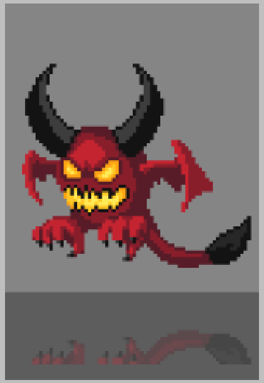
Dungeon Destroyer

Husain Patanwala (CS210)

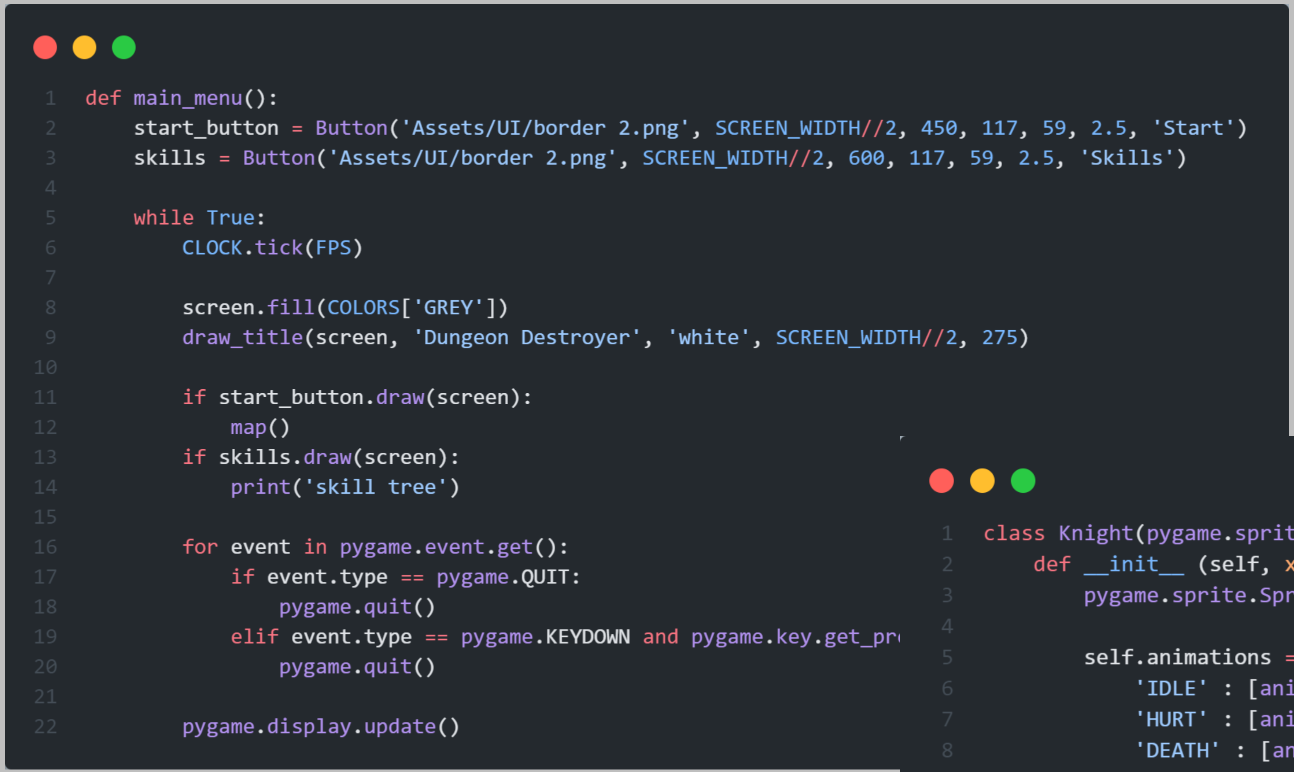
General Overview

I am considering making a turn-based RPG style game called “Dungeon Destroyer”. The main premise of the game is to traverse through the map which will be implemented using a graph, and the player has to battle monsters in turn-base combat. To learn new skills, players must gain experience and work their way through the map. Certain skills will be required to help beat certain monsters, and thus, players will have to focus on unlocking different parts of the skill tree to ensure they can pass levels. Also, user related info will be stored using hashmaps and lists, and the turn based system between player and ai enemies will be implemented using a queue. The end goal of the game is to reach the end of the map and defeat the boss. The story is still somewhat incomplete, but these are the general mechanics I would like to implement.

Data Structures:



