Evan Shearer & Seth Rapp

CS2021

April 2017

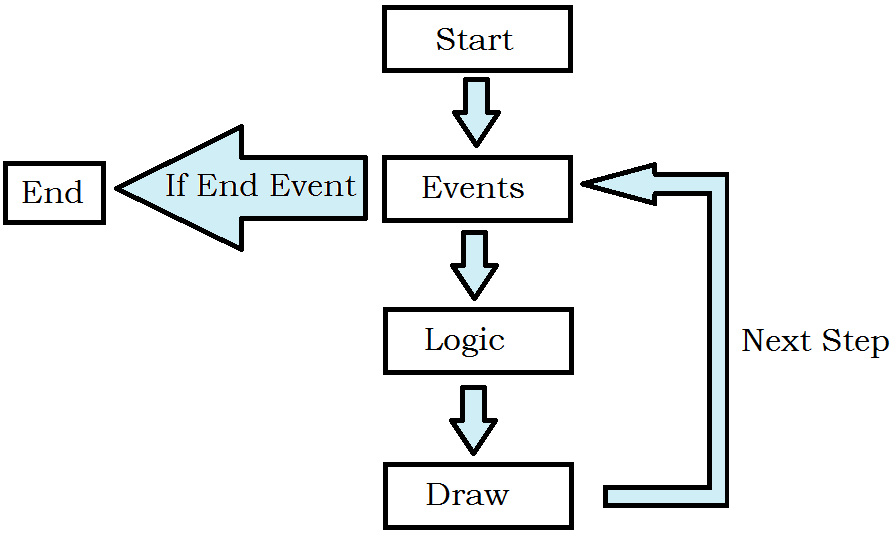
Final Project Report

Introduction

The project goal was to use an external code library for python to create a presentable project. The Library we chose to use was Pygame. Pygame is a library with classes that are meant to be used to code computer games. The goal we set for ourselves was to use Pygame to create a simple game that is playable. We chose that instead of writing code that meets its specific purpose, we chose to write code that could easily be rewritten for future works, or at least easy to change later.

When choosing the structure of our game we considered what Pygame can do, in addition to what would be easy to model beforehand. We chose to make a 2D platformer. We considered doing a top down platformer, but doing so would require more moving parts for the same complexity of game. We did not want to get bogged down by a lot of objects interacting, so we went with the 2D platformer option.

The game runs off three types of files: the main file(ensgame.py), the game file(game.py), and the sprite files. The main file creates and starts an instance of Pygame while declaring the clock and screen objects. Then the main creates an instance of the game and steps through the three logic steps at 60 ticks per second. The three logic steps that get run from the main are the handle\_events, logic, and draw\_frame.



**Game Flow Diagram**

The steps exist in the game file as functions. The handle\_events gets user input and makes decisions based on what buttons are pressed. The logic step allows for decisions to be made based on the current game state. Lastly, the draw\_frame step allows for all the current objects stored in the game to be drawn to the screen that the user sees.

The final part of the project is the sprite files. The sprite files include objects such as blocks and the player. The sprite files are children of the Sprite class that is a part of Pygame’s library. They contain data such as their position and image. The sprite children also have added information based on what actions they need to do and logic that is needed to be performed. These files and objects together form the game we set out to make.

Project Results

The original goal was to make a 2D platformer that played as a Metroidvania. A Metroidvania is a game style that typically involves progressing through a system of room and having to backtrack as new powers or abilities are obtained. The main issue with trying this would not come from level design, but the fact that a Metroidvania would need to incorporate a lot of new mechanics and abilities. Due to the long time it took to make a workable environment with a controllable player, many defining aspects of Metroidvanias were not implemented into out game.

Our game still succeeded on many aspects we wanted to have. The environment consists of rooms that the player can travel between and backtrack through. With the inclusion of splitting paths, this system of rooms slightly mirrors the style we were aiming for. Another pothole in our design was the fact that neither of us were artistic. With no artistic ability between us, we were left with very dull and poorly made sprites.

Looking at the goal to “use Pygame to create a playable game”, we succeeded. The game runs from opening ensgame.py with all other files in the same directory. Python and Pygame are needed to also be installed, but this is easily done with a machine with Python3 and running ‘pip install Pygame’. A python program exporter could be used to get it to run without such, but was not needed.

