import arcade  
import os  
  
  
# Constants  
SCREEN\_WIDTH = 1000  
SCREEN\_HEIGHT = 800  
SCREEN\_TITLE = "SUPERPUBER"  
  
# Constants used to scale our sprites from their original size  
CHARACTER\_SCALING = 0.8  
TILE\_SCALING = 0.5  
COIN\_SCALING = 0.5  
key\_SCALING = 0.5  
SPRITE\_PIXEL\_SIZE = 128  
GRID\_PIXEL\_SIZE = (SPRITE\_PIXEL\_SIZE \* TILE\_SCALING)  
  
# Speed of player  
PLAYER\_MOVEMENT\_SPEED = 5  
GRAVITY = 1  
PLAYER\_JUMP\_SPEED = 20  
  
# Pixel Scrolling  
  
LEFT\_VIEWPORT\_MARGIN = 250  
RIGHT\_VIEWPORT\_MARGIN = 250  
BOTTOM\_VIEWPORT\_MARGIN = 50  
TOP\_VIEWPORT\_MARGIN = 100  
  
PLAYER\_START\_X = SPRITE\_PIXEL\_SIZE \* TILE\_SCALING \* 2  
PLAYER\_START\_Y = 400  
  
# Facing constants  
RIGHT\_FACING = 0  
LEFT\_FACING = 1  
  
  
def load\_texture\_pair(filename):  
 return [  
 arcade.load\_texture(filename), arcade.load\_texture(filename, mirrored=True)  
 ]  
  
  
class PlayerCharacter(arcade.Sprite):  
  
 def \_\_init\_\_(self):  
  
 # Set up parent class  
 super().\_\_init\_\_()  
  
 # Default to face-right  
 self.character\_face\_direction = RIGHT\_FACING  
  
 # Used for flipping between image sequences  
 self.cur\_texture = 0  
 self.scale = CHARACTER\_SCALING  
  
 # Track our state  
 self.jumping = False  
 self.climbing = False  
 self.is\_on\_ladder = False  
  
 # Textures  
 main\_path = ":resources:images/animated\_characters/female\_person/femaleperson"  
  
 # Textures for idle standing  
 self.idle\_texture\_pair = load\_texture\_pair(f"{main\_path}\_idle.png")  
 self.jump\_texture\_pair = load\_texture\_pair(f"{main\_path}\_jump.png")  
 self.fall\_texture\_pair = load\_texture\_pair(f"{main\_path}\_fall.png")  
  
 # Load textures for walking  
 self.walk\_textures = []  
 for i in range(8):  
 texture = load\_texture\_pair(f"{main\_path}\_walk{i}.png")  
 self.walk\_textures.append(texture)  
  
 # Load textures for climbing  
 self.climbing\_textures = []  
 texture = arcade.load\_texture(f"{main\_path}\_climb0.png")  
 self.climbing\_textures.append(texture)  
 texture = arcade.load\_texture(f"{main\_path}\_climb1.png")  
 self.climbing\_textures.append(texture)  
  
 # Set the initial texture  
 self.texture = self.idle\_texture\_pair[0]  
 self.set\_hit\_box(self.texture.hit\_box\_points)  
  
 def update\_animation(self, delta\_time: float = 1 / 60):  
  
 # Climbing animation  
 if self.is\_on\_ladder:  
 self.climbing = True  
 if not self.is\_on\_ladder and self.climbing:  
 self.climbing = False  
 if self.climbing and abs(self.change\_y) > 1:  
 self.cur\_texture += 1  
 if self.cur\_texture > 7:  
 self.cur\_texture = 0  
 if self.climbing:  
 self.texture = self.climbing\_textures[self.cur\_texture // 4]  
 return  
  
 # Face left / right  
 if self.change\_x < 0 and self.character\_face\_direction == RIGHT\_FACING:  
 self.character\_face\_direction = LEFT\_FACING  
 elif self.change\_x > 0 and self.character\_face\_direction == LEFT\_FACING:  
 self.character\_face\_direction = RIGHT\_FACING  
  
