Super JOANA

Özgür Aydemir
ozgur.aydemir@unil.ch

Allan Milevaz
allan.milevaz@unil.ch

Giacomo Rousso
giacomo.rousseau@unil.ch

***Abstract*—Super JOANA is a video game derived version of the old and legendary video game of Super Mario that everyone played once in a life, especially the older generation. This game is constructed in a way that everyone(especially children) will understand easily the game and, of course, have fun with it. The aim of this project is to provide useful tool to children during the hard quarantine days due to Covid-19 and help them to keep their moral health up.**

I. Introduction

The history of video games began in the 1950 and 1960s, as computer scientists began designing simple games and simulations on mainframe computers. The first video game was created in October 1958 by a Physicist William Higinbotham. It was a very simple tennis game, similar to the classic 1970s video game Pong. As human progress is growing exponentially, the legendary game Super Mario is invented in 1985 by the Japanese electronic game manufacturer Nintendo Company Ltd. It is a very basic game constituted with several levels and allows the player a large simple ability such as jumping , running ... and at the end, it had a huge success, appreciated for the majority of population. In 2021, we are far more developed in term of creating video games and also customers become more addicted to video games and adapted this as a life style.

Although, in March 2020, an unexpected event happened and everyone had to stay in the quarantine for a very long time. The quarantine period was certainly very hard times for everyone due to Covid-19 started by March 2020. Not only countries economies and public policies have been affected in a very severe and unfavourable way, also, people's mental health and psychology were damaged. Although, adults, by their higher experience and maturity, were quickly able to adapt the changing conditions (even-though, it was not the case for everyone) but for younger generation, everyone should accept that it was a challenge.

For this project , we have been inspired by a research paper called ”Playing Video Games During the COVID-19 Pandemic and Effects on Players' Well-Being” written by Barr and Stewart which states that playing video games has had a positive effect on players' perceived well-being during Covid-19 pandemic. Games have provided an enjoyable means of maintaining social contact , stress relieving and mentally stimulating escape from the effects of lockdown.

II. Description of the research question and Relevant Literature

A. *Research Question*

Can we create our ”own” video game with our ”own” sprite and our ”own” maps by using Python ?

As explained above, we have been inspired by the paper about the effect of playing video game on players ”Well-Being”. We suddenly become so ambitious and exhausted to create our ”own video game !”

B. *Relevant Literature*

We have used three tools that helps us to create our video game. First one is the Arcade library, secondly, we have used Piskel (https://www.piskelapp.com/). Finally, Tile Map editor.

1) **Arcade Library**: Basically, the skeleton of our code relies on the Arcade library. This library contains a lot of useful information concerning how to create a video game from A to Z and provides a lot tutorial that we have been trained during the process of working with the code. We look for some other options instead of Arcade library such as Pygame but due to its lack of tutorials and also his lower performance in term of animated sprite performance, we preferred to go with the Arcade Library.
2) **Piskel**: This is the platform that enabled us to create our sprite such as tramboline.. and implement it into our game (https://www.piskelapp.com/).
3) **Tile Map Editor**: This is the platform where we created levels. So basically, we have implemented four levels including level 3 and level 6 with different level of difficulties. Basically, level 1 is the easiest level and it goes harder and harder as the level passed.

III. The methodology applied to address the research question and Algorithm

In this section, we will discuss how we used Arcade library to create our video game and what was our contribution to the game beside creating sprites and maps. Before we start, our character called JOANA is :

Before we start coding our game, We have defined some fixed characteristics concerning the screen sizes and camera view , certain characteristic of the player such as his movement speed, jump speed and his coordinate to start a game.

Screen
SCREEN_WIDTH = 1000
SCREEN_HEIGHT = 650
SCREEN_TITLE = ”Super JOANA”

Size of our sprites
CHARACTER_SCALING = 1
TILE_SCALING = 0.5
COIN_SCALING = 0.5
SPRITE_LASER_SCALING = 0.8
SPRITE_PIXEL_SIZE = 128
GRID_PIXEL_SIZE = (SPRITE_PIXEL_SIZE*TILE_SCALING
SPRITE_SIZE = int(SPRITE_PIXEL_SIZE*TILE_SCALING)

How many pixels to keep with each side of the camera
LEFT_VIEWPORT_MARGIN = 250
RIGHT_VIEWPORT_MARGIN = 250
BOTTOM_VIEWPORT_MARGIN = 250
TOP_VIEWPORT_MARGIN = 100

Player
PLAYER_MOVEMENT_SPEED = 3 horizontal speed
PLAYER_JUMP_SPEED = 13 vertical speed
PLAYER_START_X = 250 starting position (x coordinate)
PLAYER_START_Y = 2700 starting position (y coordinate)
UPDATES_PER_FRAME = 4 speed of the animation
RIGHT_FACING = 0 looks to the right at the start
LEFT_FACING = 1 looks to the left when walking to the left

