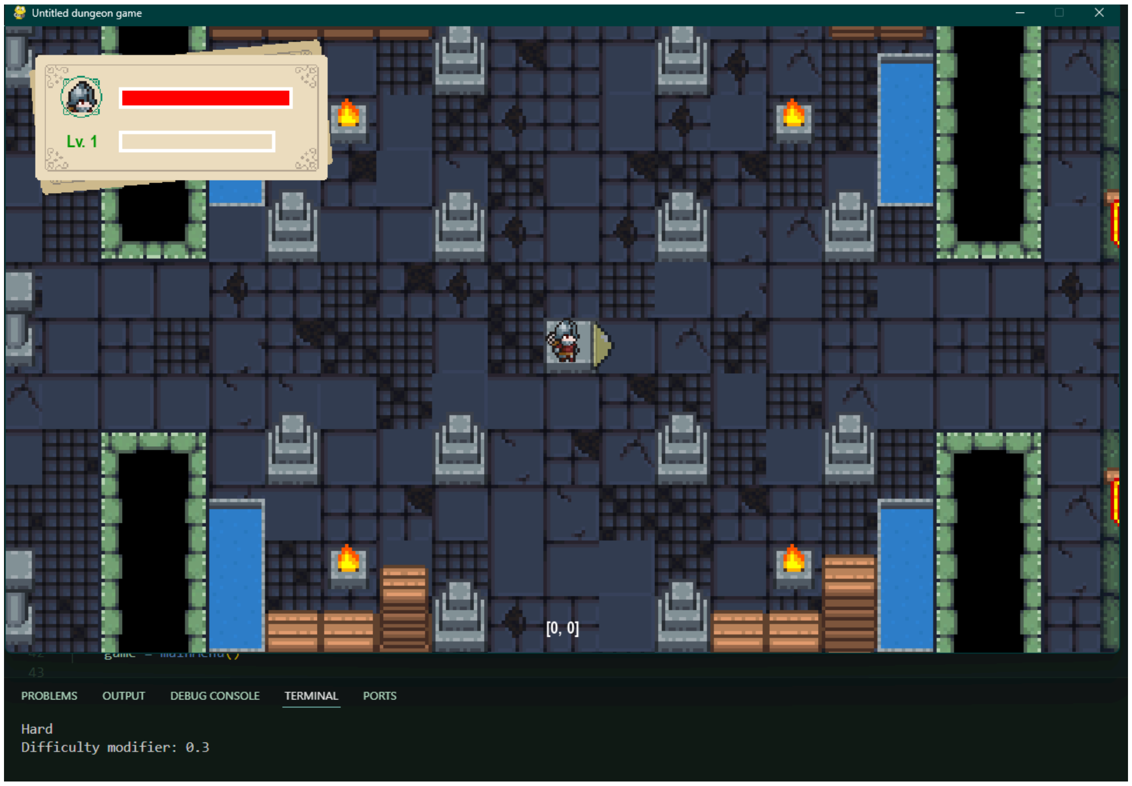
E.1 Main menu



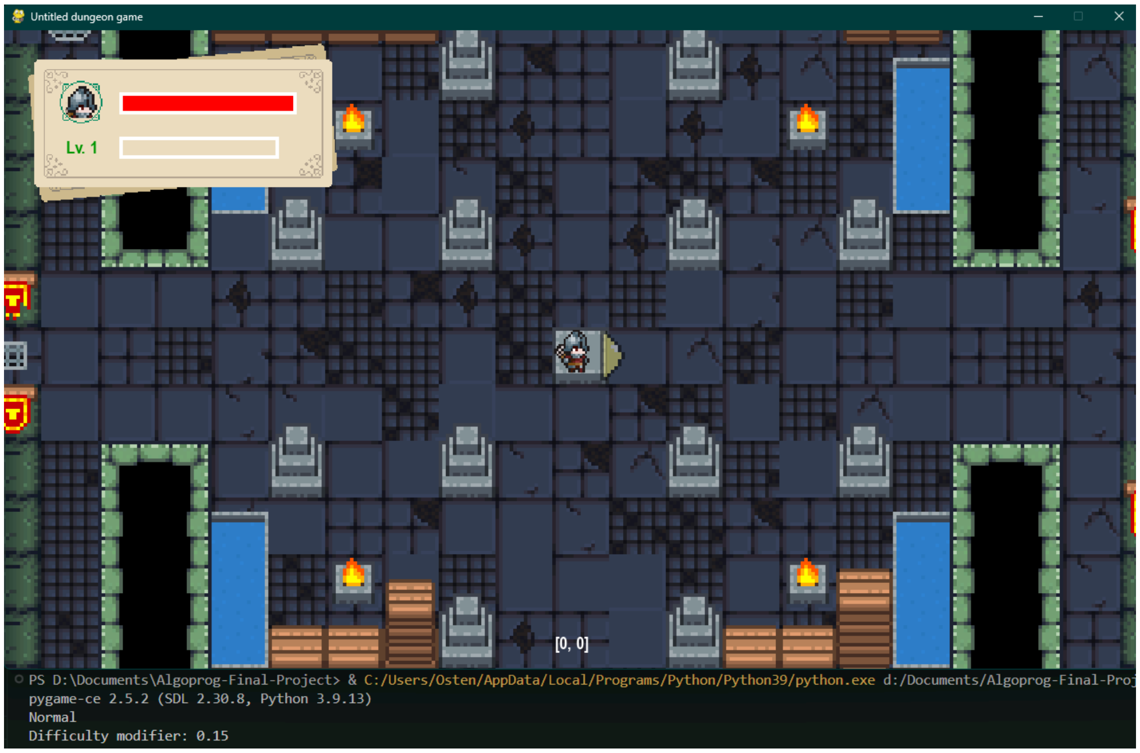
E.2 Select difficulty



E.2.3 Selected difficulty

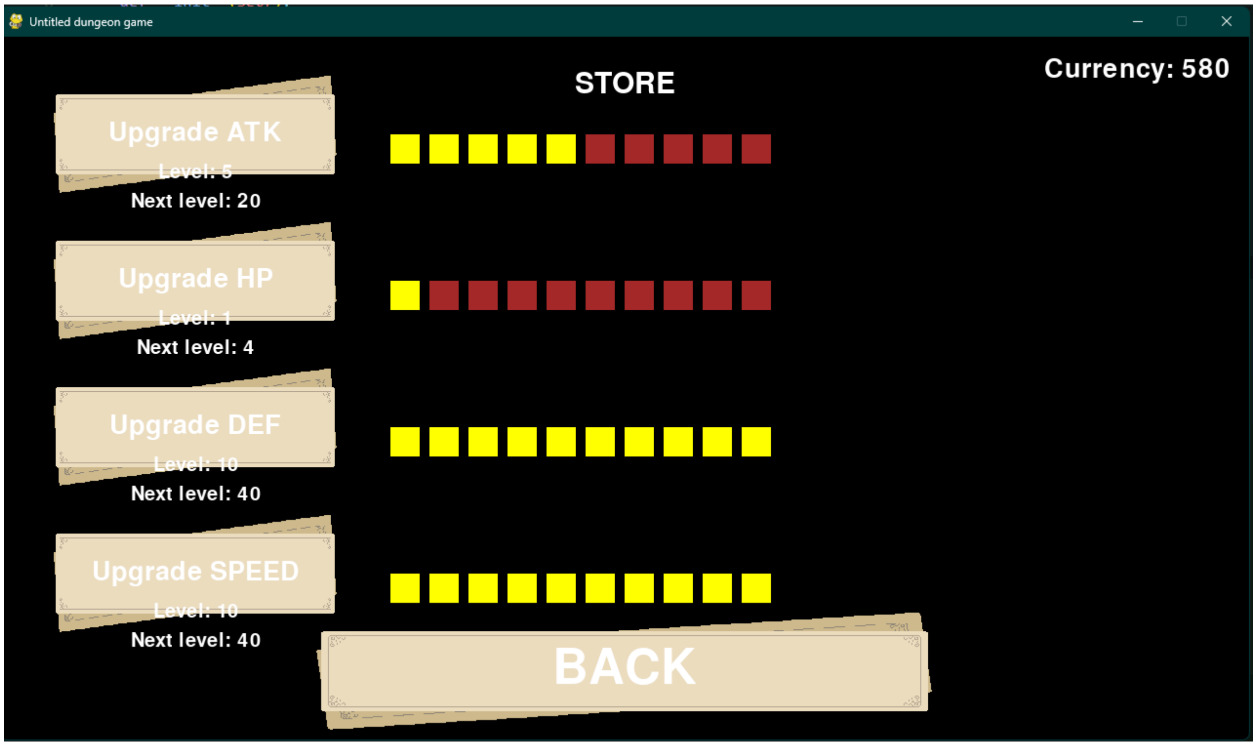


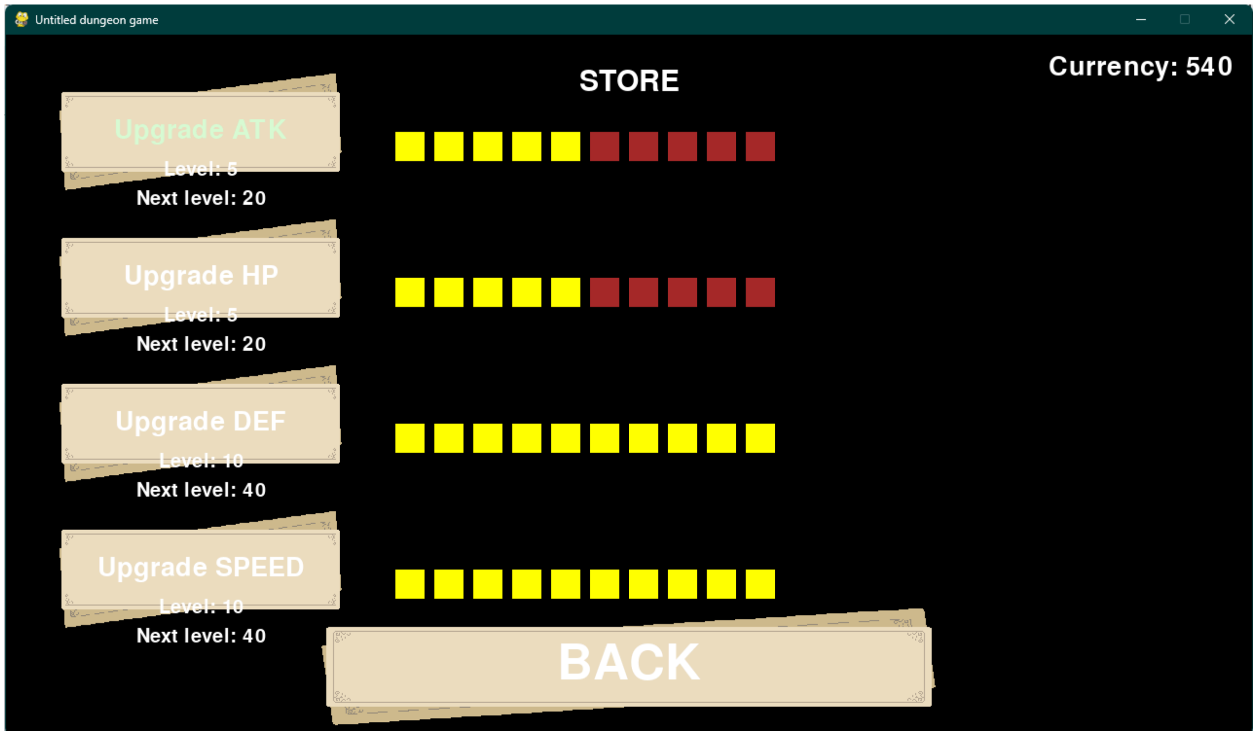
```
Hard
Difficulty modifier: 0.3