 # Jumping animation  
 if self.change\_y > 0 and not self.is\_on\_ladder:  
 self.texture = self.jump\_texture\_pair[self.character\_face\_direction]  
 return  
 elif self.change\_y < 0 and not self.is\_on\_ladder:  
 self.texture = self.fall\_texture\_pair[self.character\_face\_direction]  
 return  
  
 # idle animation  
 if self.change\_x == 0:  
 self.texture = self.idle\_texture\_pair[self.character\_face\_direction]  
 return  
  
 # walking animation  
 self.cur\_texture += 1  
 if self.cur\_texture > 7:  
 self.cur\_texture = 0  
 self.texture = self.walk\_textures[self.cur\_texture][self.character\_face\_direction]  
  
  
class MyGame(arcade.Window):  
 *"""  
 Main application class.  
 """* def \_\_init\_\_(self):  
  
 # Call the parent class and set up the window  
 super().\_\_init\_\_(SCREEN\_WIDTH, SCREEN\_HEIGHT, SCREEN\_TITLE, fullscreen=True)  
  
 # Setting the path to start with this program  
 file\_path = os.path.dirname(os.path.abspath(\_\_file\_\_))  
 os.chdir(file\_path)  
 """""  
 width, height = self.get\_size()  
 self.set\_viewport(0, width, 0, height)  
 """""  
  
 # Track the current state of what key is pressed  
 self.left\_pressed = False  
 self.right\_pressed = False  
 self.up\_pressed = False  
 self.down\_pressed = False  
 self.jump\_needs\_reset = False  
 self.game\_over = False  
 self.end\_of\_map = False  
  
 self.coin\_list = None  
 self.wall\_list = None  
 self.dont\_touch\_list = None  
 self.player\_list = None  
 self.background\_list = None  
 self.ladder\_list = None  
 self.enemy\_list = None  
 self.moving\_wall\_list = None  
 self.key\_list = None  
 self.health\_list = None  
  
 # Separate variable that holds the player sprite  
 self.player\_sprite = None  
  
 # Our engine  
 self.physics\_engine = None  
  
 self.view\_bottom = 0  
 self.view\_left = 0  
  
 # Level  
 self.level = 1  
  
 # Our score  
 self.score = 0  
  
 # Our score  
 self.score\_key = 0  
  
 self.end\_of\_map = 0  
  
 # Player Health  
 self.health = 3  
  
 # Load sounds  
 self.collect\_coin\_sound = arcade.load\_sound(":resources:sounds/coin1.wav")  
 self.collect\_key\_sound = arcade.load\_sound(":resources:sounds/coin4.wav")  
 self.collect\_health\_sound = arcade.load\_sound(":resources:sounds/upgrade1.wav")  
 self.jump\_sound = arcade.load\_sound(":resources:sounds/jump1.wav")  
 # self.game\_sound = arcade.load\_sound(":resources:sounds/Super Mario Bros.ogg")  
 self.game\_finish = arcade.load\_sound(":resources:sounds/gameover2.wav")  
 self.game\_danc = arcade.load\_sound(":resources:sounds/dancing funeral.wav")  
  
 def setup(self, level):  
 *""" Set up the game here. Call this function to restart the game. """* # arcade.play\_sound(self.game\_sound) # We need to check again  
  
 self.view\_bottom = 0  
 self.view\_left = 0  
 self.game\_over = False  
  
 # Create the Sprite lists  
 self.player\_list = arcade.SpriteList()  
 self.wall\_list = arcade.SpriteList()  
 self.background\_list = arcade.SpriteList()  
 self.coin\_list = arcade.SpriteList()  
 self.key\_list = arcade.SpriteList()  
 self.health\_list = arcade.SpriteList()  
 self.enemy\_list = arcade.SpriteList()  
 self.moving\_wall\_list = arcade.SpriteList()  
  
 # Set up the player  
 self.player\_sprite = PlayerCharacter()  
 self.player\_sprite.center\_x = PLAYER\_START\_X  
 self.player\_sprite.center\_y = PLAYER\_START\_Y  
 self.player\_list.append(self.player\_sprite)  
 self.score\_key = 0  
  
 # We will add the map here...  
  
 platforms\_layer\_name = 'Platforms'  
 coins\_layer\_name = 'Coins'  
 dont\_touch\_layer\_name = "Don't Touch"  
 moving\_platforms\_layer\_name = 'Moving Platforms'  
 key\_layer\_name = 'key'  
 health\_layer\_name = 'health'  
  