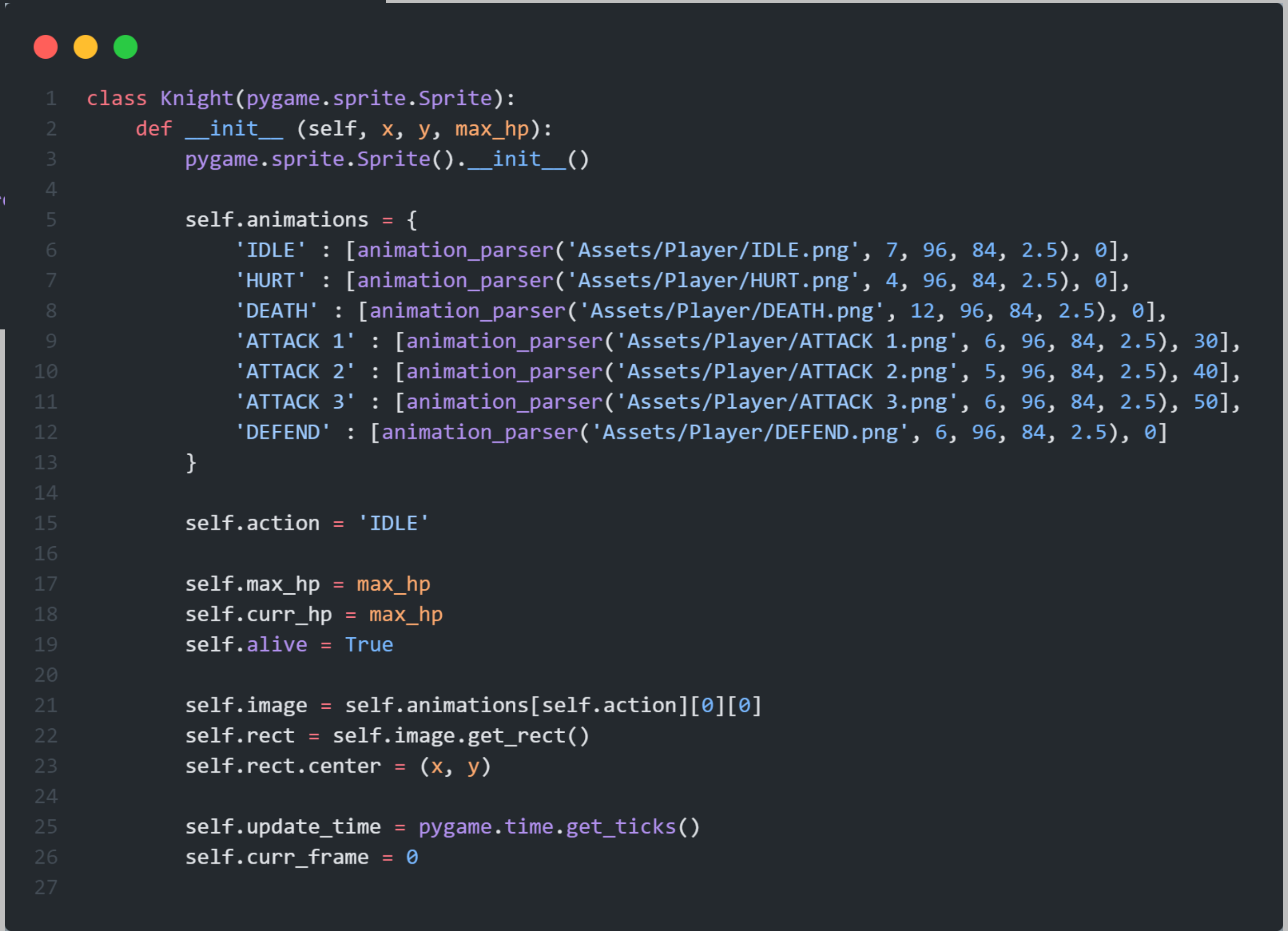
- Hashmap
  - Entity animations
  - Game colors
- Queue
  - Each level has a queue to simulate turn based combat
- List
  - Store frames for each animation sequence
- Tree
  - A skill tree used to show player progression; new skill will unlock based on level completion
- Graph
  - Used to represent the map structure



```python
def main_menu():
    start_button = Button('Assets/UI/border 2.png', SCREEN_WIDTH//2, 450, 117, 59, 2.5, 'Start')
    skills = Button('Assets/UI/border 2.png', SCREEN_WIDTH//2, 600, 117, 59, 2.5, 'Skills')

    while True:
        CLOCK.tick(FPS)

        screen.fill(COLORS['GREY'])
        draw_title(screen, 'Dungeon Destroyer', 'white', SCREEN_WIDTH//2, 275)

        if start_button.draw(screen):
            map()
        if skills.draw(screen):
            print('skill tree')

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
            elif event.type == pygame.KEYDOWN and pygame.key.get_pr
                pygame.quit()

        pygame.display.update()
```