```



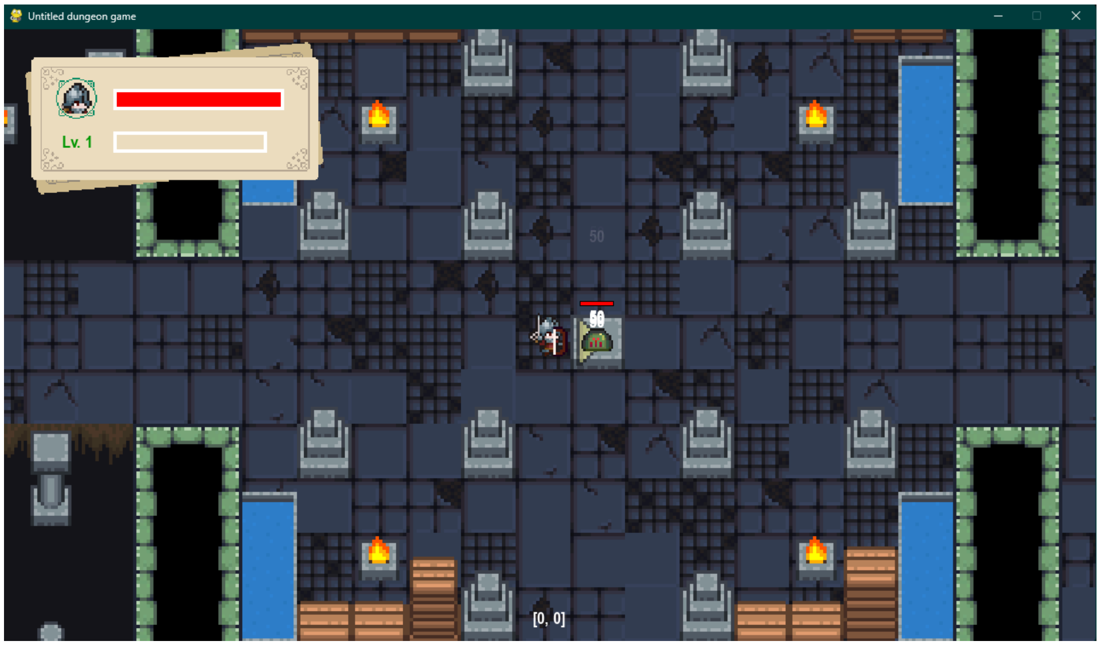
```
PS D:\Documents\Algoprog-Final-Project> & C:/Users/Osten/AppData/Local/Programs/Python/Python39/python.exe d:/Documents/Algoprog-Final-Proj
pygame-ce 2.5.2 (SDL 2.30.8, Python 3.9.13)
Normal
Difficulty modifier: 0.15
```