Since there were many resources online to learn the Pygame library and common practices, the hardest issue was the time constraint. The time given to finish the game was plenty, but not enough to create a full game with many features. The default viewing windows resolution and windowned state was a decision made to focus time on designing the game.

Division of Group Work

There were only two members, Evan and Seth, working on this project; because of this, work mitigation and sharing of responsibilities was easy. Evan started by researching Pygame and finding useful resources to self-teach ourselves Pygame. We collaborated, discussing game design and possibilities for the game’s type. Seth, in the end, decided that a 2D Metroidvania-esc game would be better in terms of quality than if we produce a top-down game.

Evan started the code by making the ensgame.py file and creating the game.py file. We both worked together to sort out the sprite’s children classes and determined how the player exists and interacts with other objects. Evan wrote most the code for the block and player classes while Seth wrote all the room classes, including their parent class. In the end, all the code was finished up by them both. The sprites and backgrounds were made by Evan, while Seth did some touch-up details.

Self-Assessment Paragraphs:

By Evan:

The project was a very fun thing to work on. My contributions were many but could not have made it all without Seth. I ended up making the platform for the game to run on (ensgame.py and game.py) as well as most of the sprites and backgrounds used. I made the simple item class that was a child of Sprites so that the player and block classes could be used consistently. I also made the player class and the block classes for the most part. I had also made color.py which was meant for easy RGB color usage, but was almost never used.

By Seth:

This was enjoyable and frustrating at the same time. Until Evan finished the platform, I mostly contributed by helping him make design decisions, such as jump heights and decay factors. After the platform was finished, I helmed the level designs and implemented the layout. I drew a grid on a piece of paper, designed the room, wrote pseudo-code for how to implement it in room.py, then hard-coded it, and tested the room. I would then evaluate the difficulty of the room, take note of bugs, and if the room was unique and enjoyable.

Bibliography

Craven, Paul Vincent. "Program Arcade Games With Python And Pygame." *Program Arcade Games With Python And Pygame*. N.p., 2015. Web. 26 Apr. 2017.

Code Appendix

#ensgame.py

#Contains the main and initializes both the Game and Pygame

#ensgame stands for Evan and Seth's game

import pygame as p

from game import Game

import constants

def main():

#Initializes pygame

p.init()

#Creates a display that the game prints objects to

size = [constants.SCREEN\_WIDTH, constants.SCREEN\_HEIGHT]

display = p.display.set\_mode(size)

#Sets window's name and icon

p.display.set\_caption('Evan and Seth\'s Game!')

p.display.set\_icon(p.image.load('Images\\icon.png'))

#If True, the game closes and ends

done = False

#Clock lets the game pause for a tiny bit so that the target framerate is not exceeded

clock = p.time.Clock()

#initializes Game from game.py

game = Game()

while not done:

done = game.handle\_events()

game.logic()

game.draw\_frame(display)

clock.tick(60)#Target framerate is 60 frames per second

p.quit()

if \_\_name\_\_ == '\_\_main\_\_':

main()

#game.py

#Contains all the information needed to run the game

import pygame as p

from item import Item

from block import Block

from player import Player

from room import \*

import color

class Game():

def \_\_init\_\_(self):

#Flag for if player finds secret room

self.flag = 0

#Initialize list of rooms

self.rooms = []

room = Room0()

self.rooms.append(room)

room = Room1()

self.rooms.append(room)

room = Room2()

self.rooms.append(room)

room = Room3()

self.rooms.append(room)

room = Room4()

self.rooms.append(room)

room = Room5()

self.rooms.append(room)

room = Room6()

self.rooms.append(room)

room = Room7()

self.rooms.append(room)

room = Room8()

self.rooms.append(room)

room = Room9()

self.rooms.append(room)

room = Room10()

self.rooms.append(room)

room = Room11()

self.rooms.append(room)

room = Room12()

self.rooms.append(room)

room = Room13()

self.rooms.append(room)

#Initialize starting room

self.current\_room = self.rooms[0]