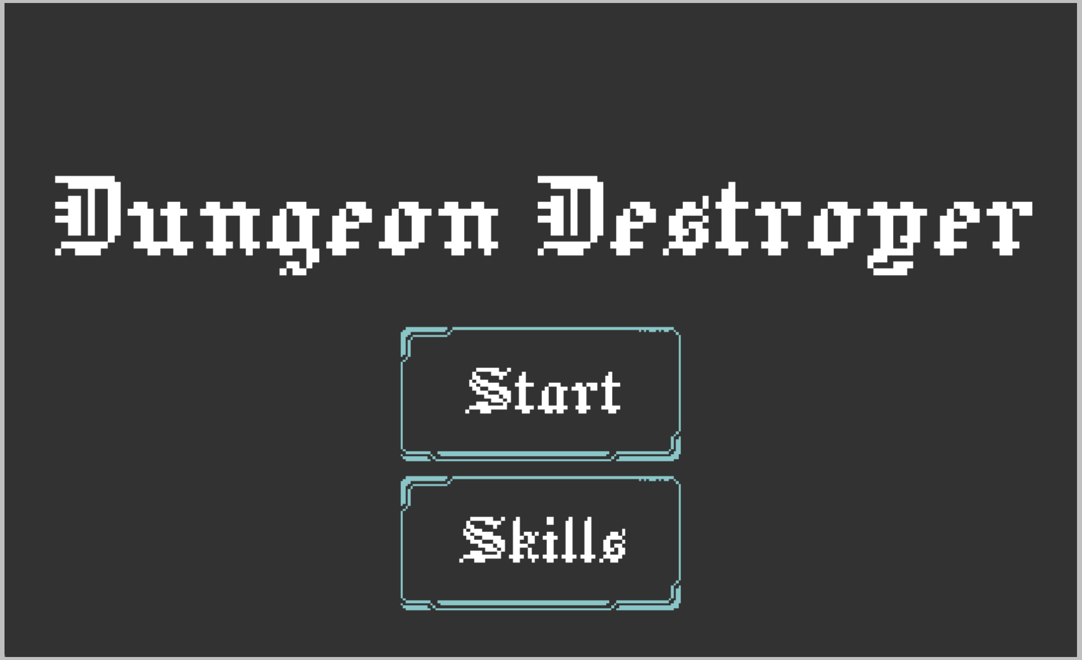
← Main menu function

Knight constructor →

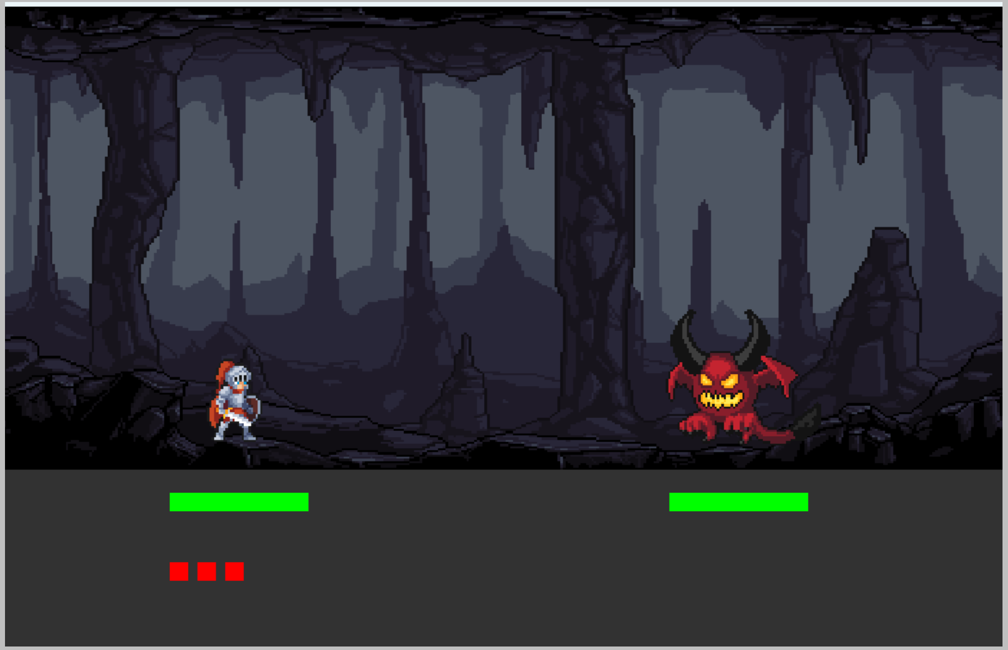


```python
class Knight(pygame.sprite.Sprite):
    def __init__ (self, x, y, max_hp):
        pygame.sprite.Sprite().__init__()

        self.animations = {
            'IDLE' : [animation_parser('Assets/Player/IDLE.png', 7, 96, 84, 2.5), 0],
            'HURT' : [animation_parser('Assets/Player/HURT.png', 4, 96, 84, 2.5), 0],
            'DEATH' : [animation_parser('Assets/Player/DEATH.png', 12, 96, 84, 2.5), 0],
            'ATTACK 1' : [animation_parser('Assets/Player/ATTACK 1.png', 6, 96, 84, 2.5), 30],
            'ATTACK 2' : [animation_parser('Assets/Player/ATTACK 2.png', 5, 96, 84, 2.5), 40],
            'ATTACK 3' : [animation_parser('Assets/Player/ATTACK 3.png', 6, 96, 84, 2.5), 50],
            'DEFEND' : [animation_parser('Assets/Player/DEFEND.png', 6, 96, 84, 2.5), 0]
        }

        self.action = 'IDLE'

        self.max_hp = max_hp
        self.curr_hp = max_hp
        self.alive = True

        self.image = self.animations[self.action][0][0]
        self.rect = self.image.get_rect()
        self.rect.center = (x, y)

        self.update_time = pygame.time.get_ticks()
        self.curr_frame = 0
```

