# Zelda

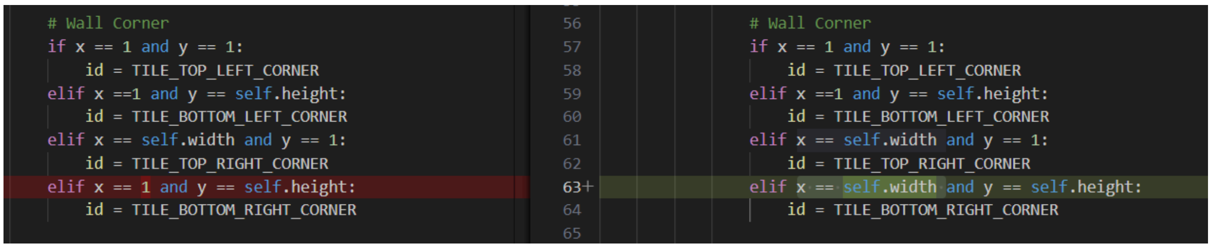
## Overview

This modified version of Zelda includes new features and improved functionality. Players can now interact with a pot object, which they can lift, carry, and throw. Additionally, a UFO object can be collected simply by walking past it, causing it to attach to the player. When the spacebar is pressed, the UFO moves randomly, destroying any skeletons it collides with. Another new feature is the Cloak of Invisibility, which appears when the player's health falls to 1. Collecting this cloak makes the player invisible to skeletons, allowing the player to walk past enemies without being damaged. When being invisible, the player can still throw pots, release UFOs and swing a sword to attack enemies. Finally, the doors only open when the player activates a switch after defeating all skeletons.

## Basic Adjustment

- **Fixing the Wall Corner**

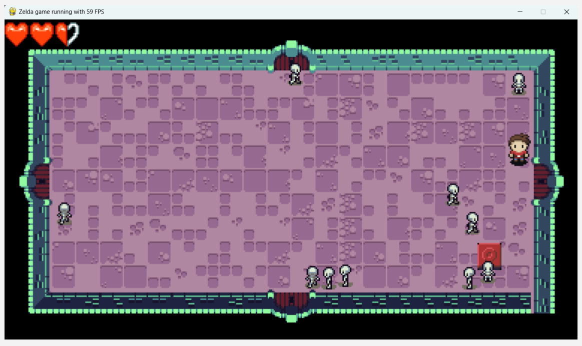
The bottom right corner of the wall was adjusted: x == 1 to x == self.width in Room.py



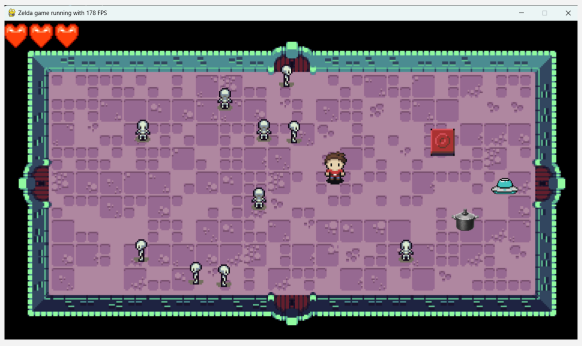
```
# Wall Corner
if x == 1 and y == 1:
    id = TILE_TOP_LEFT_CORNER
elif x ==1 and y == self.height:
    id = TILE_BOTTOM_LEFT_CORNER
elif x == self.width and y == 1:
    id = TILE_TOP_RIGHT_CORNER
elif x == 1 and y == self.height:
    id = TILE_BOTTOM_RIGHT_CORNER
```

```
# Wall Corner
if x == 1 and y == 1:
    id = TILE_TOP_LEFT_CORNER
elif x ==1 and y == self.height:
    id = TILE_BOTTOM_LEFT_CORNER
elif x == self.width and y == 1:
    id = TILE_TOP_RIGHT_CORNER
elif x == self.width and y == self.height:
    id = TILE_BOTTOM_RIGHT_CORNER
```