E.3 Store

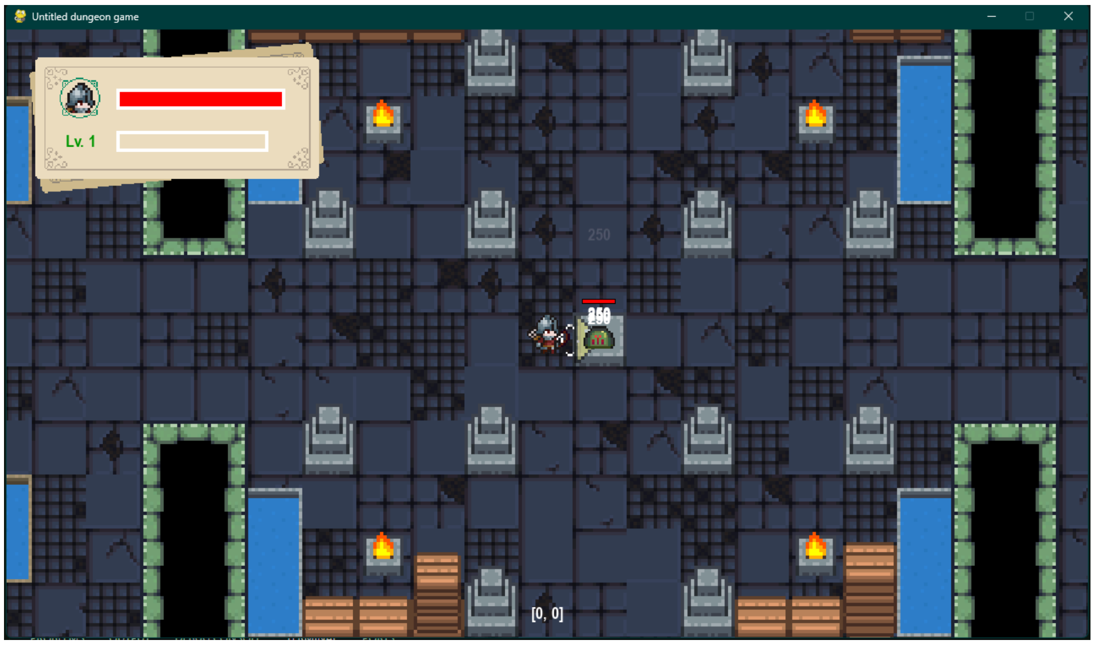

Untitled dungeon game
STORE
Currency: 580
Upgrade ATK
Level: 5
Next level: 20
Upgrade HP
Level: 1
Next level: 4
Upgrade DEF
Level: 10
Next level: 40
Upgrade SPEED
Level: 10
Next level: 40
BACK


Untitled dungeon game
STORE
Currency: 540
Upgrade ATK
Level: 5
Next level: 20
Upgrade HP
Level: 5
Next level: 20
Upgrade DEF
Level: 10
Next level: 40
Upgrade SPEED
Level: 10
Next level: 40
BACK

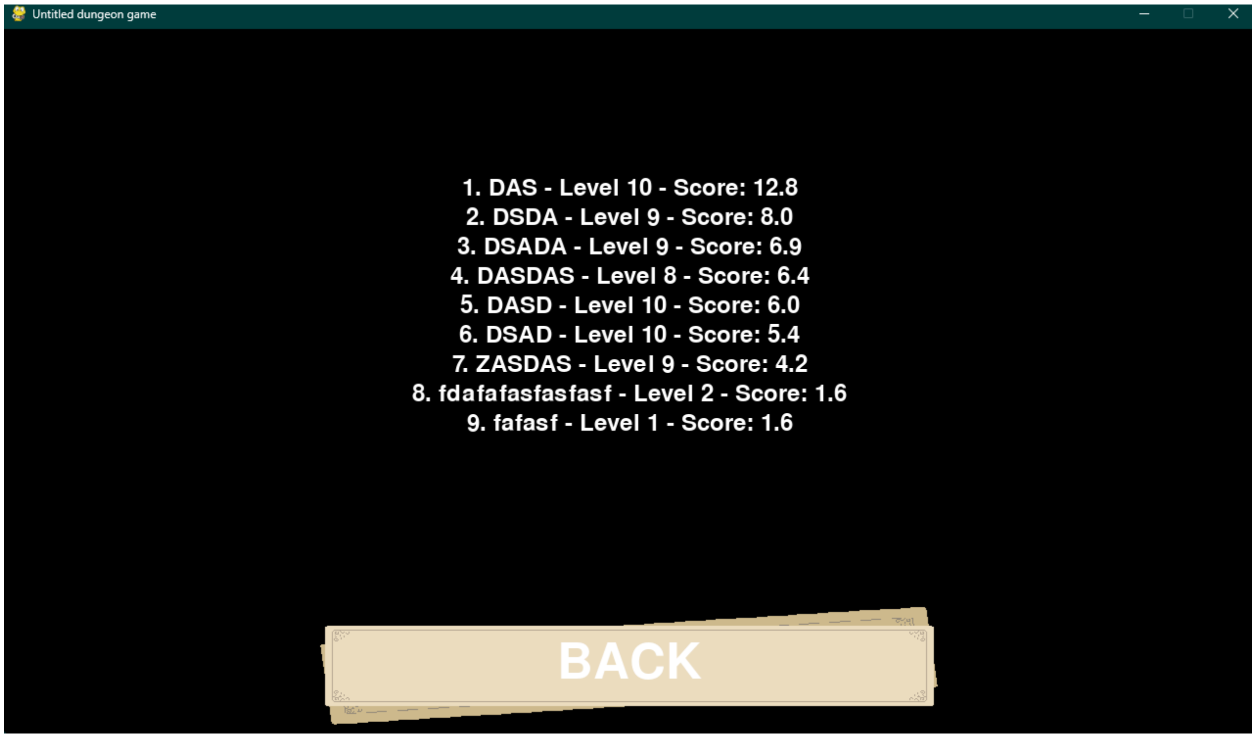
Before upgrade (ATK):



After upgrade (ATK):