#Initialize all the blocks and sprites

self.blocks = p.sprite.Group()

self.blocks = p.sprite.Group()

self.sprites = p.sprite.Group()

#Starting position for player

self.player = Player(pos = (200,200))

self.sprites.add(self.player)

self.blocks = self.current\_room.block\_list

self.paused = False

for block in self.blocks:

block.followers = self.sprites

def handle\_events(self):

#Event handling here

for event in p.event.get():

#If statement to end game

if event.type == p.QUIT:

return True

#Key press events

elif event.type == p.KEYDOWN:

if event.key == p.K\_SPACE:

self.player.jump\_start()

elif event.key == p.K\_a or event.key == p.K\_LEFT:

self.player.left = True

elif event.key == p.K\_d or event.key == p.K\_RIGHT:

self.player.right = True

elif event.key == p.K\_ESCAPE:

self.paused = not self.paused

elif event.type == p.KEYUP:

if event.key == p.K\_SPACE:

self.player.jump\_stop()

elif event.key == p.K\_a or event.key == p.K\_LEFT:

self.player.left = False

elif event.key == p.K\_d or event.key == p.K\_RIGHT:

self.player.right = False

return False

def logic(self):

#Logic goes here

#Player goes too far right

if self.player.rect.x > constants.SCREEN\_WIDTH - 16:

if self.current\_room.right != None:

self.current\_room = self.rooms[self.current\_room.right]

self.current\_room.\_\_init\_\_()

self.player.rect.x = -16

#Player goes too far left

if self.player.rect.x < -16:

if self.current\_room.left != None:

self.current\_room = self.rooms[self.current\_room.left]

self.current\_room.\_\_init\_\_()

self.player.rect.x = constants.SCREEN\_WIDTH - 16

#Player goes too far down

if self.player.rect.y > constants.SCREEN\_HEIGHT:

if self.current\_room.down != None:

self.current\_room = self.rooms[self.current\_room.down]

self.current\_room.\_\_init\_\_()

self.player.rect.y = 0

#Player goes too far up

if self.player.rect.y < 0:

if self.current\_room.up != None:

self.current\_room = self.rooms[self.current\_room.up]

self.current\_room.\_\_init\_\_()

self.player.rect.y = constants.SCREEN\_HEIGHT

#Player found secret room

if self.current\_room.id == 6:

self.flag = 1

elif self.current\_room.id == 13:

if self.flag == 1:

self.current\_room.update()

#Update blocks if room changed

self.blocks = self.current\_room.block\_list

for block in self.blocks:

block.followers = self.sprites

if self.paused == False:

self.player.blocks = self.blocks # Get player from list first

self.sprites.update()

self.blocks.update()

def draw\_frame(self, screen):

#Drawing goes here

if self.paused == False:

screen.blit(self.current\_room.background, dest = (0,0))

self.sprites.draw(screen)

self.blocks.draw(screen)

p.display.flip()

#room.py

#Contains classes of every room in the room, as well as the parent class

import pygame as p

import constants

from block import Block

#Parent room class with basic data

class Room():

def \_\_init\_\_(self):

#Integer value of room

self.id = None

#References to adjacent rooms

self.left = None

self.right = None

self.up = None

self.down = None

#Initialize group of blocks

self.block\_list = p.sprite.Group()

#Initialize default background

self.background = p.image.load("Images\\default\_background.png")

class Room0(Room):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.id = 0

self.left = None

self.right = 1

self.up = None

self.down = None

i = 0

while i <= constants.SCREEN\_WIDTH:

self.block\_list.add(Block(pos = (i,0)))

self.block\_list.add(Block(pos = (i, constants.SCREEN\_HEIGHT-64)))

self.block\_list.add(Block(pos = (0, i+64)))

i += 64

class Room1(Room):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.id = 1

self.left = 0

self.right = 2

self.up = None

self.down = None

i = 0

while i <= constants.SCREEN\_WIDTH:

self.block\_list.add(Block(pos = (i,0)))

self.block\_list.add(Block(pos = (i, constants.SCREEN\_HEIGHT-64)))

if i >= 320 and i <= 448:

self.block\_list.add(Block(pos = (704, i)))