Other
GRAVITY = 0.8
BULLET_SPEED = 10
EXPLOSION_TEXTURE_COUNT = 60

Once we fixed the constant variables , We have created three main classes that each corresponds to particular set of characteristics that we are using in our game. First class is called “PlayerCharacter()” where we start by defining the parameters related to our character specificity. It allows to visualize the walking animation of the sprite.Therefore, we are adding her some walking animation meaning that every time we press the button left or right, she is going to walk by using her legs which constitutes a part the animation. After that, we have a second function“def update_animation()” which allow us to visualize the animation once we are in the

game. Here, we are providing the complete code:

```
# Create a class for the player
# ==================================================================================
class PlayerCharacter(arcade.Sprite):
    def _init_(self):
        super()._init_()
        # Will face right when = 0 and left when = 1
        self.character_face_direction = RIGHT_FACING
        # flipping between images
        self.cur_texture = 0
        self.scale = CHARACTER_SCALING
        # Adjust the collision box (the four corners)
        self.points = [[-22, -64], [22, -64], [22, 28], [-22, 28]]
        # Load the player's sprite
        main_path =
        ":resources:images/animated_characters/female_adventurer/femaleAdventurer"
        # load texture for idle standing
        self.idle_texture_pair = load_texture_pair(f"{main_path}_idle.png")
        # Walking
        self.walk_textures = []
        for i in range(8):
            # load all the frame to create the animation when walking
            texture = load_texture_pair(f"{main_path}_walk{i}.png")
            self.walk_textures.append(texture)
    def update_animation(self, delta_time: float = 1/60):
        # figure out if we need to flip face left or 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
        # idle animation
        if self.change_x == 0 and self.change_y == 0:
            self.texture = self.idle_texture_pair[self.character_face_direction]
            return
        # Walking animation
        self.cur_texture += 1
        if self.cur_texture > 7 * UPDATES_PER_FRAME:
            self.cur_texture = 0
        frame = self.cur_texture // UPDATES_PER_FRAME
        direction = self.character_face_direction
        self.texture = self.walk_textures[frame][direction]
# ==================================================================================
```

As follows , we have another class called Explosion which is used to describe another animation in our game where , each time player shoots to kill the enemies by using the Mouse and every time the bullet reach the enemy, it will generate an explosion. However, we observe only the animation part here. More explicit detail concerning how the explosion works will be elaborated in the further section.

```
class Explosion(arcade.Sprite):

    def _init_(self, texture_list):
        # Start at the first frame
        super()._init_()
        self.current_texture = 0
        self.textures = texture_list

    def update(self):
        # Move to the next frame until the animation is over
        self.current_texture += 1
        if self.current_texture < len(self.textures):
            self.set_texture(self.current_texture)
        else:
            self.remove_from_sprite_lists()
```

Finally, the last class **”MyGame(arcade.Window)”** constitutes the main body of our code and also the skeleton of our game. In the first step, we are placing the screen width, height and title into **”super().__init__()”** which defines our size of the screen as the class arcade.Window compose these parameters concerning sizes of our windows . Afterwards, we are defining every single objects, items, players, enemies, explosions and sounds (game over sounds, gun sounds ..) that we want see in the game.

”def_setup(self,level)” In the next step, we are defining a function setup inside the main class that takes the “level” as a parameter. This parameter ”level” is very important if our map contains many levels (which is the case), without defining it, we can not pass from one level to another. In addition, inside this function, we are defining every single parameters as a **”arcade.SpriteList()”**. This function is useful in many ways because it saves lots of code writing from us. If we want to create several enemies or several ladders or any useful items for our game, instead of defining one by one and updating them, we can use SpriteList() function to update once for all.

In the next stage, we are defining the layers name used in our map that we named it. The function **”arcade.tilemap.read tmx()”** allows to read our map composed of several levels with the function of **”arcade.tilemap.process_layer”** we are attributing ,in some sort, to our SpriteList() all the items that we created for the game .In the map , we already created layers such as “Coins” , “Enemies” and “Ladder”. We have used a very important function here which is **“arcade.PhysicsEnginePlateformer()”**, what it does is to add the physical properties to the layers to ensure us not to cross the Wall ahead or to go through into the ladder without touching it.

The third function is **“def on draw(self):”** where we draw all the parameters that we have (coins, bullets, enemies . . .) by using simply draw function such that “self.parameter name.draw()”. It will draw and display in the screen of the game. Although, the function arcade.start_render() has to be used before everything else.