GUI Mockups + Progress so Far

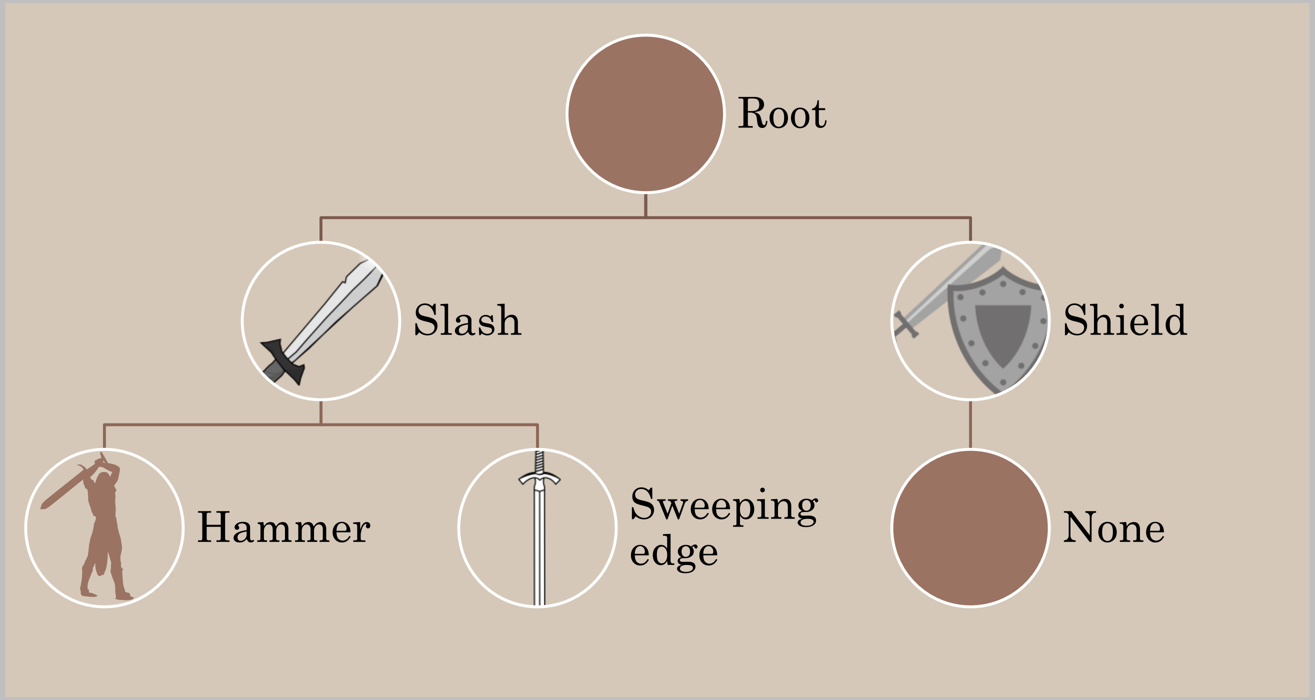


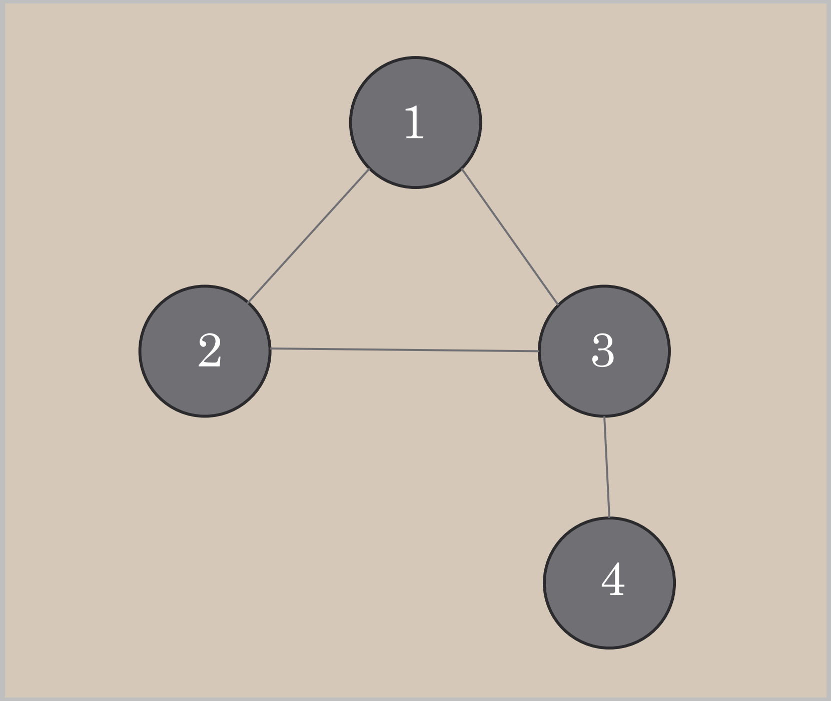
This is the main menu/start screen of the game. From here you can either view the game map (in progress) or the skill tree (tbd).



This is an example of a possible level. The health bars are show in green, and adapt to damage changes. The red buttons are placeholders for attack input (they work btw).