Bottom right corner - Before



Bottom right corner - After



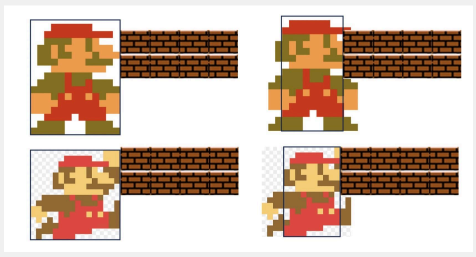
- **Framerate: from 60  to 200**

Panisara Srisan 6422781326

## New Collision function

A new Collision function was added in Room.py to make the collision between the player and pot, pot and skeleton, and UFO and skeleton looks more natural. The  idea is from lecture number 2 where we reduce the border for detecting collisions.

Image from Lecture 2



**def Collides_with_shrunken_border(self, skeleton, target, shrunken_border** = **10):**
This new function in Room.py takes three parameters: target1 and target2, which are the objects/entities to check for collision, and shrunken_border, which specifies the amount by which to shrink the detection area of each target.

```
    def Collides_with_shrunken_border(self, target1, target2, shrunken_border =
10):
        shrunken_target2_rect = pygame.Rect(
            target2.x + shrunken_border,
            target2.y + shrunken_border,
            target2.width - 2 * shrunken_border,
            target2.height - 2 * shrunken_border
        )
        shrunken_target1_rect = pygame.Rect(
            target1.x + shrunken_border,
            target1.y + shrunken_border,
            target1.width - 2 * shrunken_border,
            target1.height - 2 * shrunken_border
        )
        return shrunken_target1_rect.colliderect(shrunken_target2_rect)
```

## New Conditions

- The doors will be opened only when the player walk pass the switch after all skeletons are destroyed
- The player can only carry/hold either pot or UFO at a time. The player also cannot attack a skeleton by sword when holding one of these items until the player presses the spacebar to throw/ release the pot or UFO.

## Pot Object

Pot object was added for the player to lift, carry, and throw to damage a skeleton.

### Pot

Added gPot_image_list in resources.py

```
# Pot image
gPot_image = pygame.image.load("./graphics/Pot.png")
gPot_image_resized = pygame.transform.scale(gPot_image, (55, 45))
gPot_image_list = [gPot_image_resized]
```

Implemented it in class ObjectConf (including 3 states: standstill, picked, thrown)

```
'pot': ObjectConf('pot', gPot_image_list, 1, False, "standstill", {'standstill':0,
'picked':0, 'thrown':0}, width=55, height=45),
```

Added Pot object to Room.py

```
    self.pot = GameObject(GAME_OBJECT_DEFS['pot'],
                                  x=random.randint(MAP_RENDER_OFFSET_X + TILE_SIZE,
WIDTH-TILE_SIZE*2 - 48),
                                  y=random.randint(MAP_RENDER_OFFSET_Y+TILE_SIZE,
HEIGHT-(HEIGHT-MAP_HEIGHT*TILE_SIZE) + MAP_RENDER_OFFSET_Y - TILE_SIZE - 48))
```

Room.py, if the player collides with pot, pot state changes to ‘picked’. If state = ‘picked’,  a pot will stick to the player.

```
        def pot_collide():
            if self.Collides_with_shrunken_border(self.player, self.pot, 15) == True:
                if self.pot.state == "standstill" and self.ufo.state != 'picked':
                    gSounds['pick_item'].play()
                    self.player.ChangeState('benddown_pot')
                    self.pot.state = "picked"

        def pot_update(dt):
            if self.pot.state == "picked" and self.player.is_carrypotidle == True:
                self.pot.x = self.player.x - 3
                self.pot.y = self.player.y - 20
```