The following function is **”def on_mouse_press”** “” which is called whenever the Mouse button is clicked. We wanted that our player shoot bullets when facing the enemies. The enemies considered are bees with different colors. So , we first created the bullet and display the gun sound every time we click the Mouse. In the next step, we give the initial coordinates of the bullets which makes it go from the player. So, the initial coordinates are equal to the center coordinates of the player as it will go from the player. Secondly , we had to tell the bullets that if it touches nothing, it will just move away from the screen. As we have a scrolling screen, it was quite challenging to adapt the bullet orientation and make it understand last point. So, We have added into “dest_x” function a self.view_left for the movement in x-axis and self.view_bottom for the y-axis.

The next one is **”def on_key_press”**, it has a very simple function, it allow us to use our clavier to guide our player by making him to the right when we press the right key or to D and if we want him go climb the ladder or jump, we press either UP or W or SPACE. If we release the button, we want to player not to move and wait for the upcoming directive from us. This is controlled by using the “def on key release” function which is very easy to understand the code and how it is working.

Finally the last function and the most important function in this class is called **”def on_update”**. All the functions explained before was used mostly defining and creating items in the game but now we will answer the question “Who is doing what ? ” we are basically telling to every items what they are supposed to do when they are called. Our task is to implement and attribute to every single layers the appropriate characteristic while playing the game. For example, a layer “Ladder” should carry the speciality such that the player can climb on this or a coins layer “Coins” could be collected in case of a physical interaction with the player and the score should be increased by the quantity that equals to the amount of coins collected and so on. As a first step, we wanted that enemies can shoot us with the bullet as well as we can shoot them, so we calculated the distance between the enemies that are already implemented in the map corresponding to the layer “Enemies”. This distance is found as a difference between enemies current location and the central position of the player (the x and y coordinates). As we have a scrolling screen, we had to allow the enemies to shoot us only if they see the player and we see them. So with a basic if statement, we found a threshold of the corresponding distance x and distance y such as the distance between enemy and the player should be less that 600 for the x coordinate and less than 450 for the y coordinate. If this condition is satisfied, then it is allowed to shoot us. Another useful function corresponding to arcade library, that we use so much of, it is **”arcade.check_for_collision_with_list() ”** , it takes two argument in the form of arcade.Spritelist(), so it basically means the interaction of two items. We used it to deal with the interaction of the coins and our player to see that if our player touches the coins , the coins will be removed from the game (**coin.remove_from_sprite_lists()**) and a score will be updated bu the number of the coins collected.

A very similar logic was employed for the bullet such as if there is an interaction between the bullet and Wall, the bullet will be removed by adding the gun sound. Our main reason was to prevent the bullets crossing from one Wall to another.

We wanted to improve more the game so we applied this method in several places and several times. The following scenario has been created such that if the bullet, going from the player, touches the coins , a score will be updated and the respective coin will be removed by displaying a coin sound. In case of the interaction between bullet coming from the player to the enemy, the explosion will appear in the coordinate of where the enemy is located and a gun sound will appear by removing the enemy. Again, the scrolling screen properties plays an important role when we want to get rid off bullets and as always , we have to include self.view bottom and self.view left in the right places.

In the next steps , we have to manage the scrolling in a sense that when some specific conditions met, we set the camera view in the beginning coordinates where the player starts the game. For example if player fall down , which is equivalent of y coordinates lower than -100, we set the camera to the beginning coordinates and play a sound of game over. Game over also happens in case of the interaction with the player and Don’t touch list where we defined in our map. Basically, it corresponds to the lava or water .. something that the player must not touch. When the player is touched by the bullet coming from the enemy, he is also dead and he restart the game with the initial coordinate.

At the end, we are running all the code by the command **”def main()”** where it contains arcade.run() function and also setup() function. Without them, we can not run our game.

IV. A DISCUSSION OF THE IMPLEMENTATION OF THE ALGORITHM

One important aspect that we must certainly highlight is that we developed our own code and implemented to the game as an other challenge for the game player to render the game more difficult but also funnier. One new implementation of the rule is to reward the player such as if he/she manage to score 10 coins, automatically, one life will be given to the gamer. Second rule is much more against the player, if by any chance, JOANA dies in the game , the gamer will lose one life, which is quite normal, but also lose three points acquired by coins. Now we go the most important rule of the game and also the one that will make all the players buy this game to try it. This new rule is being discussed several times among us and also all the levels have been re-created just to respect this new rule. JOANA has five lifes in total that is attributed to her automatically every time we run the game. If the player managed to pass several levels and JOANA lose all her lifes in the middle of the game level. She will restart again from the beginning. In order to make attractive this new rule, we have coded level 4,5 and 6 quite difficult so that the player will keep trying again and again until he/she will manage to finish all the level with success..