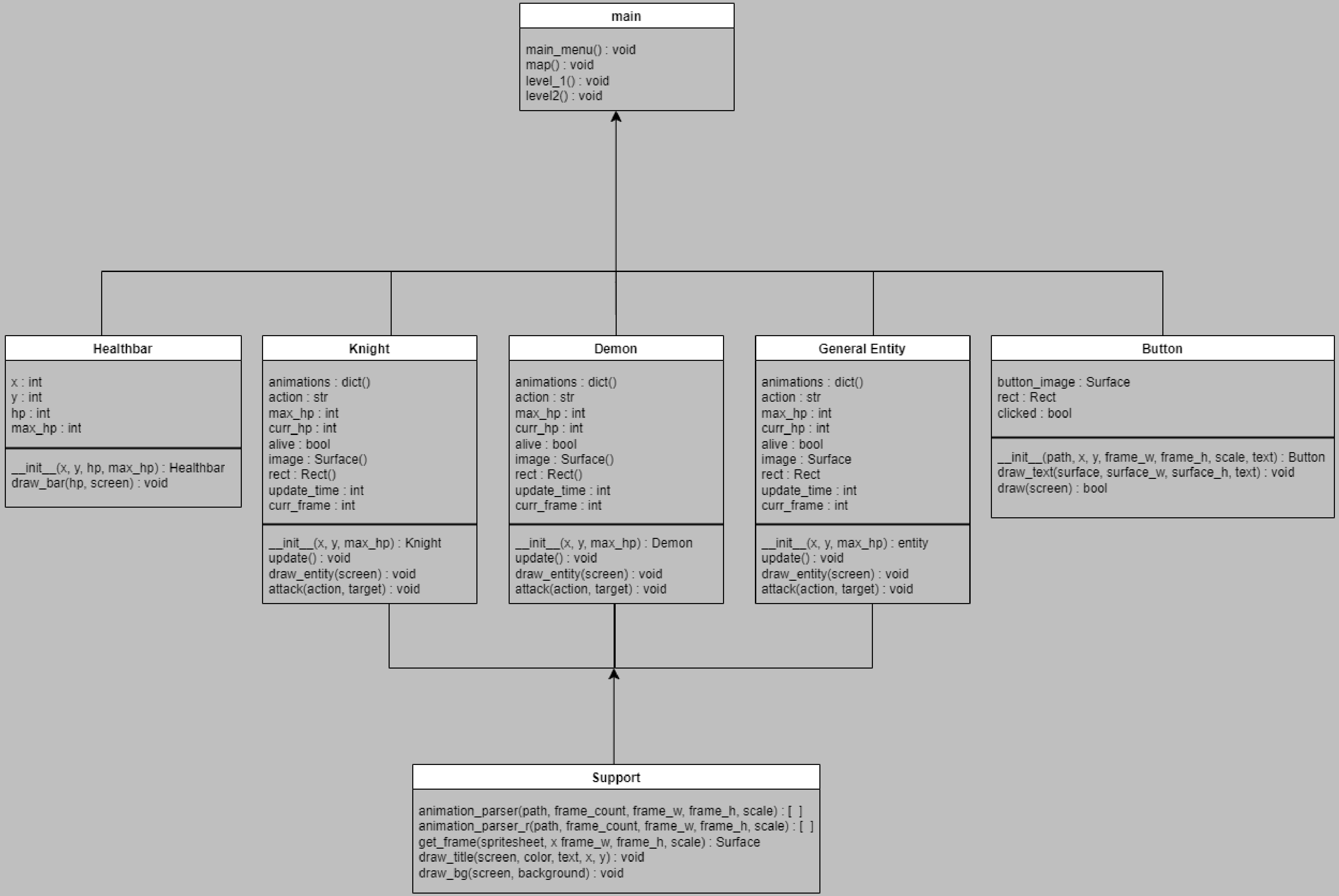
GUI Mockups + Progress so Far cont.



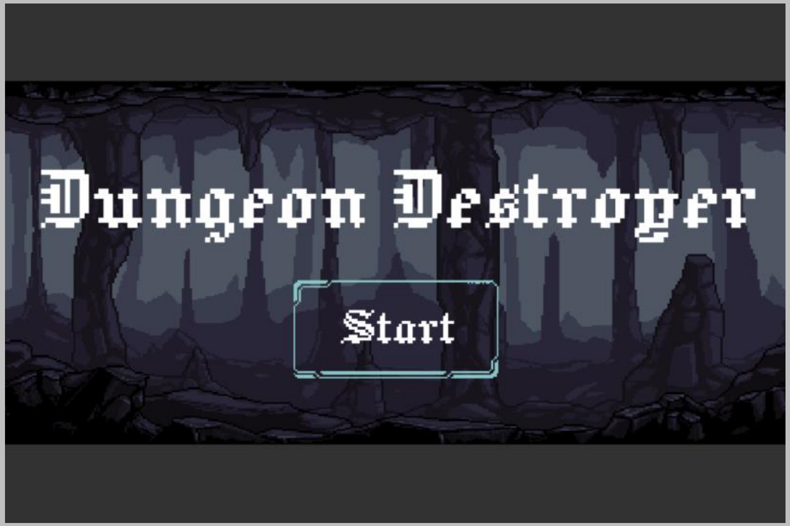


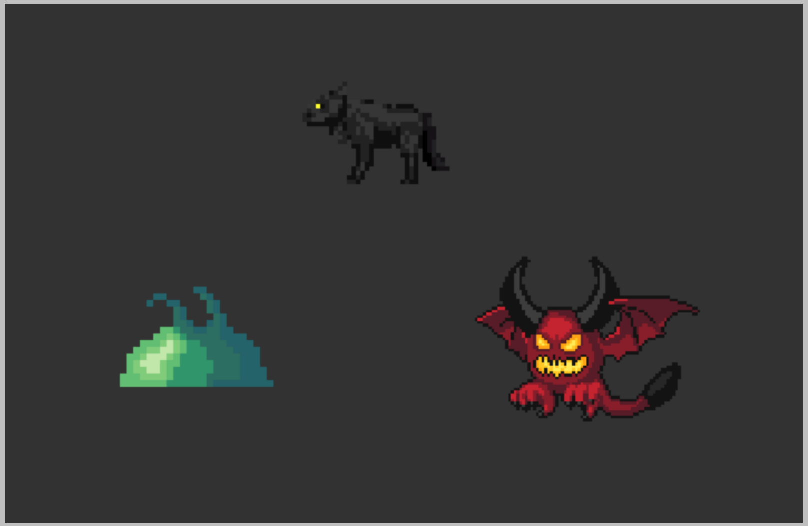
This is a mockup of the skill tree. What I'm thinking is that for each level you can possibly unlock a new level on the skill tree…