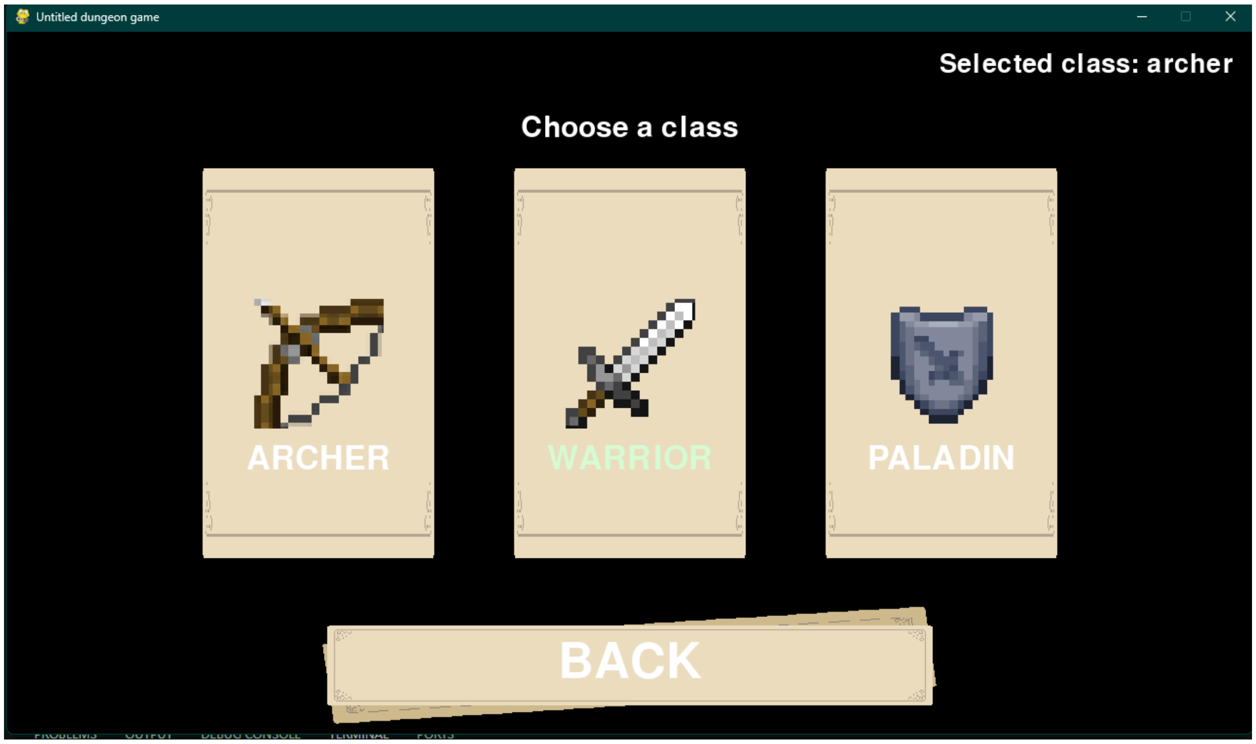
E.4 Leaderboard



Untitled dungeon game

1. DAS - Level 10 - Score: 12.8
2. DSDA - Level 9 - Score: 8.0
3. DSADA - Level 9 - Score: 6.9
4. DASDAS - Level 8 - Score: 6.4
5. DASD - Level 10 - Score: 6.0
6. DSAD - Level 10 - Score: 5.4
7. ZASDAS - Level 9 - Score: 4.2
8. fdafafasfasfasf - Level 2 - Score: 1.6
9. fafasf - Level 1 - Score: 1.6