 # Map name  
 map\_name = f":resources:tmx\_maps/ws500\_{level}.tmx"  
  
 # Read in the tiled map  
 my\_map = arcade.tilemap.read\_tmx(map\_name)  
  
 # Calculate the right edge of the my\_map in pixels  
 self.end\_of\_map = my\_map.map\_size.width \* GRID\_PIXEL\_SIZE  
  
 # PLATFORMS  
 self.wall\_list = arcade.tilemap.process\_layer(my\_map, platforms\_layer\_name, TILE\_SCALING)  
  
 # Moving Platforms  
 moving\_platforms\_list = arcade.tilemap.process\_layer(my\_map, moving\_platforms\_layer\_name, TILE\_SCALING)  
 for sprite in moving\_platforms\_list:  
 self.wall\_list.append(sprite)  
  
 # -- Background objects  
 self.background\_list = arcade.tilemap.process\_layer(my\_map, "Background", TILE\_SCALING)  
  
 # Background objects  
 self.ladder\_list = arcade.tilemap.process\_layer(my\_map, "Ladder", TILE\_SCALING)  
  
 # Coins  
 self.coin\_list = arcade.tilemap.process\_layer(my\_map, coins\_layer\_name, TILE\_SCALING)  
 # key  
 self.key\_list = arcade.tilemap.process\_layer(my\_map, key\_layer\_name, TILE\_SCALING)  
 # health  
 self.health\_list = arcade.tilemap.process\_layer(my\_map, health\_layer\_name, TILE\_SCALING)  
  
 # Create the 'physics engine'  
 self.physics\_engine = arcade.PhysicsEnginePlatformer(self.player\_sprite,  
 self.wall\_list,  
 gravity\_constant=GRAVITY,  
 ladders=self.ladder\_list)  
  
 # Don't Touch Layer  
 self.dont\_touch\_list = arcade.tilemap.process\_layer(my\_map, dont\_touch\_layer\_name, TILE\_SCALING)  
  
 if self.level == 1:  
 # Draw a enemy on the platform1 for level 1  
 enemy = arcade.Sprite(":resources:images/enemies/wormGreen.png", CHARACTER\_SCALING / 2)  
  
 enemy.bottom = GRID\_PIXEL\_SIZE \* 17  
  
 # for the position of the enemy  
 enemy.left = GRID\_PIXEL\_SIZE \* 19  
  
 enemy.boundary\_right = GRID\_PIXEL\_SIZE \* 5  
 enemy.boundary\_left = GRID\_PIXEL\_SIZE \* 2  
  
 # Set enemy initial speed  
 enemy.change\_x = 2.5  
 self.enemy\_list.append(enemy)  
  
 # Draw a enemy on the platform2 for level 1  
 enemy = arcade.Sprite(":resources:images/enemies/wormGreen.png", CHARACTER\_SCALING / 2)  
  
 enemy.bottom = GRID\_PIXEL\_SIZE \* 9  
  
 # for the position of the enemy  
 enemy.left = GRID\_PIXEL\_SIZE \* 7  
  
 enemy.boundary\_right = GRID\_PIXEL\_SIZE \* 5  
 enemy.boundary\_left = GRID\_PIXEL\_SIZE \* 2  
  
 # Set enemy initial speed  
 enemy.change\_x = 2.5  
 self.enemy\_list.append(enemy)  
  
 # Draw a enemy on the platform2  
 enemy = arcade.Sprite(":resources:images/enemies/fly.png", CHARACTER\_SCALING / 2)  
  
 enemy.bottom = GRID\_PIXEL\_SIZE \* 3  
  
 # for the position of the enemy  
 enemy.left = GRID\_PIXEL\_SIZE \* 22  
  
 enemy.boundary\_right = GRID\_PIXEL\_SIZE \* 5  
 enemy.boundary\_left = GRID\_PIXEL\_SIZE \* 2  
  