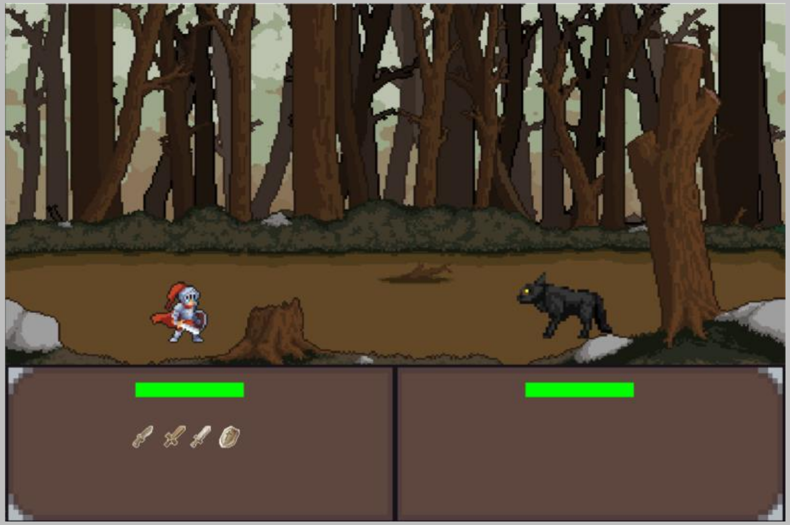
This is a simple graph, and this is how I want to lay it out. Although, I have to figure out how I'm going to implement it.

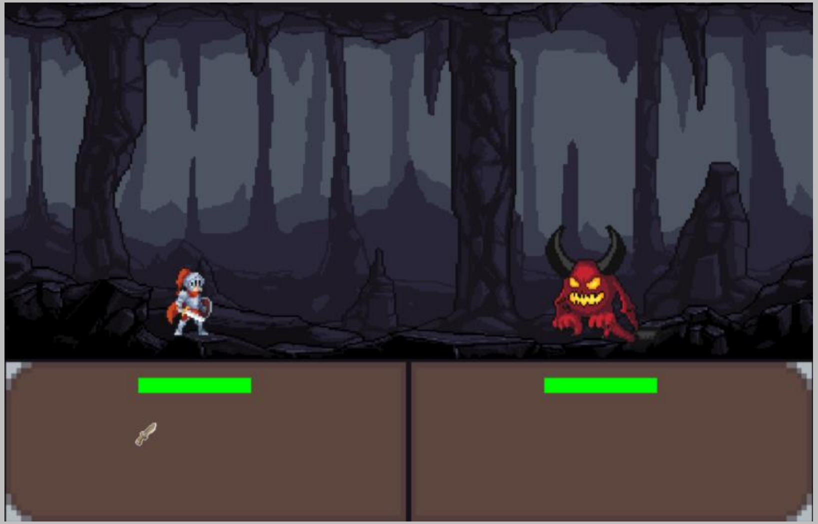

main
main_menu() : void
map() : void
level_1() : void
level2() : void
Healthbar
x : int
y : int
hp : int
max_hp : int
__init__(x, y, hp, max_hp) : Healthbar
draw_bar(hp, screen) : void
Knight
animations : dict()
action : str
max_hp : int
curr_hp : int
alive : bool
image : Surface()
rect : Rect()
update_time : int
curr_frame : int
__init__(x, y, max_hp) : Knight
update() : void
draw_entity(screen) : void
attack(action, target) : void
Demon
animations : dict()
action : str
max_hp : int
curr_hp : int
alive : bool
image : Surface()
rect : Rect()
update_time : int
curr_frame : int
__init__(x, y, max_hp) : Demon
update() : void
draw_entity(screen) : void
attack(action, target) : void
General Entity
animations : dict()
action : str
max_hp : int
curr_hp : int
alive : bool
image : Surface
rect : Rect
update_time : int
curr_frame : int
__init__(x, y, max_hp) : entity
update() : void
draw_entity(screen) : void
attack(action, target) : void
Button
button_image : Surface
rect : Rect
clicked : bool
__init__(path, x, y, frame_w, frame_h, scale, text) : Button
draw_text(surface, surface_w, surface_h, text) : void
draw(screen) : bool
Support
animation_parser(path, frame_count, frame_w, frame_h, scale) : [ ]
animation_parser_r(path, frame_count, frame_w, frame_h, scale) : [ ]
get_frame(spritesheet, x frame_w, frame_h, scale) : Surface
draw_title(screen, color, text, x, y) : void
draw_bg(screen, background) : void