i += 64

self.block\_list.add(Block(pos = (320, 384)))

self.block\_list.add(Block(pos = (512, 256)))

class Room2(Room):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.id = 2

self.left = 1

self.right = 3

self.up = 6

self.down = None

i = 0

while i < constants.SCREEN\_WIDTH:

if i <= 192 or i >= 512:

self.block\_list.add(Block(pos = (i,0)))

if i >= 320 and i <= 448:

self.block\_list.add(Block(pos = (0, i)))

self.block\_list.add(Block(pos = (constants.SCREEN\_WIDTH-64, i)))

if i <= 256 or i >= 448:

self.block\_list.add(Block(pos = (i, constants.SCREEN\_HEIGHT-64)))

i += 64

self.block\_list.add(Block(pos = (640,128), moving = (2,640,256,0,0,0)))

self.block\_list.add(Block(pos = (64,384)))

self.block\_list.add(Block(pos = (64, 448)))

class Room3(Room):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.id = 3

self.left = 2

self.right = None

self.up = None

self.down = 4

i = 0

while i < constants.SCREEN\_WIDTH:

self.block\_list.add(Block(pos = (i,0)))

self.block\_list.add(Block(pos = (constants.SCREEN\_WIDTH-64, i+64)))

if i <= 448:

self.block\_list.add(Block(pos = (i, constants.SCREEN\_HEIGHT-64)))

if i >= 320:

self.block\_list.add(Block(pos = (0,i)))

i += 64

self.block\_list.add(Block(pos = (64,384)))

self.block\_list.add(Block(pos = (64, 448)))

class Room4(Room):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.id = 4

self.left = None

self.right = None

self.up = 3

self.down = 5

i = 0

while i < constants.SCREEN\_WIDTH:

if i <= 384:

self.block\_list.add(Block(pos = (i,0)))

self.block\_list.add(Block(pos = (448, i)))

if i > 128:

self.block\_list.add(Block(pos = (320,i)))

if i != 384:

self.block\_list.add(Block(pos = (i, constants.SCREEN\_HEIGHT-64)))

i += 64

self.block\_list.add(Block(pos = (512,384), moving = (2,704,512,0,0,0)))

self.block\_list.add(Block(pos = (0,256), moving = (4,192,0,0,0,0)))

class Room5(Room):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.id = 5

self.left = 10

self.right = 7

self.up = 4

self.down = None

self.background = p.image.load("Images\\fork.png")

i = 0

while i < constants.SCREEN\_WIDTH:

if i != 384:

self.block\_list.add(Block(pos = (i, 0)))

if i != 384 and i != 448:

self.block\_list.add(Block(pos = (0, i)))

self.block\_list.add(Block(pos = (i, constants.SCREEN\_HEIGHT-64)))

if i >= 64 and i <= 640:

self.block\_list.add(Block(pos = (i, constants.SCREEN\_HEIGHT-128)))

if i >= 128 and i <= 576:

self.block\_list.add(Block(pos = (i, constants.SCREEN\_HEIGHT-192)))

i += 64

class Room6(Room):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.id = 6

self.left = None

self.right = None

self.up = None

self.down = 2

self.background = p.image.load("Images\\secret\_background.png")

i = 0

while i < constants.SCREEN\_WIDTH:

if i != 320 and i != 384:

self.block\_list.add(Block(pos = (i, constants.SCREEN\_HEIGHT-64)))

i += 64

self.block\_list.add(Block(pos = (320,constants.SCREEN\_HEIGHT+64), moving = (0,0,0,2,576,320)))

self.block\_list.add(Block(pos = (384,constants.SCREEN\_HEIGHT+64), moving = (0,0,0,2,576,320)))

class Room7(Room):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.id = 7

self.left = 5

self.right = None

self.up = 8

self.down = None

i = 0

while i <= constants.SCREEN\_WIDTH:

self.block\_list.add(Block(pos = (i, constants.SCREEN\_HEIGHT-64)))

if i != 64:

self.block\_list.add(Block(pos = (i, 0)))

i += 64

self.block\_list.add(Block(pos = (384,448), moving = (0,0,0,1,448,256)))