Room.py: If a pot state = thrown: move straightly based on the player direction when throwing

```
            if self.pot.state == "thrown":
                # TODO: write all four direction cases
                if self.pot.direction == 'right':
                    self.pot.x += dt * 300
                elif self.pot.direction == 'left':
```

Panisara Srisan 6422781326

```
                    self.pot.x -= dt * 300
                elif self.pot.direction == 'up':
                    self.pot.y -= dt * 300
                elif self.pot.direction == 'down':
                    self.pot.y += dt * 300
```

Room.py: If a pot collides with a skeleton after being thrown, a skeleton will be destroyed. A pot will also disappear.

```
        for entity in self.entities:
            # TODO new collision function
            if self.Collides_with_shrunken_border(entity, self.pot, 10) and
self.pot.state == 'thrown':
                if self.pot in self.objects:
                    gSounds['hit_enemy'].play()
                    self.entities.remove(entity)
                    self.objects.remove(self.pot)
```

Room.py: If a Pot collides with the wall after being thrown, pot will be destroyed.

```
                    # TODO: check wall collision
                if self.pot in self.objects:
                    if self.pot.x <= MAP_RENDER_OFFSET_X + TILE_SIZE:
                        self.objects.remove(self.pot)
                    if self.pot.x + self.pot.width >= WIDTH - TILE_SIZE * 2:
                        self.objects.remove(self.pot)
                    if self.pot.y <= MAP_RENDER_OFFSET_Y + TILE_SIZE -
self.pot.height /2:
                        self.objects.remove(self.pot)
                    bottom_edge = HEIGHT - (HEIGHT - MAP_HEIGHT * TILE_SIZE) +
MAP_RENDER_OFFSET_Y - TILE_SIZE
                    if self.pot.y + self.pot.height >= bottom_edge:
                        self.objects.remove(self.pot)
```

## Player Animation for Pot object

I wrote CharacterBendDown.json and CharacterPotWalk.json to handle character_pot_lift.png and character_pot_walk.png. Added json files in Util.py. Then added in gPlayer_animation_list in resource.py

```
gPlayer_animation_list = {"down": sprite_collection["character_walk_down"].animation,
                          "right": sprite_collection["character_walk_right"].animation,
                          "up": sprite_collection["character_walk_up"].animation,
                          "left": sprite_collection["character_walk_left"].animation,
                          "attack_down":
sprite_collection["character_attack_down"].animation,
                          "attack_right":
sprite_collection["character_attack_right"].animation,
                          "attack_up": sprite_collection["character_attack_up"].animation,
                          "attack_left":
sprite_collection["character_attack_left"].animation,
                          "pot_walkdown":
sprite_collection["character_pot_walk_down"].animation,
                         "pot_walkright":
sprite_collection["character_pot_walk_right"].animation,
                         "pot_walkup":
sprite_collection["character_pot_walk_up"].animation,
                         "pot_walkleft":
sprite_collection["character_pot_walk_left"].animation,
                         "pot_benddown_down":
sprite_collection["character_benddown_down"].animation,
                         "pot_benddown_right":
sprite_collection["character_benddown_right"].animation,
                         "pot_benddown_up":
sprite_collection["character_benddown_up"].animation,
                         "pot_benddown_left":
sprite_collection["character_benddown_left"].animation
```

3 new states including benddown_pot, carry_pot_idle, and carry_pot_walk were added in class PlayState.

```
        self.player.state_machine.SetStates({
            'walk': PlayerWalkState(self.player, self.dungeon),    # Define animation state
            'idle': PlayerIdleState(self.player),
            'swing_sword': PlayerAttackState(self.player, self.dungeon),
            'benddown_pot': PlayerBendDownState(self.player),
            'carry_pot_idle': PlayerCarryPotIdleState(self.player),
            'carry_pot_walk': PlayerCarryPotWalkState(self.player, self.dungeon)
        })
```