 # Set enemy initial speed  
 enemy.change\_x = 3  
 self.enemy\_list.append(enemy)  
  
 # Draw a enemy on the platform3  
 enemy = arcade.Sprite(":resources:images/enemies/fly.png", CHARACTER\_SCALING / 2)  
  
 enemy.bottom = GRID\_PIXEL\_SIZE \* 3  
  
 # for the position of the enemy  
 enemy.left = GRID\_PIXEL\_SIZE \* 30  
  
 enemy.boundary\_right = GRID\_PIXEL\_SIZE \* 5  
 enemy.boundary\_left = GRID\_PIXEL\_SIZE \* 2  
  
 # Set enemy initial speed  
 enemy.change\_x = 3  
 self.enemy\_list.append(enemy)  
  
 # Draw a enemy on the platform4  
 enemy = arcade.Sprite(":resources:images/enemies/fly.png", CHARACTER\_SCALING / 2)  
  
 enemy.bottom = GRID\_PIXEL\_SIZE \* 3  
  
 # for the position of the enemy  
 enemy.left = GRID\_PIXEL\_SIZE \* 60  
  
 enemy.boundary\_right = GRID\_PIXEL\_SIZE \* 5  
 enemy.boundary\_left = GRID\_PIXEL\_SIZE \* 2  
  
 # Set enemy initial speed  
 enemy.change\_x = 3  
 self.enemy\_list.append(enemy)  
  
 if self.level == 2:  
 # Draw a enemy on the platform1 for level 1  
 enemy = arcade.Sprite(":resources:images/enemies/wormGreen.png", CHARACTER\_SCALING / 2)  
  
 enemy.bottom = GRID\_PIXEL\_SIZE \* 16  
  
 # for the position of the enemy  
 enemy.left = GRID\_PIXEL\_SIZE \* 19  
  
 enemy.boundary\_right = GRID\_PIXEL\_SIZE \* 5  
 enemy.boundary\_left = GRID\_PIXEL\_SIZE \* 2  
  
 # Set enemy initial speed  
 enemy.change\_x = 3  
 self.enemy\_list.append(enemy)  
  
 # Draw a enemy on the platform2 for level 1  
 enemy = arcade.Sprite(":resources:images/enemies/wormGreen.png", CHARACTER\_SCALING / 2)  
  
 enemy.bottom = GRID\_PIXEL\_SIZE \* 23  
  
 # for the position of the enemy  
 enemy.left = GRID\_PIXEL\_SIZE \* 50  
  
 enemy.boundary\_right = GRID\_PIXEL\_SIZE \* 5  
 enemy.boundary\_left = GRID\_PIXEL\_SIZE \* 2  
  
 # Set enemy initial speed  
 enemy.change\_x = 3  
 self.enemy\_list.append(enemy)  
  
 # Draw a enemy on the platform2  
 enemy = arcade.Sprite(":resources:images/enemies/fly.png", CHARACTER\_SCALING / 2)  
  
 enemy.bottom = GRID\_PIXEL\_SIZE \* 15  
  
 # for the position of the enemy  
 enemy.left = GRID\_PIXEL\_SIZE \* 50  
  
 enemy.boundary\_right = GRID\_PIXEL\_SIZE \* 5  
 enemy.boundary\_left = GRID\_PIXEL\_SIZE \* 2  
  
 # Set enemy initial speed  
 enemy.change\_x = 3  
 self.enemy\_list.append(enemy)  
  
 # Draw a enemy on the platform3  
 enemy = arcade.Sprite(":resources:images/enemies/fly.png", CHARACTER\_SCALING / 2)  
  
 enemy.bottom = GRID\_PIXEL\_SIZE \* 40  
  
 # for the position of the enemy  
 enemy.left = GRID\_PIXEL\_SIZE \* 50  
  
 enemy.boundary\_right = GRID\_PIXEL\_SIZE \* 5  
 enemy.boundary\_left = GRID\_PIXEL\_SIZE \* 2  
  
 # Set enemy initial speed  
 enemy.change\_x = 3  
 self.enemy\_list.append(enemy)  
  
 # Draw a enemy on the platform4  
 enemy = arcade.Sprite(":resources:images/enemies/fly.png", CHARACTER\_SCALING / 2)  
  
 enemy.bottom = GRID\_PIXEL\_SIZE \* 29  
  
 # for the position of the enemy  
 enemy.left = GRID\_PIXEL\_SIZE \* 8  
  
 enemy.boundary\_right = GRID\_PIXEL\_SIZE \* 5  
 enemy.boundary\_left = GRID\_PIXEL\_SIZE \* 2  
  