self.block\_list.add(Block(pos = (256,192), moving = (2,320,128,0,0,0)))

self.block\_list.add(Block(pos = (64,256), moving = (0,0,0,2,256,0)))

class Room8(Room):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.id = 8

self.left = None

self.right = None

self.up = 9

self.down = 7

i = 0

while i <= constants.SCREEN\_WIDTH:

self.block\_list.add(Block(pos = (0, i)))

if i != 64:

self.block\_list.add(Block(pos = (i, constants.SCREEN\_HEIGHT-64)))

if i >= 192:

self.block\_list.add(Block(pos = (i, 448)))

if i >= 256:

self.block\_list.add(Block(pos = (i, 384)))

if i >= 320:

self.block\_list.add(Block(pos = (i, 320)))

if i >= 384:

self.block\_list.add(Block(pos = (i, 256)))

if i >= 448:

self.block\_list.add(Block(pos = (i, 192)))

if i >= 512:

self.block\_list.add(Block(pos = (i, 128)))

if i >= 576:

self.block\_list.add(Block(pos = (i, 64)))

i += 64

self.block\_list.add(Block(pos = (640, 0)))

self.block\_list.add(Block(pos = (704, 0)))

#Complete

class Room9(Room):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.id = 9

self.left = 13

self.right = None

self.up = None

self.down = 8

i = 0

while i < 768:

self.block\_list.add(Block(pos = (i, 0)))

self.block\_list.add(Block(pos = (704, i)))

if i != 512 and i != 576:

self.block\_list.add(Block(pos = (i, 512)))

i += 64

class Room10(Room):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.id = 10

self.left = None

self.right = 5

self.up = 11

self.down = None

i = 0

while i < 768:

self.block\_list.add(Block(pos = (0,i)))

if i != 384 and i != 448:

self.block\_list.add(Block(pos = (704,i)))

if i >= 64 and i <= 512:

self.block\_list.add(Block(pos = (320,i)))

i += 64

self.block\_list.add(Block(pos = (256, 512), moving = (2,256,64,0,0,0)))

self.block\_list.add(Block(pos = (64,0), moving = (0,0,0,2,448,0)))

class Room11(Room):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.id = 11

self.left = None

self.right = None

self.up = 12

self.down = 10

i = 0

while i < 768:

if i != 320 and i != 384:

self.block\_list.add(Block(pos = (i, 0)))

i += 64

self.block\_list.add(Block(pos = (320, 128), moving = (0,0,0,2,192,0)))

self.block\_list.add(Block(pos = (192, 192), moving = (2,256,64,0,0,0)))

self.block\_list.add(Block(pos = (512, 256), moving = (2,640,512,0,0,0)))

self.block\_list.add(Block(pos = (0, 320), moving = (2,128,0,0,0,0)))

self.block\_list.add(Block(pos = (384, 320), moving = (2,384,256,0,0,0)))

self.block\_list.add(Block(pos = (704, 448), moving = (2,640,512,0,0,0)))

self.block\_list.add(Block(pos = (0, 512)))

self.block\_list.add(Block(pos = (704, 512)))

class Room12(Room):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.id = 12

self.left = None

self.right = 13

self.up = None

self.down = 11

i = 0

while i < 768:

self.block\_list.add(Block(pos = (448, i)))

if i != 320 and i != 384:

self.block\_list.add(Block(pos = (i, 512)))

i += 64

class Room13(Room):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.id = 13

self.left = 12

self.right = 9

self.up = None

self.down = None

self.background = p.image.load("Images\\win.png")

i = 0

while i < 768:

self.block\_list.add(Block(pos = (i, 0)))

self.block\_list.add(Block(pos = (i, 512)))

i += 64

#In case player finds the secret room

def update(self):

self.background = p.image.load("Images\\true\_win.png")

#block.py

#Contains the block class used for the walls, floorws, etc.

import pygame as p

from item import Item

class Block(Item):

def \_\_init\_\_(self, pos=(0,0), moving=[0,0,0,0,0,0]):

super().\_\_init\_\_(pos=pos, img = p.image.load('Images\\block.png'))

self.moving = moving

self.followers = None

def update(self):