Benddown_pot state was implemented in PlayerBendDown.py. When the player collides with a pot, the player will bend down to lift the pot. I used a timer to freeze the player when bending down for about 0.3 second. After that, the state will be changed to carry_pot_idle.

```
class PlayerBendDownState(EntityIdleState):
    def __init__(self, player):
        super(PlayerBendDownState, self).__init__(player)

        self.player = player
        self.player.curr_animation.Refresh()
        self.player.ChangeAnimation("pot_benddown_"+self.player.direction)
        self.counter: float = 0
        self.countsecondpassed: float = 0

    def Enter(self, params):
        self.entity.offset_y = 15
        self.entity.offset_x = 0
        super().Enter(params)

    def Exit(self):
        pass



     def update(self, dt, events):
        def freezeplayer(self, dt):
            self.counter += dt
            if self.counter >= 0.3:
```

```
            self.entity.ChangeState('carry_pot_idle')
            self.counter = 0


    pressedKeys = pygame.key.get_pressed()
    if pressedKeys[pygame.K_LEFT]:
        self.entity.direction = 'left'
        self.entity.ChangeAnimation('pot_benddown_left')
        freezeplayer(self, dt)
    elif pressedKeys[pygame.K_RIGHT]:
        self.entity.direction = 'right'
        self.entity.ChangeAnimation('pot_benddown_right')
        freezeplayer(self, dt)
    elif pressedKeys[pygame.K_DOWN]:
        self.entity.direction = 'down'
        self.entity.ChangeAnimation('pot_benddown_down')
        freezeplayer(self, dt)
    elif pressedKeys[pygame.K_UP]:
        self.entity.direction = 'up'
        self.entity.ChangeAnimation('pot_benddown_up')
        freezeplayer(self, dt)
    else:
        self.entity.ChangeAnimation("pot_benddown_"+self.player.direction)
        freezeplayer(self, dt)
```

Carry_pot_idle state was implemented in PlayerCarryPotIdleState.py. It gets the player's direction before changing into this state to correctly display the player's current direction. After the player presses any movement keys, the state will be changed to 'carry_pot_walk, But if the spacebar is pressed (player throwing pot), the state will be changed to idle.

```
from src.states.entity.EntityIdleState import EntityIdleState
import pygame
from src.recourses import *

class PlayerCarryPotIdleState(EntityIdleState):
    def __init__(self, player):
        super(PlayerCarryPotIdleState, self).__init__(player)

        self.player = player
        self.player.curr_animation.Refresh()
        self.player.ChangeAnimation("pot_walk"+self.player.direction)

    def Enter(self, params):
        self.entity.offset_y = 15
        self.entity.offset_x = 0
        super().Enter(params)
        self.player.curr_animation.Refresh()
        self.player.ChangeAnimation("pot_walk"+self.player.direction)

    def Exit(self):
        pass

    def update(self, dt, events):
        self.player.is_carrypotidle = True
        pressedKeys = pygame.key.get_pressed()
        if pressedKeys[pygame.K_LEFT] or pressedKeys [pygame.K_RIGHT] or pressedKeys
[pygame.K_UP] or pressedKeys [pygame.K_DOWN]:
            self.entity.ChangeState('carry_pot_walk')

        for event in events:
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_SPACE:
                    gSounds['release_object'].play()
                    self.player.is_carrypotidle = False
                    self.entity.ChangeState('idle')
```