 # Set enemy initial speed  
 enemy.change\_x = 3  
 self.enemy\_list.append(enemy)  
  
 if self.level == 3:  
 # Draw a enemy on the platform1 for level 1  
 enemy = arcade.Sprite(":resources:images/enemies/wormGreen.png", CHARACTER\_SCALING / 2)  
  
 enemy.bottom = GRID\_PIXEL\_SIZE \* 26  
  
 # for the position of the enemy  
 enemy.left = GRID\_PIXEL\_SIZE \* 14  
  
 enemy.boundary\_right = GRID\_PIXEL\_SIZE \* 5  
 enemy.boundary\_left = GRID\_PIXEL\_SIZE \* 2  
  
 # Set enemy initial speed  
 enemy.change\_x = 3  
 self.enemy\_list.append(enemy)  
  
 # Draw a enemy on the platform2 for level 1  
 enemy = arcade.Sprite(":resources:images/enemies/wormGreen.png", CHARACTER\_SCALING / 2)  
  
 enemy.bottom = GRID\_PIXEL\_SIZE \* 41  
  
 # for the position of the enemy  
 enemy.left = GRID\_PIXEL\_SIZE \* 15  
  
 enemy.boundary\_right = GRID\_PIXEL\_SIZE \* 5  
 enemy.boundary\_left = GRID\_PIXEL\_SIZE \* 2  
  
 # Set enemy initial speed  
 enemy.change\_x = 3  
 self.enemy\_list.append(enemy)  
  
 # Draw a enemy on the platform2  
 enemy = arcade.Sprite(":resources:images/enemies/fly.png", CHARACTER\_SCALING / 2)  
  
 enemy.bottom = GRID\_PIXEL\_SIZE \* 15  
  
 # for the position of the enemy  
 enemy.left = GRID\_PIXEL\_SIZE \* 47  
  
 enemy.boundary\_right = GRID\_PIXEL\_SIZE \* 5  
 enemy.boundary\_left = GRID\_PIXEL\_SIZE \* 2  
  
 # Set enemy initial speed  
 enemy.change\_x = 3  
 self.enemy\_list.append(enemy)  
  
 # Draw a enemy on the platform3  
 enemy = arcade.Sprite(":resources:images/enemies/fly.png", CHARACTER\_SCALING / 2)  
  
 enemy.bottom = GRID\_PIXEL\_SIZE \* 15  
  
 # for the position of the enemy  
 enemy.left = GRID\_PIXEL\_SIZE \* 110  
  
 enemy.boundary\_right = GRID\_PIXEL\_SIZE \* 5  
 enemy.boundary\_left = GRID\_PIXEL\_SIZE \* 2  
  
 # Set enemy initial speed  
 enemy.change\_x = 3  
 self.enemy\_list.append(enemy)  
  
 # Draw a enemy on the platform4  
 enemy = arcade.Sprite(":resources:images/enemies/fly.png", CHARACTER\_SCALING / 2)  
  
 enemy.bottom = GRID\_PIXEL\_SIZE \* 28  
  
 # for the position of the enemy  
 enemy.left = GRID\_PIXEL\_SIZE \* 56  
  
 enemy.boundary\_right = GRID\_PIXEL\_SIZE \* 5  
 enemy.boundary\_left = GRID\_PIXEL\_SIZE \* 2  
  
 # Set enemy initial speed  
 enemy.change\_x = 3  
 self.enemy\_list.append(enemy)  
  
 def on\_draw(self):  
 *""" Render the screen. """* # Clear the screen to the background color  
 arcade.start\_render()  
 # Draw our sprites  
 self.wall\_list.draw()  
 self.background\_list.draw()  
 self.ladder\_list.draw()  
 self.coin\_list.draw()  
 self.key\_list.draw()  
 self.player\_list.draw()  
 self.dont\_touch\_list.draw()  
 self.enemy\_list.draw()  
 self.moving\_wall\_list.draw()  
 self.health\_list.draw()  
 if self.game\_over:  
 arcade.draw\_text("Game Over", 330 + self.view\_left, self.view\_bottom + 200, arcade.color.BLACK, 30)  
 arcade.draw\_text("Do You Want To Restart ?", 260 + self.view\_left, self.view\_bottom + 150, arcade.color.  
 BLACK, 30)  
 arcade.set\_background\_color(arcade.csscolor.DARK\_RED)  
  