#for moving

moved\_y = False

if self.moving[0] > 0:

if self.rect.x < self.moving[1]:

if not self.followers == None:

self.rect.x += 2

self.rect.y -= 2

test = p.sprite.spritecollide(self, self.followers, False) # IF COLLISION, test >

self.rect.x -= 2

self.rect.y += 2

if len(test) > 0:

for follower in test:

follower.move\_x(self.moving[0])

self.rect.x += self.moving[0]

else:

self.moving = (-self.moving[0],self.moving[1],self.moving[2],self.moving[3],self.moving[4],self.moving[5])

elif self.moving[0] < 0:

if self.rect.x > self.moving[2]:

if not self.followers == None:

self.rect.x -= 2

self.rect.y -= 2

test = p.sprite.spritecollide(self, self.followers, False)

self.rect.x += 2

self.rect.y += 2

if len(test) > 0:

for follower in test:

follower.move\_x(self.moving[0])

self.rect.x += self.moving[0]

else:

self.moving = (-self.moving[0],self.moving[1],self.moving[2],self.moving[3],self.moving[4],self.moving[5])

if self.moving[3] > 0:

if self.rect.y < self.moving[4]:

if not self.followers == None:

#For pull down

self.rect.y -= 2

test = p.sprite.spritecollide(self, self.followers, False)

self.rect.y += 2

if len(test) > 0:

self.rect.y += self.moving[3]

for follower in test:

follower.move\_y(self.moving[3])

self.rect.y -= self.moving[3]

#for push down

self.rect.y += 2

test = p.sprite.spritecollide(self, self.followers, False)

self.rect.y -= 2

if len(test) > 0:

self.rect.y += self.moving[3]

for follower in test:

follower.move\_y(self.moving[3])

self.rect.y -= self.moving[3]

self.rect.y += self.moving[3]

else:

self.moving = (self.moving[0],self.moving[1],self.moving[2],-self.moving[3],self.moving[4],self.moving[5])

elif self.moving[3] < 0:

if self.rect.y > self.moving[5]:

if not self.followers == None:

self.rect.y -= 2

test = p.sprite.spritecollide(self, self.followers, False)

self.rect.y += 2

if len(test) > 0:

for follower in test:

follower.move\_y(self.moving[3])

self.rect.y += self.moving[3]

else:

self.moving = (self.moving[0],self.moving[1],self.moving[2],-self.moving[3],self.moving[4],self.moving[5])

#player.py

#Class for the player's character

import pygame as p

from item import Item

import constants

import color

class Player(Item):

def \_\_init\_\_(self, pos=(0,0)):

self.images = [p.image.load("Images\\stand.png"),p.image.load("Images\\walk.png"),p.image.load("Images\\jump.png"),p.image.load("Images\\fall.png"),p.image.load("Images\\hurt.png"),p.image.load("Images\\dead.png")]#Load all images here

super().\_\_init\_\_(pos=pos,img=p.image.load("Images\\stand.png"))

for im in self.images:

im.set\_colorkey(color.BLACK)

#Jump sounds

self.jump1 = p.mixer.Sound("Sounds\\jump.wav")

self.jump2 = p.mixer.Sound("Sounds\\jump2.wav")

#action decay

self.fallen = 0

self.jump = 0

self.shot = 0

self.dead = 0

self.hurt = 0

#for movement

self.xvel = 0

self.yvel = 0

#for some sprite calculations

self.clock = 120

#for collision testing

self.blocks = None

#for walking

self.right = False

self.left = False

def update(self):

#Internal Clock Tick

self.clock -= 1

if self.clock <= 0:

self.clock = 120

#Falling check

if not self.grounded() and self.jump == 0:#IF not grounded and not mid jump

if(self.clock % 3 == 0 or self.yvel == 0):#Calculate gravity

self.yvel = min(constants.TERMINAL\_VELOCITY, self.yvel + 1)

else:

self.yvel = min(0, self.yvel)#If falling, stop. Else: keep going(up)

if self.jump > 0:

self.yvel = -constants.JUMP\_SPEED

if self.jump == 1:

self.yvel = - (2 \* constants.JUMP\_SPEED) // 3

#Walking

if self.left and self.right:

self.xvel = self.xvel - self.sign(self.xvel)

elif self.right:

self.xvel = min(self.xvel + constants.WALK\_ACCELERATION, constants.TOP\_SPEED)

elif self.left:

self.xvel = max(self.xvel - constants.WALK\_ACCELERATION, -constants.TOP\_SPEED)

elif self.grounded():

self.xvel = self.xvel - self.sign(self.xvel)