GUI Implementation

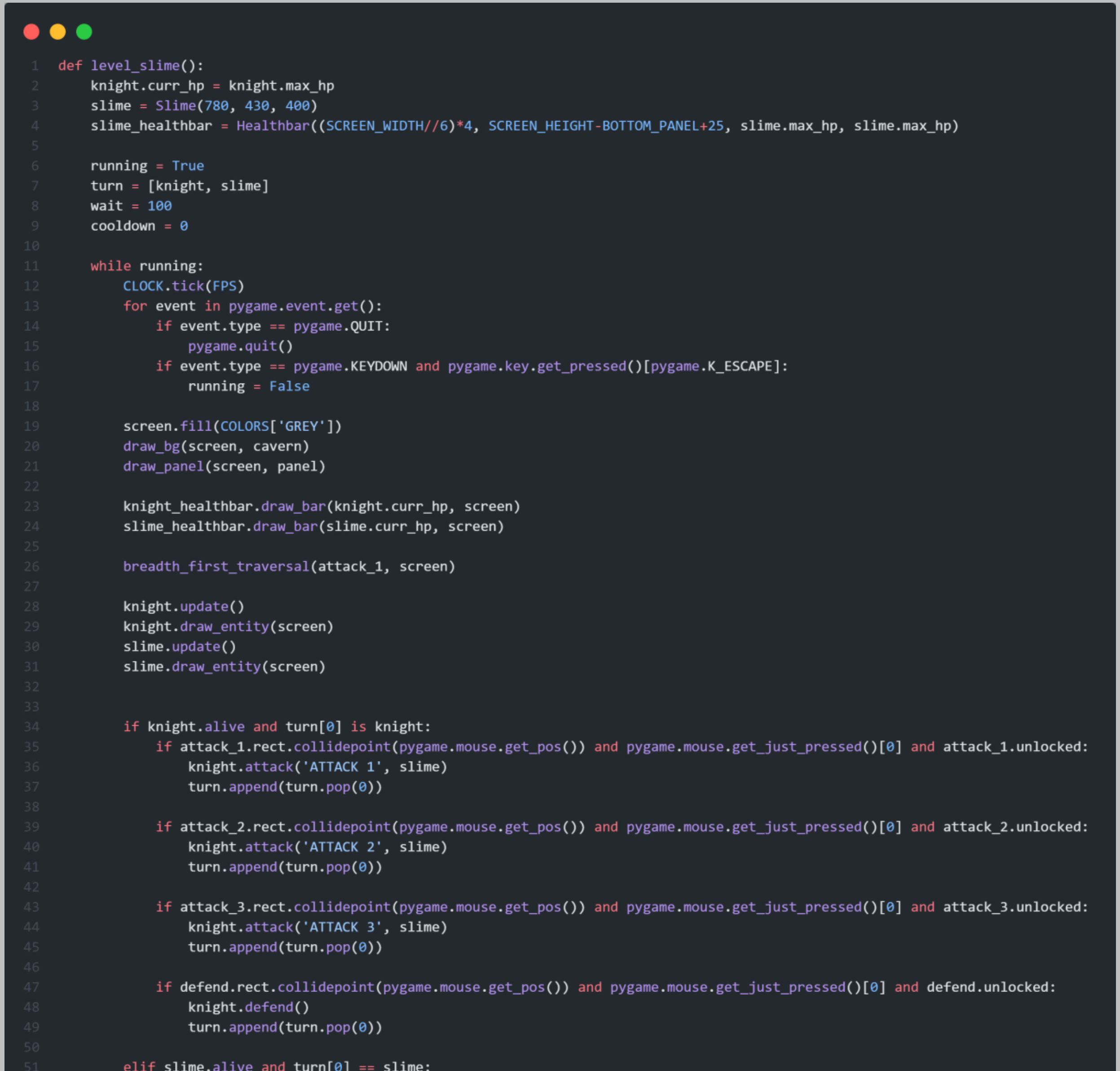




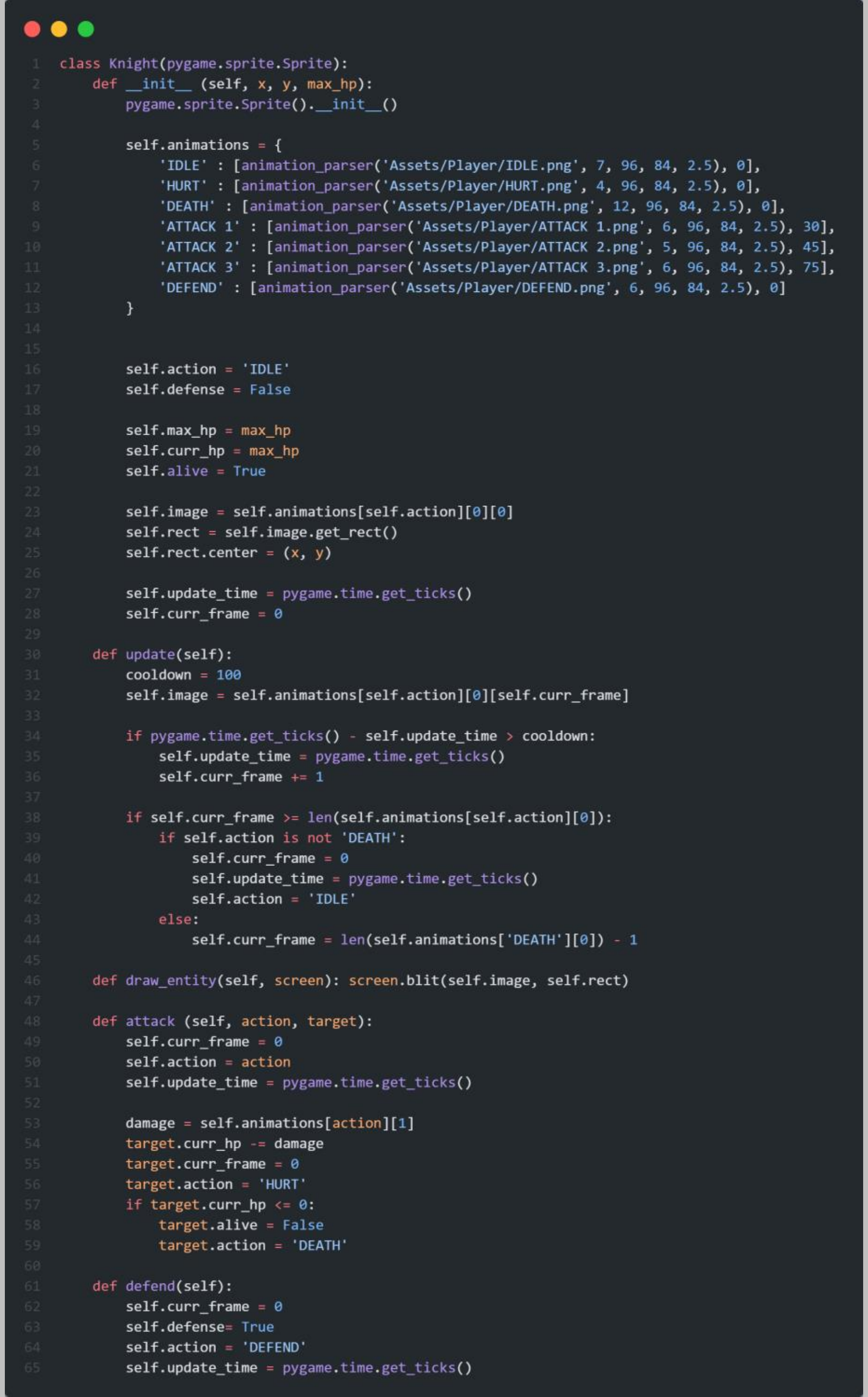




Code Implementation



```python
def level_slime():
    knight.curr_hp = knight.max_hp
    slime = Slime(780, 430, 400)
    slime_healthbar = Healthbar((SCREEN_WIDTH//6)*4, SCREEN_HEIGHT-BOTTOM_PANEL+25, slime.max_hp, slime.max_hp)

    running = True
    turn = [knight, slime]
    wait = 100
    cooldown = 0

    while running:
        CLOCK.tick(FPS)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
            if event.type == pygame.KEYDOWN and pygame.key.get_pressed()[pygame.K_ESCAPE]:
                running = False

        screen.fill(COLORS['GREY'])
        draw_bg(screen, cavern)
        draw_panel(screen, panel)

        knight_healthbar.draw_bar(knight.curr_hp, screen)
        slime_healthbar.draw_bar(slime.curr_hp, screen)

        breadth_first_traversal(attack_1, screen)

        knight.update()
        knight.draw_entity(screen)
        slime.update()
        slime.draw_entity(screen)


        if knight.alive and turn[0] is knight:
            if attack_1.rect.collidepoint(pygame.mouse.get_pos()) and pygame.mouse.get_just_pressed()[0] and attack_1.unlocked:
                knight.attack('ATTACK 1', slime)
                turn.append(turn.pop(0))

            if attack_2.rect.collidepoint(pygame.mouse.get_pos()) and pygame.mouse.get_just_pressed()[0] and attack_2.unlocked:
                knight.attack('ATTACK 2', slime)
                turn.append(turn.pop(0))

            if attack_3.rect.collidepoint(pygame.mouse.get_pos()) and pygame.mouse.get_just_pressed()[0] and attack_3.unlocked:
                knight.attack('ATTACK 3', slime)
                turn.append(turn.pop(0))

            if defend.rect.collidepoint(pygame.mouse.get_pos()) and pygame.mouse.get_just_pressed()[0] and defend.unlocked:
                knight.defend()
                turn.append(turn.pop(0))

        elif slime.alive and turn[0] == slime:
```



```python
class Knight(pygame.sprite.Sprite):
    def __init__ (self, x, y, max_hp):
        pygame.sprite.Sprite().__init__()

        self.animations = {
            'IDLE' : [animation_parser('Assets/Player/IDLE.png', 7, 96, 84, 2.5), 0],