E.5 Class

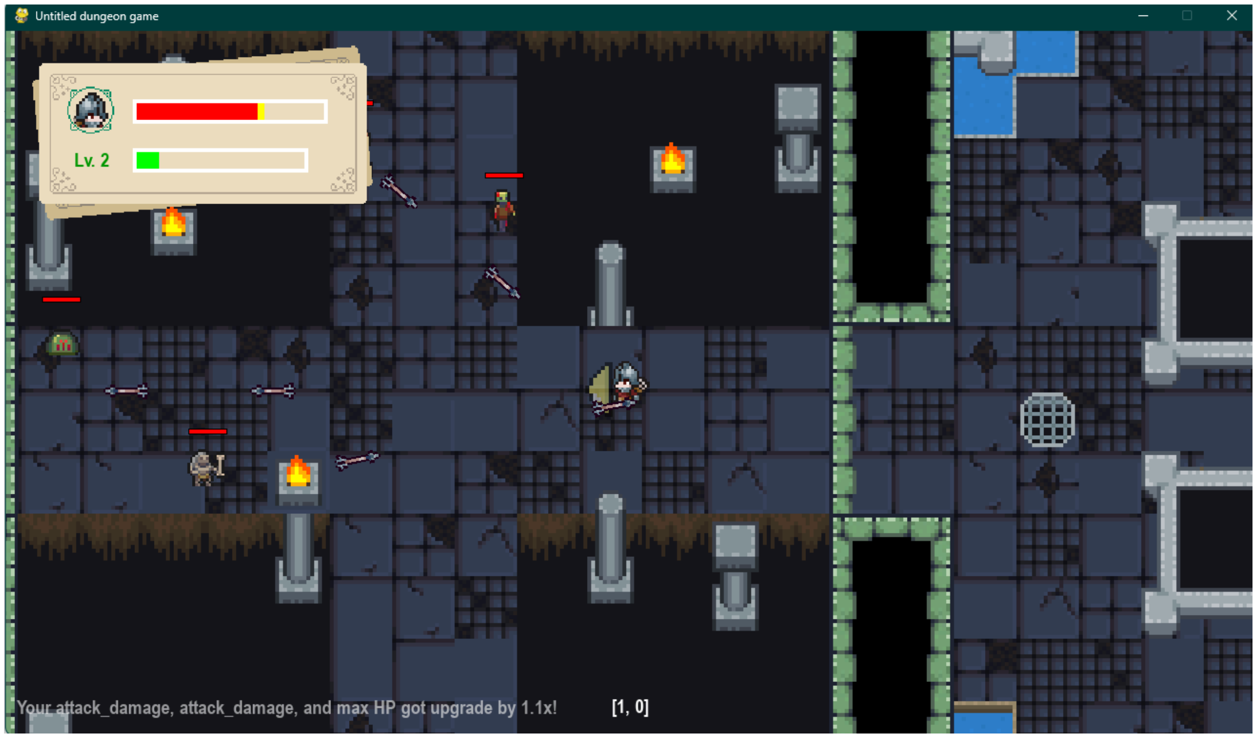


Untitled dungeon game

Selected class: archer

Choose a class

E.6 Game

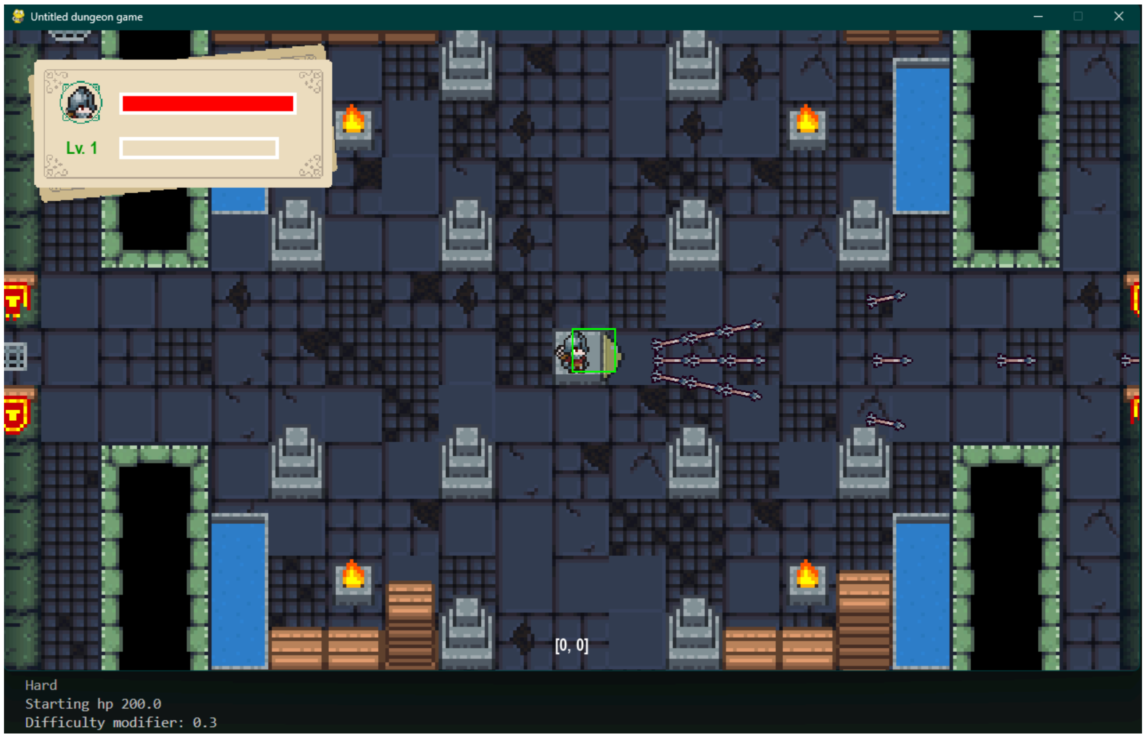


Your attack_damage, attack_damage, and max HP got upgrade by 1.1x!

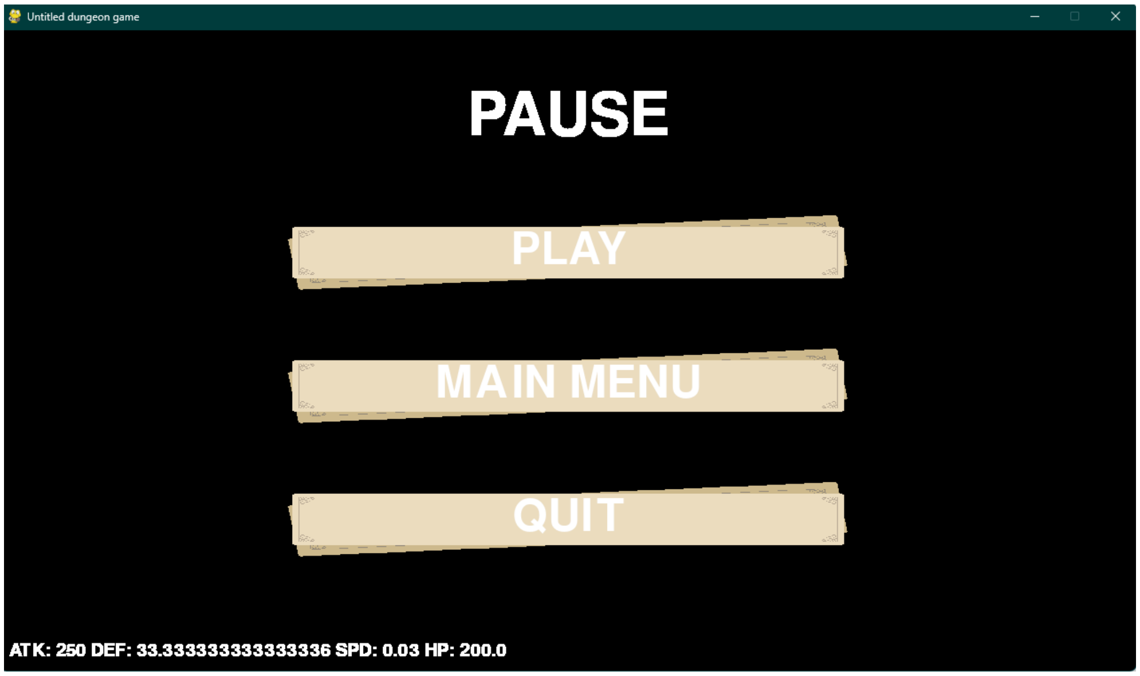
[1, 0]

E.7 Classes system (Green is the swing range, more can be seen in the demo video)

E.7.1 Archer

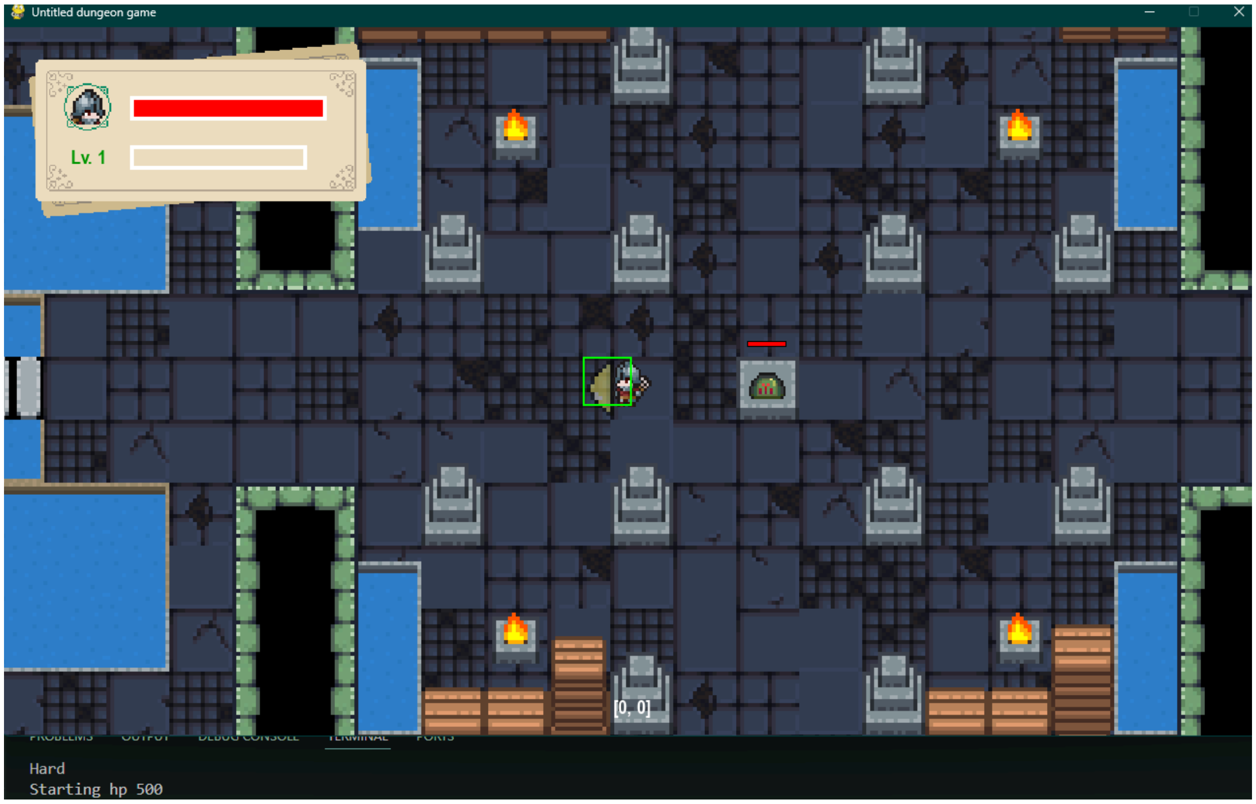


```
Hard
Starting hp 200.0
Difficulty modifier: 0.3
```