 # For Showing Score  
 score\_text = f"Score: {self.score}"  
 arcade.draw\_text(score\_text, 10 + self.view\_left, 710 + self.view\_bottom, arcade.csscolor.WHITE, 18)  
  
 # For Showing Score\_key  
 score\_key = f"Key: {self.score\_key}/3"  
 arcade.draw\_text(score\_key, 10 + self.view\_left, 690 + self.view\_bottom, arcade.csscolor.WHITE, 18)  
  
 # For showing Health  
 score\_health = f"Health: {self.health}"  
 arcade.draw\_text(score\_health, 10 + self.view\_left, 750 + self.view\_bottom, arcade.csscolor.WHITE, 18)  
  
 # For showing Level  
 score\_level = f"Level: {self.level}"  
 arcade.draw\_text(score\_level, 10 + self.view\_left, 730 + self.view\_bottom, arcade.csscolor.WHITE, 18)  
 if self.level == 1:  
 arcade.set\_background\_color(arcade.csscolor.CORNFLOWER\_BLUE)  
 if self.level == 2:  
 arcade.set\_background\_color(arcade.csscolor.SNOW)  
 if self.level == 3:  
 arcade.set\_background\_color(arcade.csscolor.LIGHT\_GOLDENROD\_YELLOW)  
  
 """  
 #create a wall for the end of the map ,but its make the game so slowly  
 if self.score\_key != 3:  
 for y in range(0, 500, 1000):  
 wall = arcade.Sprite(":resources:images/tiles/boxCrate\_double.png", TILE\_SCALING)  
 wall.center\_x = 465  
 wall.center\_y = y  
 self.wall\_list.append(wall)  
 """  
 def process\_keychange(self):  
 # Called when we change a key up/down or we move on/off a ladder.  
  
 # process up/down  
 if self.up\_pressed and not self.down\_pressed:  
 if self.physics\_engine.is\_on\_ladder():  
 self.player\_sprite.change\_y = PLAYER\_MOVEMENT\_SPEED  
 elif self.physics\_engine.can\_jump() and not self.jump\_needs\_reset:  
 self.player\_sprite.change\_y = PLAYER\_JUMP\_SPEED  
 self.jump\_needs\_reset = True  
 arcade.play\_sound(self.jump\_sound)  
 elif self.down\_pressed and not self.up\_pressed:  
 if self.physics\_engine.is\_on\_ladder():  
 self.player\_sprite.change\_y = -PLAYER\_MOVEMENT\_SPEED  
  
 # Process up/down when no movement  
 if self.physics\_engine.is\_on\_ladder():  
 if not self.up\_pressed and not self.down\_pressed:  
 self.player\_sprite.change\_y = 0  
 elif self.up\_pressed and self.down\_pressed:  
 self.player\_sprite.change\_y = 0  
  
 # process left/right  
 if self.right\_pressed and not self.left\_pressed:  
 self.player\_sprite.change\_x = PLAYER\_MOVEMENT\_SPEED  
 elif self.left\_pressed and not self.right\_pressed:  
 self.player\_sprite.change\_x = -PLAYER\_MOVEMENT\_SPEED  
 else:  
 self.player\_sprite.change\_x = 0  
  
 def on\_key\_press(self, key, modifiers): # Keyboard functions  
 if key == arcade.key.UP or key == arcade.key.W:  
 self.up\_pressed = True  
 elif key == arcade.key.DOWN or key == arcade.key.S:  
 self.down\_pressed = True  
 elif key == arcade.key.LEFT or key == arcade.key.A:  
 self.left\_pressed = True  
 elif key == arcade.key.RIGHT or key == arcade.key.D:  
 self.right\_pressed = True  
 if key == arcade.key.F:  
 # User hits f. Flip between full and not full screen.  
 self.set\_fullscreen(not self.fullscreen)  
  