#Move and Collision

collided\_y = self.move\_y(self.yvel)

if collided\_y:

#If collided on y, either

self.jump = 0 # if on top

self.yvel = 0 #either collision on top or bottom

collided\_x = self.move\_x(self.xvel)

if collided\_x:

#If collided on x, either

self.xvel = 0

#Sprite Calculations and Decay(If multiple frames for a sprite, use % on the decay factor or on clock)

if self.dead > 0:

self.image = self.images[5]

self.dead -= 1

elif self.yvel < 0 or self.jump > 0:

self.image = self.images[2]

self.jump = max(self.jump - 1, 0)

elif self.yvel > 0:

self.image = self.images[3]

elif not self.xvel == 0 and self.grounded():

self.image = self.images[1]

else:

self.image = self.images[0]

def move\_x(self, dist=1):

#return True on collsion, False otherwise

if not dist == 0:

tick = self.sign(dist)

for i in range(abs(dist)):

self.rect.x += tick

collide\_blocks = p.sprite.spritecollide(self, self.blocks, False)

if len(collide\_blocks) > 0:

self.rect.x -= tick

return True

return False

def move\_y(self, dist=1):

#return True on collsion, False otherwise

if not dist == 0:

tick = self.sign(dist)

for i in range(abs(dist)):

self.rect.y += tick

collide\_blocks = p.sprite.spritecollide(self, self.blocks, False)

if len(collide\_blocks) > 0:

self.rect.y -= tick

return True

return False

def jump\_start(self):

if self.grounded():

self.jump = constants.JUMP\_TIME

if self.clock % 2 == 0:

self.jump1.play()

else:

self.jump2.play()

def jump\_stop(self):

self.jump = min(self.jump, 1)

def grounded(self):

if not self.blocks == None:

self.rect.y += 1

grounded\_plats = p.sprite.spritecollide(self, self.blocks, False)

self.rect.y -= 1

if len(grounded\_plats) > 0:

return True

else:

return False

else:

return False

def sign(self, num):# For Various math

if num == 0 or num == None:

return 0

return num//abs(num)

#item.py

#Base class for player and block to inherit from p.sprite.Sprite

import pygame as p

class Item(p.sprite.Sprite):

def \_\_init\_\_(self, img=p.image.load('Images\\null.png'), pos=(0,0)):

super().\_\_init\_\_()

self.image = img

self.rect = self.image.get\_rect()

self.rect.x = pos[0]

self.rect.y = pos[1]

#constants.py

#contains constants used for various calculations

TERMINAL\_VELOCITY = 20

SCREEN\_WIDTH = 768

SCREEN\_HEIGHT = 576

JUMP\_SPEED = 8

JUMP\_TIME = 15

TOP\_SPEED = 3 # for walking

WALK\_ACCELERATION = 1

#color.py

#contains functions to return a color rgb tuple by name of color

def toGrayscale(color):

avg = (color[0] + color[1] + color[2]) // 3

return (avg, avg, avg)

RED = (255, 0, 0)

GREEN = (0, 255, 0)

BLUE = (0, 0, 255)

WHITE = (255, 255, 255)

BLACK = (0,0,0)

GREY = (128, 128, 128)

GRAY = (128, 128, 128)

CYAN = (0, 255, 255)

PURPLE = (128, 0, 128)

YELLOW = (255, 255, 0)

ORANGE = (255, 165, 0)

MAROON = (128, 0, 0)

DARK\_RED = (139, 0, 0)

BROWN = (165, 42, 42)

SALMON = (250, 128, 114)

OLIVE = (128, 128, 0)

TURQOUISE = (64, 224, 208)

NAVY = (0, 0, 128)

INDIGO = (75, 0, 130)

PINK = (255, 192, 203)