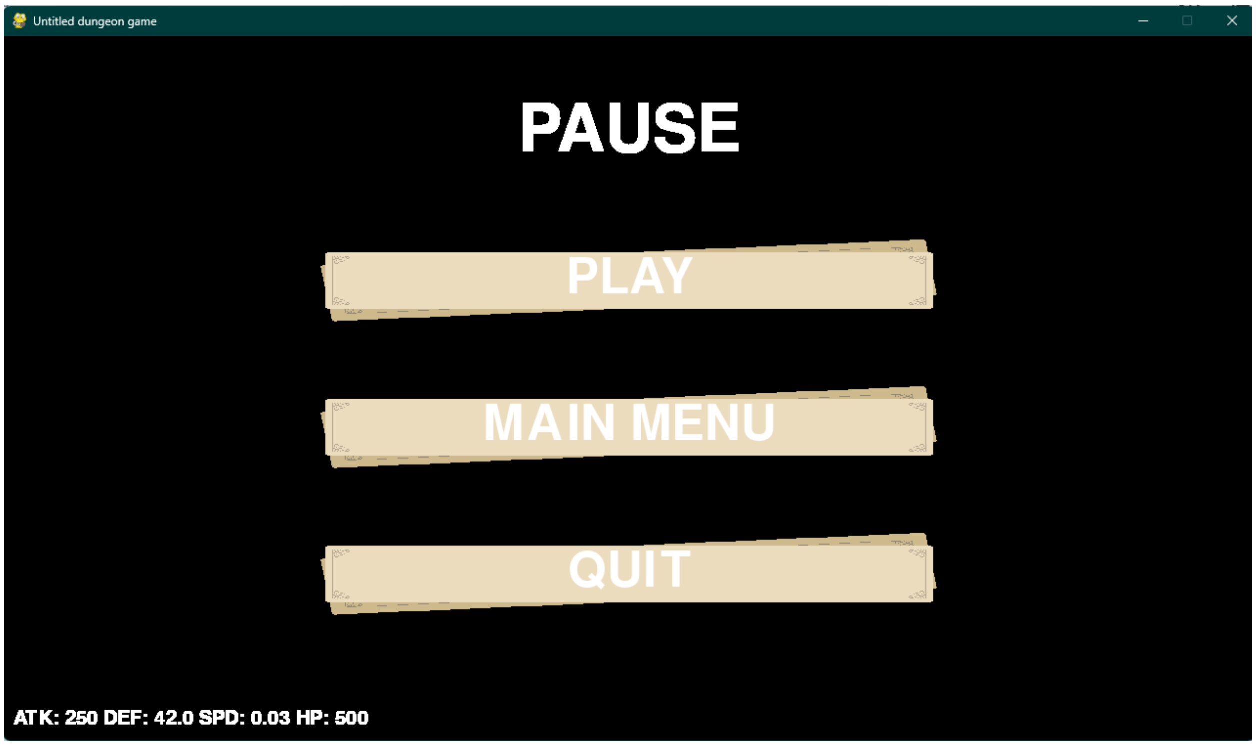


```
ATK: 250 DEF: 33.333333333333336 SPD: 0.03 HP: 200.0
```