 width, height = self.get\_size()  
 self.set\_viewport(0, width, 0, height)  
  
 if key == arcade.key.ESCAPE:  
 # User hits s. Flip between full and not full screen.  
 self.set\_fullscreen(not self.fullscreen)  
  
 self.set\_viewport(0, SCREEN\_WIDTH, 0, SCREEN\_HEIGHT)  
 if key == arcade.key.Y:  
 self.game\_over = False  
 # MyGame()  
 arcade.set\_background\_color(arcade.csscolor.CORNFLOWER\_BLUE)  
  
 self.process\_keychange()  
  
 def on\_key\_release(self, key, modifiers):  
  
 if key == arcade.key.UP or key == arcade.key.W:  
 self.up\_pressed = False  
 self.jump\_needs\_reset = False  
 elif key == arcade.key.DOWN or key == arcade.key.S:  
 self.down\_pressed = False  
 elif key == arcade.key.LEFT or key == arcade.key.A:  
 self.left\_pressed = False  
 elif key == arcade.key.RIGHT or key == arcade.key.D:  
 self.right\_pressed = False  
  
 self.process\_keychange()  
  
 def on\_update(self, delta\_time):  
  
 # We're calling physics engine  
 self.physics\_engine.update()  
  
 # Update animations  
 if self.physics\_engine.can\_jump():  
 self.player\_sprite.can\_jump = False  
 else:  
 self.player\_sprite.can\_jump = True  
  
 if self.physics\_engine.is\_on\_ladder() and not self.physics\_engine.can\_jump():  
 self.player\_sprite.is\_on\_ladder = True  
 self.process\_keychange()  
 else:  
 self.player\_sprite.is\_on\_ladder = False  
 self.process\_keychange()  
 self.enemy\_list.update()  
 self.coin\_list.update\_animation(delta\_time)  
 self.key\_list.update\_animation(delta\_time)  
 self.player\_list.update\_animation(delta\_time)  
 self.health\_list.update\_animation(delta\_time)  
 self.ladder\_list.draw()  
 # Update walls, used with moving platforms  
 self.wall\_list.update()  
  
 # Update the player based on the physics engine  
 if not self.game\_over:  
 # Move the enemies  
 self.enemy\_list.update()  
  
 # Check each enemy  
 for enemy in self.enemy\_list:  
 # If the enemy hit a wall, reverse  
 if len(arcade.check\_for\_collision\_with\_list(enemy, self.wall\_list)) > 0:  
 enemy.change\_x \*= -1  
  
 # See if the wall hit a boundary and needs to reverse direction.  
 for wall in self.wall\_list:  
  
 if wall.boundary\_right and wall.right > wall.boundary\_right and wall.change\_x > 0:  
 wall.change\_x \*= -1  
 if wall.boundary\_left and wall.left < wall.boundary\_left and wall.change\_x < 0:  
 wall.change\_x \*= -1  
 if wall.boundary\_top and wall.top > wall.boundary\_top and wall.change\_y > 0:  
 wall.change\_y \*= -1  
 if wall.boundary\_bottom and wall.bottom < wall.boundary\_bottom and wall.change\_y < 0:  
 wall.change\_y \*= -1  
  
 # if you hit any coins  
 coin\_hit\_list = arcade.check\_for\_collision\_with\_list(self.player\_sprite, self.coin\_list)  
  
 for coin in coin\_hit\_list:  
 self.score += 1  
 # Remove the coin  
 coin.remove\_from\_sprite\_lists()  
 # Play sound  
 arcade.play\_sound(self.collect\_coin\_sound)  
  
 # if you hit any key  
 key\_hit\_list = arcade.check\_for\_collision\_with\_list(self.player\_sprite, self.key\_list)  
  
 for key in key\_hit\_list:  
 self.score\_key += 1  
 # Remove the key  
 key.remove\_from\_sprite\_lists()  
 # Play sound  
 arcade.play\_sound(self.collect\_key\_sound)  
  