            'HURT' : [animation_parser('Assets/Player/HURT.png', 4, 96, 84, 2.5), 0],
            'DEATH' : [animation_parser('Assets/Player/DEATH.png', 12, 96, 84, 2.5), 0],
            'ATTACK 1' : [animation_parser('Assets/Player/ATTACK 1.png', 6, 96, 84, 2.5), 30],
            'ATTACK 2' : [animation_parser('Assets/Player/ATTACK 2.png', 5, 96, 84, 2.5), 45],
            'ATTACK 3' : [animation_parser('Assets/Player/ATTACK 3.png', 6, 96, 84, 2.5), 75],
            'DEFEND' : [animation_parser('Assets/Player/DEFEND.png', 6, 96, 84, 2.5), 0]
        }


        self.action = 'IDLE'
        self.defense = False

        self.max_hp = max_hp
        self.curr_hp = max_hp
        self.alive = True

        self.image = self.animations[self.action][0][0]
        self.rect = self.image.get_rect()
        self.rect.center = (x, y)

        self.update_time = pygame.time.get_ticks()
        self.curr_frame = 0

    def update(self):
        cooldown = 100
        self.image = self.animations[self.action][0][self.curr_frame]

        if pygame.time.get_ticks() - self.update_time > cooldown:
            self.update_time = pygame.time.get_ticks()
            self.curr_frame += 1

        if self.curr_frame >= len(self.animations[self.action][0]):
            if self.action is not 'DEATH':
                self.curr_frame = 0
                self.update_time = pygame.time.get_ticks()
                self.action = 'IDLE'
            else:
                self.curr_frame = len(self.animations['DEATH'][0]) - 1

    def draw_entity(self, screen): screen.blit(self.image, self.rect)

    def attack (self, action, target):
        self.curr_frame = 0
        self.action = action
        self.update_time = pygame.time.get_ticks()

        damage = self.animations[action][1]
        target.curr_hp -= damage
        target.curr_frame = 0
        target.action = 'HURT'
        if target.curr_hp <= 0:
            target.alive = False
            target.action = 'DEATH'

    def defend(self):
        self.curr_frame = 0
        self.defense= True
        self.action = 'DEFEND'
        self.update_time = pygame.time.get_ticks()
```