E.7.2 Paladin

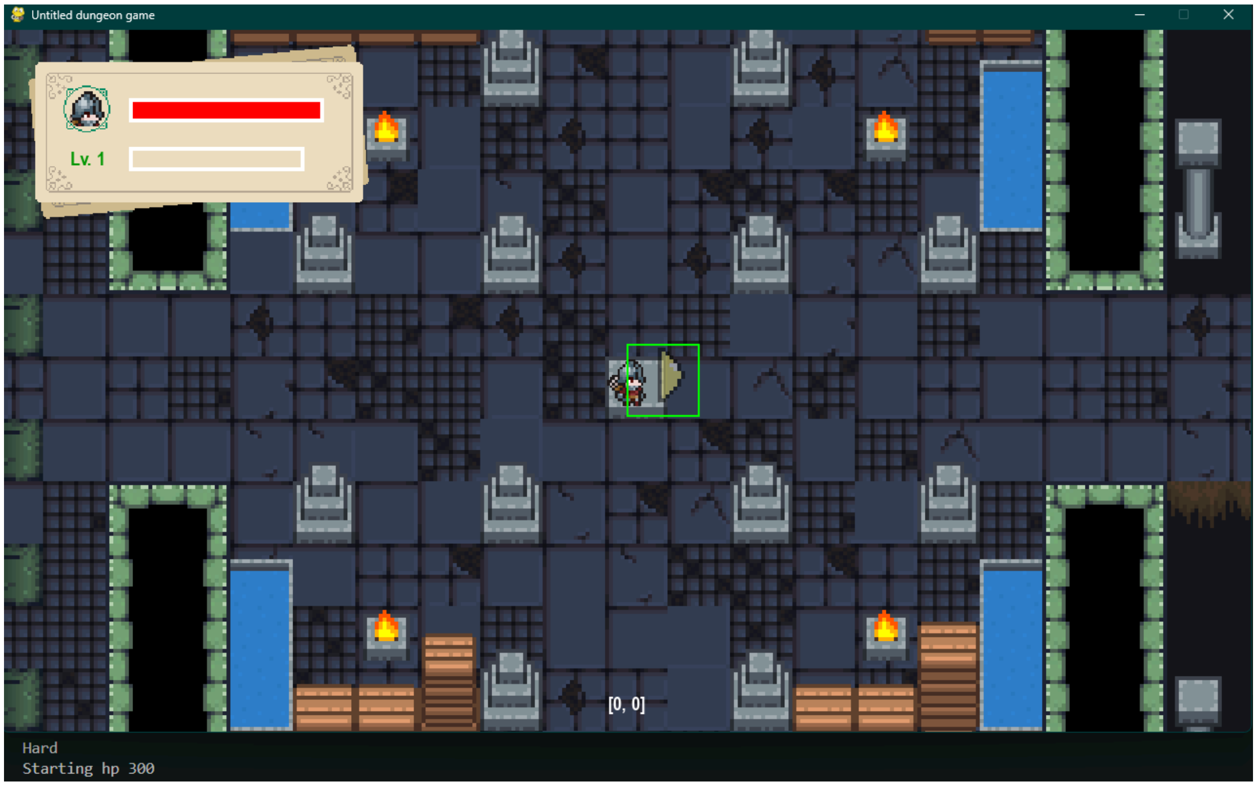


```
Hard
Starting hp 500
```

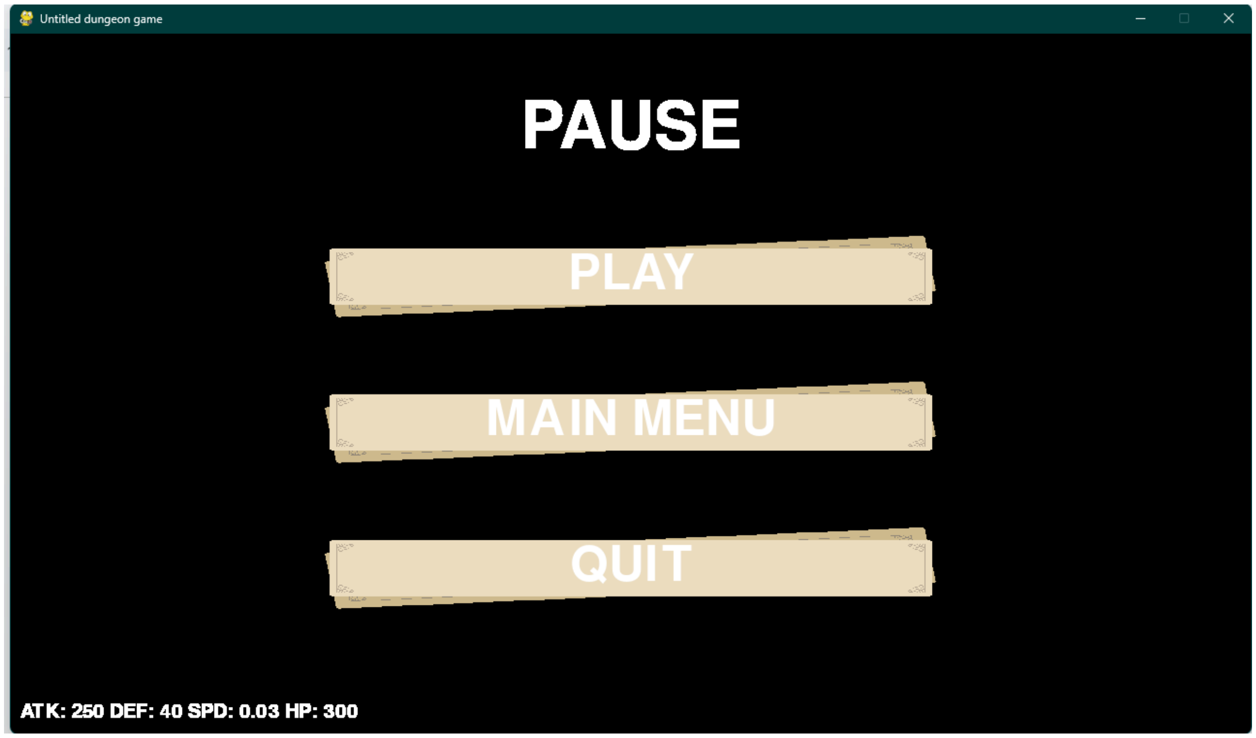


ATK: 250 DEF: 42.0 SPD: 0.03 HP: 500

E.7.3 Warrior



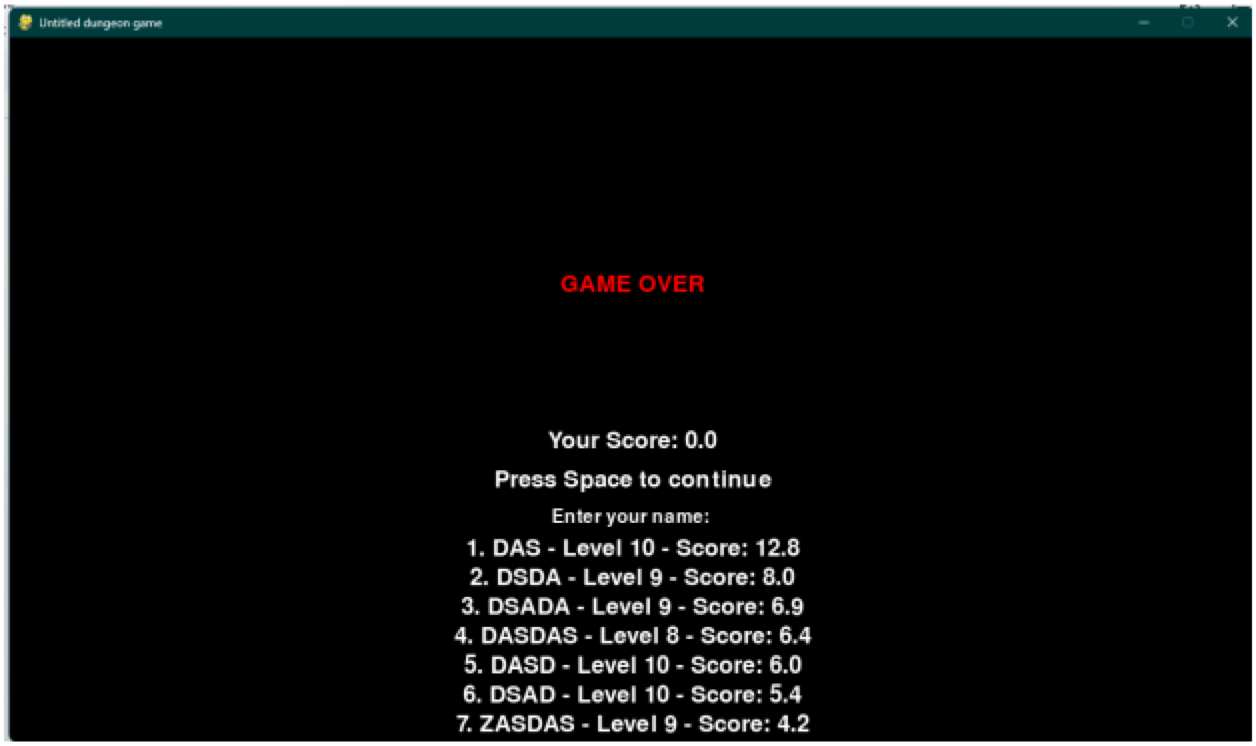
Hard
Starting hp 300



ATK: 250 DEF: 40 SPD: 0.03 HP: 300

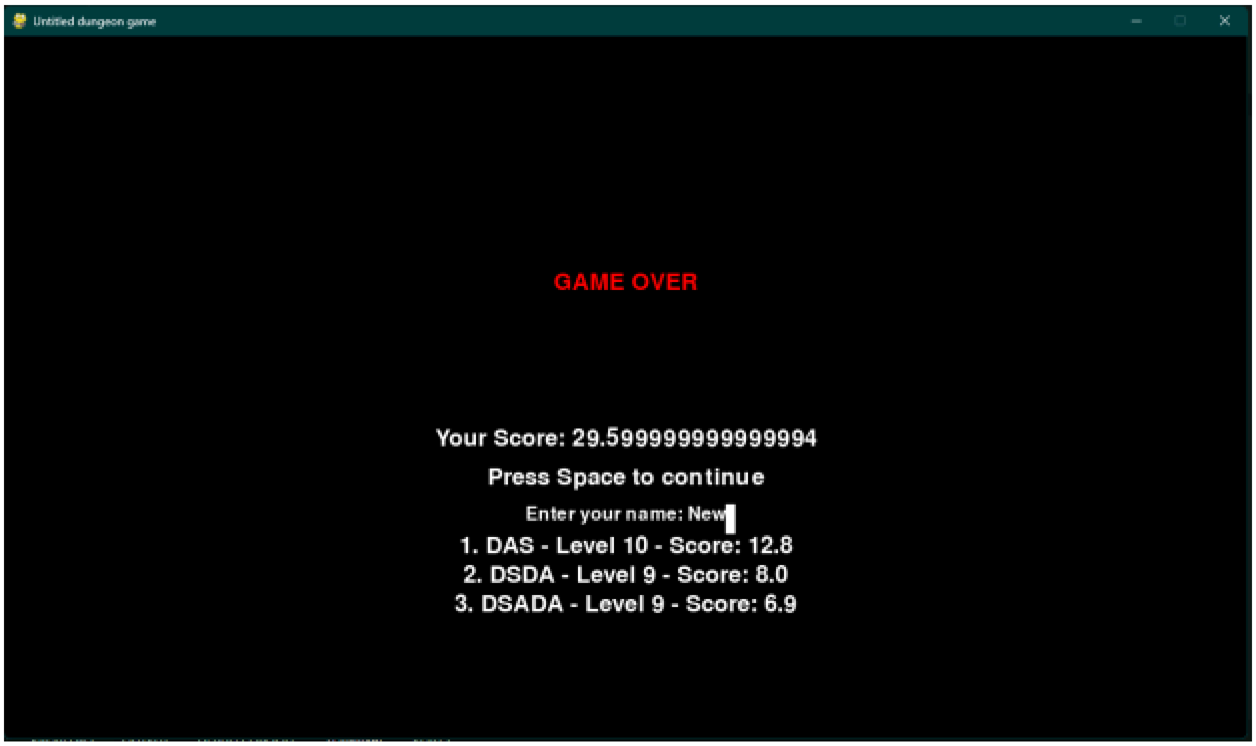
E.8 Pause

E.9 Gameover screen

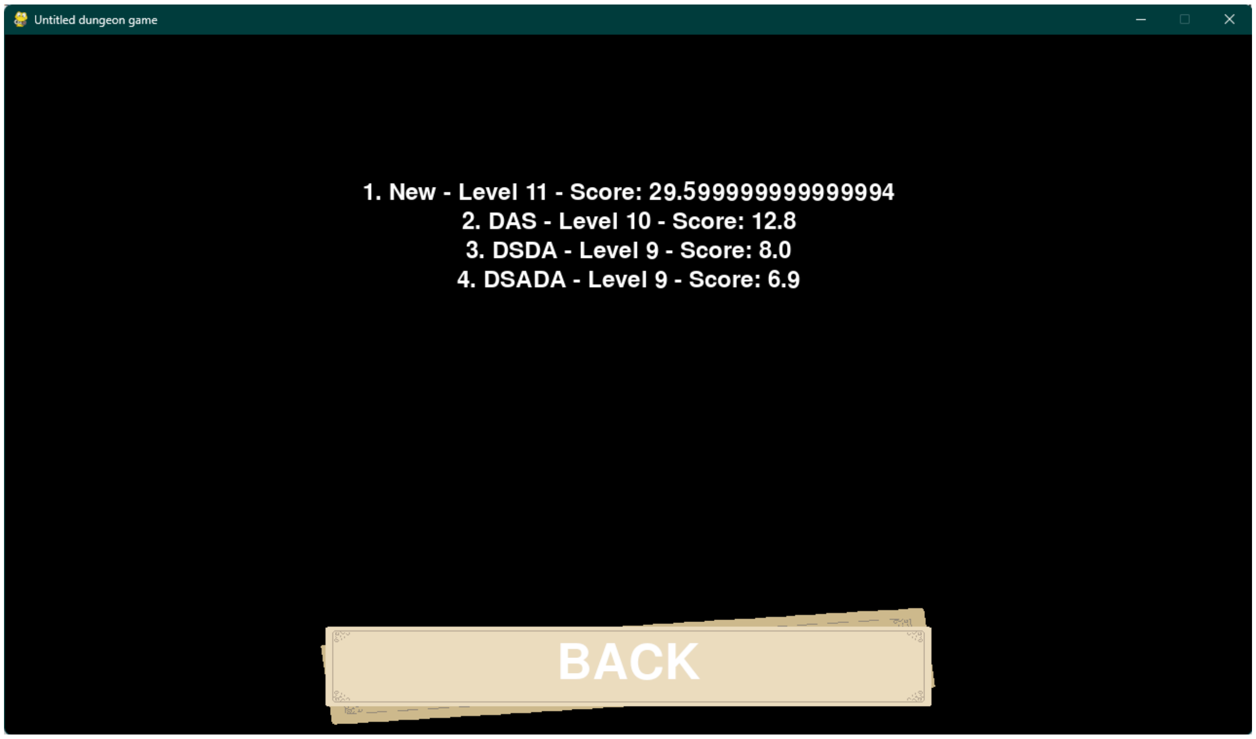


```
GAME OVER

Your Score: 0.0
Press Space to continue
Enter your name:
1. DAS - Level 10 - Score: 12.8
2. DSDA - Level 9 - Score: 8.0
3. DSADA - Level 9 - Score: 6.9
4. DASDAS - Level 8 - Score: 6.4
5. DASD - Level 10 - Score: 6.0
6. DSAD - Level 10 - Score: 5.4
7. ZASDAS - Level 9 - Score: 4.2
```

E.10 Gameover update leaderboard



```
GAME OVER

Your Score: 29.599999999999994
Press Space to continue
Enter your name: New
1. DAS - Level 10 - Score: 12.8
2. DSDA - Level 9 - Score: 8.0
3. DSADA - Level 9 - Score: 6.9
```



1. New - Level 11 - Score: 29.599999999999994
2. DAS - Level 10 - Score: 12.8