 # if you hit any health  
 health\_hit\_list = arcade.check\_for\_collision\_with\_list(self.player\_sprite, self.health\_list)  
  
 for health in health\_hit\_list:  
 self.health += 1  
 # Remove the health  
 health.remove\_from\_sprite\_lists()  
 # Play sound  
 arcade.play\_sound(self.collect\_health\_sound)  
 changed\_viewport = False  
  
 # if player falls  
 if self.player\_sprite.center\_y < -100:  
 self.player\_sprite.center\_x = PLAYER\_START\_X  
 self.player\_sprite.center\_y = PLAYER\_START\_Y  
  
 # Set the camera  
 self.view\_left = 0  
 self.view\_bottom = 0  
 changed\_viewport = True  
 self.health -= 1  
 arcade.play\_sound(self.game\_finish)  
  
 # Did the player touch something they should not?anything  
 if arcade.check\_for\_collision\_with\_list(self.player\_sprite, self.dont\_touch\_list):  
 self.player\_sprite.change\_x = 0  
 self.player\_sprite.change\_y = 0  
 self.player\_sprite.center\_x = PLAYER\_START\_X  
 self.player\_sprite.center\_y = PLAYER\_START\_Y  
 self.health -= 1  
 arcade.play\_sound(self.game\_finish)  
 # Did the player touch something they should not?anything  
 if arcade.check\_for\_collision\_with\_list(self.player\_sprite, self.enemy\_list):  
 self.player\_sprite.change\_x = 0  
 self.player\_sprite.change\_y = 0  
 self.player\_sprite.center\_x = PLAYER\_START\_X  
 self.player\_sprite.center\_y = PLAYER\_START\_Y  
 self.health -= 1  
 arcade.play\_sound(self.game\_finish)  
  
 # See if the user got to the end of the level  
 """  
 if self.player\_sprite.center\_x >= self.end\_of\_map:  
 # Advance to the next level  
 self.level += 1  
 # Load the next level  
 self.setup(self.level)  
  
 # Set the camera to the start  
 self.view\_left = 0  
 self.view\_bottom = 0  
 changed\_viewport = True  
 """  
  
 if self.health == 0:  
 self.game\_over = True  
  
 if self.score == 50:  
 self.health += 1  
 self.score = 0  
  
 if self.score\_key == 3:  
 if self.player\_sprite.center\_x >= self.end\_of\_map:  
 self.level += 1  
 # Load the next level  
 self.setup(self.level)  
 self.view\_left = 0  
 self.view\_bottom = 0  
 changed\_viewport = True  
  
 # Manage Scrolling  
  
 # Scroll left  
 left\_boundary = self.view\_left + LEFT\_VIEWPORT\_MARGIN  
 if self.player\_sprite.left < left\_boundary:  
 self.view\_left -= left\_boundary - self.player\_sprite.left  
 changed\_viewport = True  
  
 # Scroll right  
 right\_boundary = self.view\_left + SCREEN\_WIDTH - RIGHT\_VIEWPORT\_MARGIN  
 if self.player\_sprite.right > right\_boundary:  
 self.view\_left += self.player\_sprite.right - right\_boundary  
 changed\_viewport = True  
  
 # Scroll up  
 top\_boundary = self.view\_bottom + SCREEN\_HEIGHT - TOP\_VIEWPORT\_MARGIN  
 if self.player\_sprite.top > top\_boundary:  
 self.view\_bottom += self.player\_sprite.top - top\_boundary  
 changed\_viewport = True  
  
 # Scroll down  
 bottom\_boundary = self.view\_bottom + BOTTOM\_VIEWPORT\_MARGIN  
 if self.player\_sprite.bottom < bottom\_boundary:  
 self.view\_bottom -= bottom\_boundary - self.player\_sprite.bottom  
 changed\_viewport = True  
  
 if changed\_viewport:  
 self.view\_bottom = int(self.view\_bottom)  
 self.view\_left = int(self.view\_left)  
  
 # Done the Scrolling  
  
 arcade.set\_viewport(self.view\_left, SCREEN\_WIDTH + self.view\_left, self.view\_bottom,  
 SCREEN\_HEIGHT + self.view\_bottom)  
  
  
def main():  
 *""" Main method """* window = MyGame()  
 window.setup(window.level)  
 arcade.run()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()