Carry_pot_walk was implemented in PlayerCarryPotWalkState.py. The player state will be reseted to idle if the player walks past the doors while holding pot.

```
class PlayerCarryPotWalkState(EntityWalkState):
    def __init__(self, player, dungeon):
        super(PlayerCarryPotWalkState, self).__init__(player, dungeon)

        self.entity.ChangeAnimation('down')
        self.dungeon = dungeon

    def Exit(self):
        pass

    def Enter(self, params):
        self.entity.offset_y = 15
        self.entity.offset_x = 0

    def update(self, dt, events):
        pressedKeys = pygame.key.get_pressed()
        if pressedKeys[pygame.K_LEFT]:
            self.entity.direction = 'left'
            self.entity.ChangeAnimation('pot_walkleft')
        elif pressedKeys[pygame.K_RIGHT]:
            self.entity.direction = 'right'
            self.entity.ChangeAnimation('pot_walkright')
        elif pressedKeys[pygame.K_DOWN]:
            self.entity.direction = 'down'
            self.entity.ChangeAnimation('pot_walkdown')
        elif pressedKeys[pygame.K_UP]:
            self.entity.direction = 'up'
            self.entity.ChangeAnimation('pot_walkup')
        else:
            self.entity.ChangeState('carry_pot_idle')

        for event in events:
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_SPACE:
                    gSounds['release_object'].play()
                    self.entity.is_carrypotidle = False
                    self.entity.ChangeState('idle')

        #move and bump to the wall check
        super().update(dt, events)
        if self.bumped:
            if self.entity.direction == 'left':
                #temporal move to the wall (bumping effect)
                self.entity.x = self.entity.x - PLAYER_WALK_SPEED * dt

                for doorway in self.dungeon.current_room.doorways:
                    # self.entity.is_invisible = False
                    if self.entity.Collides(doorway) and doorway.open:
                        self.entity.ChangeState('idle')
                        self.entity.y = doorway.y + 12
                        self.dungeon.BeginShifting(-WIDTH, 0)

                self.entity.x = self.entity.x + PLAYER_WALK_SPEED * dt

            elif self.entity.direction == 'right':
                self.entity.x = self.entity.x + PLAYER_WALK_SPEED * dt

                for doorway in self.dungeon.current_room.doorways:
                    if self.entity.Collides(doorway) and doorway.open:
                        self.entity.ChangeState('idle')
                        self.entity.y = doorway.y + 12
                        self.dungeon.BeginShifting(WIDTH, 0)

                self.entity.x = self.entity.x - PLAYER_WALK_SPEED * dt

            elif self.entity.direction == 'up':
                self.entity.y = self.entity.y - PLAYER_WALK_SPEED * dt

                for doorway in self.dungeon.current_room.doorways:
                    if self.entity.Collides(doorway) and doorway.open:
                        self.entity.ChangeState('idle')
                        self.entity.y = doorway.x + 24
                        self.dungeon.BeginShifting(0,  -HEIGHT)

                self.entity.y = self.entity.y + PLAYER_WALK_SPEED * dt

            else:
                self.entity.y = self.entity.y + PLAYER_WALK_SPEED * dt

                for doorway in self.dungeon.current_room.doorways:
                    if self.entity.Collides(doorway) and doorway.open:
                        self.entity.ChangeState('idle')
                        self.entity.y = doorway.x + 24
                        self.dungeon.BeginShifting(0,  HEIGHT)

                self.entity.y = self.entity.y - PLAYER_WALK_SPEED * dt
```

Panisara Srisan 6422781326

## New Features

### 1) UFO

Added gUfo_image_list in resource.py

```
# TODO UFO
gUfo_image = pygame.image.load("./graphics/ufo.png")
gUfo_image_resized = pygame.transform.scale(gUfo_image, (55, 33))
gUfo_image_list = [gUfo_image_resized]
```

Added UFO in class ObjectConf

```
 # TODO UFO
    'ufo' : ObjectConf('ufo', gUfo_image_list, 1, False, "standstill",
{'standstill':0, 'picked':0, 'thrown':0}, width=55, height=33)
```

Room.py: UFO added

```
        self.ufo = GameObject(GAME_OBJECT_DEFS['ufo'],
                              x=random.randint(MAP_RENDER_OFFSET_X + TILE_SIZE,
WIDTH-TILE_SIZE*2 - 48),
                              y=random.randint(MAP_RENDER_OFFSET_Y+TILE_SIZE,
HEIGHT-(HEIGHT-MAP_HEIGHT*TILE_SIZE) + MAP_RENDER_OFFSET_Y - TILE_SIZE - 48))
```