3. DSDA - Level 9 - Score: 8.0
4. DSADA - Level 9 - Score: 6.9

F. References

Clear Code. (2024, February 11). *Creating an amazing 2D platformer in Python [SNES inspired]*. YouTube. https://www.youtube.com/watch?v=WViyCAa6yLI

Coding With Russ. (2023, April 23). *Pygame Sprites And Groups Explained!* Www.youtube.com. https://www.youtube.com/watch?v=4TfZjhw0J-8

F.1 Resources used

DeepDiveGameStudio. (2023). *Undead Asset Pack [16x16]*. Itch.io. https://deepdivegamestudio.itch.io/undead-asset-pack

DeepDiveGameStudio. (2024). *Monster Asset Pack [16x16]*. Itch.io. https://deepdivegamestudio.itch.io/monsterassetpack

Humble Pixel. (2024). *Pocket Inventory Series #5 : Player Status*. Itch.io. https://humblepixel.itch.io/pocket-inventory-series-5-player-status

Pine Druid. (2025). *Damp Dungeon - Tileset and Sprites*. Itch.io. https://pine-druid.itch.io/damp-dungeon-tileset-and-sprites

Zerie. (2024). *Tiny RPG Character Asset Pack v1.03*. Itch.io. https://zerie.itch.io/tiny-rpg-character-asset-pack