Room.py: if the player collides with UFO, the player state will be changed to carry_pot_idle (actually, it is just a state for animation when the player is holding something, so I borrowed this state from pot object). The UFO will stick to the player just like pot object. If the UFO is released (by pressing spacebar), it will move randomly and will bounce back when colliding with the wall.

```
        def ufo_collide():
            if self.ufo.state == "standstill" and self.pot.state != 'picked':
                gSounds['pick_item'].play()
                self.ufo.state = "picked"
                print("ufo picked")
                self.player.ChangeState("carry_pot_idle")
        def ufo_update(dt):
            if self.ufo.state == "picked":
                self.ufo.x = self.player.x - 3
                self.ufo.y = self.player.y - 20

            if self.ufo.state == "thrown":
                self.ufo.x += self.ufo_dx * dt
                self.ufo.y += self.ufo_dy * dt
                # Constrain within walls and bounce on collision
                if self.ufo.x <= MAP_RENDER_OFFSET_X + TILE_SIZE:
                    self.ufo.x = MAP_RENDER_OFFSET_X + TILE_SIZE
                    self.ufo_dx *= -1
                elif self.ufo.x + self.ufo.width >= WIDTH - TILE_SIZE * 2:
                    self.ufo.x = WIDTH - TILE_SIZE * 2 - self.ufo.width
                    self.ufo_dx *= -1

                if self.ufo.y <= MAP_RENDER_OFFSET_Y + TILE_SIZE - self.ufo.height
/ 2:
                    self.ufo.y = MAP_RENDER_OFFSET_Y + TILE_SIZE - self.ufo.height
/ 2
                    self.ufo_dy *= -1
                elif self.ufo.y + self.ufo.height >= HEIGHT - (HEIGHT - MAP_HEIGHT
* TILE_SIZE) + MAP_RENDER_OFFSET_Y - TILE_SIZE:
                    self.ufo.y = HEIGHT - (HEIGHT - MAP_HEIGHT * TILE_SIZE) +
MAP_RENDER_OFFSET_Y - TILE_SIZE - self.ufo.height
                    self.ufo_dy *= -1
```

Room.py: UFO can destroy skeletons if it collides with them

```
            elif self.Collides_with_shrunken_border(entity, self.ufo, 3) and
self.ufo.state == 'thrown':
                if self.ufo in self.objects:
                    gSounds['hit_enemy'].play()
                    self.entities.remove(entity)
```

## 2) The Cloak of Invisibility

Room,py: When the player's health is equal to 1 (player only has half of a heart left), the cloak of invisibility will appear.

```
        if self.player.health <= 1 and self.player.is_invisible == False:
            if self.cloak not in self.objects:
                self.objects.append(self.cloak)
```

If the player walks past the cloak, the opacity of the player will be reduced in Player.py

```
        # TODO Cloak
        if self.is_invisible == True:
            self.curr_animation.image.set_alpha(120)
            if self.curr_animation.idleSprite != None:
                self.curr_animation.idleSprite.set_alpha(120)
        else:
            self.curr_animation.image.set_alpha(255)
            if self.curr_animation.idleSprite != None:
                self.curr_animation.idleSprite.set_alpha(255)
```

Meaning that the player is invisible to skeletons and will not get damaged when collided with them. The below code is in Room.py. Boolean is_invisible in class Player is used to check whether the player is invisible.

```
            if not entity.is_dead and self.player.Collides(entity) and not
self.player.invulnerable:
                # TODO Cloak
                # print("check invisible after collide with skeleton",
self.player.is_invisible)
                if self.player.is_invisible == False:
                    gSounds['hit_player'].play()
                    self.player.Damage(1)
                    self.player.SetInvulnerable(1.5)
```

The player can still attack the skeletons by sword, pot, or UFO when being invisible.