V. RESULTS

At the end, we managed to create our first video game where we have implemented everything that we want. Our sprite ”JOANA” is now able to do every commands that we are asking her to do, First of all, she can move with animation with her legs. Secondly, she can jump and climb the ladders. Thirdly, facing with the enemies, she can shoot them with the bullets to destroy them. We also added some functionality to the enemies such that they can shoot the JOANA once a specific distance is respected between the two. The enemies can adapt JOANA’s angle and rotate towards where she is located. Importantly, a result that we enjoyed so much to succed is to be able to add our own sprites and our own maps into the game platform that we have created ourselfs.

But a video game is endless, and it is only limited to our imagination. There is so much novelty and possibility to implement new characteristic to our game through maps, players, items or different enemies.

VI. CONCLUSION

As a conclusion , we are very satisfied and proud of our first video game which was quite challenging and also funny to work on. We first had to learn how to use Arcade library for creating a video game and master Tile Map Editor to create our own maps (different levels for our game) in the form of ”.tmx” and Piskel to create our own sprites. Our character is able to do shooting bullets to kill enemies, walking with animation, jumping and climbing the ladder in order to pass every level. At the bottom left, we have a score measure and coins score which provides us the amount of time spent to finish one level and the amount of gold collected for each level.

This game constitutes a first step of what we aim to do in the future and we are very happy and proud to provide this ambitious project ”Super JOANA” and we hope you will enjoy playing the game as much as we enjoyed coding it.

VII. REFERENCES

https://www.aps.org/publications/apsnews/200810/physicshistory.cfm4
https://en.wikipedia.org/wiki/2021_in_video_games

VIII. APPENDIX

```
class MyGame(arcade.Window):
    """
    Main application class.
    """

    # Init
    # =================
    def _init_(self):
        # Call the parent class and set up the window
        super()._init_(SCREEN_WIDTH, SCREEN_HEIGHT, SCREEN_TITLE)

        # These are 'lists' that keep track of our sprites
        self.coin_list = None
        self.wall_list = None
        self.player_list = None
        self.enemy_list = None
        self.bullet_list = None
        self.bullet_enemy_list = None
        self.explosion_list = None
        self.ladder_list = None
        self.background_list = None
        self.foreground_list = None
        self.dont_touch_list = None
        self.trampoline_list = None

        # Enemies that shoot me
        self.frame_count = 0

        # Remove the comment if you don't want to see the mouse cursor
        #self.set_mouse_visible(False)

        # Pre-load the animation frames.
        self.explosion_texture_list = []
        columns = 16
        count = 60
        sprite_width = 256
        sprite_height = 256
        file_name = f":resources:images/spritesheets/explosion.png"
        # Load the explosion from a sprite sheet
        self.explosion_texture_list =arcade.load_spritesheet
        (file_name,sprite_width, sprite_height, columns, count)

        # Keep track of the score and life
        self.score = 0
        self.life = 5

        # Separate variable that holds the player sprite
        self.player_sprite = None

        # Our physics engine
        self.physics_engine = None

        # Used to keep track of our scrolling
        self.view_bottom = 0
        self.view_left = 0

        # right edge of the map
        self.end_of_map = 0

        # Starting level when loading the game
        self.level = 3

        # Load sounds
        self.collect_coin_sound = arcade.load_sound (":resources:sounds/coin1
        self.jump_sound = arcade.load_sound(":resources:sounds/jump1.wav")
        self.game_over = arcade.load_sound(":resources:sounds/gameover1.wav")
        self.gun_sound = arcade.sound.load_sound(":resources:sounds/laser1.wa
        self.hit_sound = arcade.sound.load_sound(":resources:sounds/explosion

        # Put a blue background
        arcade.set_background_color(arcade.csscolor.CORNFLOWER_BLUE)

        # Add timer
        self.total_time = 0.0
    # =======================
```

```
        # Map name
        map_name = f"map2_level_{self.level}.tmx"

        # read in the tiled map
        my_map = arcade.tilemap.read_tmx(str(map_name))

        # calculate the right edge of my_map
        self.end_of_map = my_map.map_size.width * GRID_PIXEL_SIZE

        # Link the layer from tiled with our game
        self.background_list = arcade.tilemap.process_layer
               (my_map,background_layer_name, TILE_SCALING)
        self.foreground_list = arcade.tilemap.process_layer
               (my_map, foreground_layer_name, TILE_SCALING)
        self.coin_list = arcade.tilemap.process_layer
               (my_map, coins_layer_name, TILE_SCALING, use_spatial_hash=True)
        self.dont_touch_list = arcade.tilemap.process_layer
               (my_map, dont_touch_layer_name, TILE_SCALING, use_spatial_hash=True)
        self.trampoline_list = arcade.tilemap.process_layer
               (my_map, trampoline_layer_name, TILE_SCALING, use_spatial_hash=True)
        self.enemy_list = arcade.tilemap.process_layer
               (my_map, enemy_layer_name, TILE_SCALING)
        self.ladder_list = arcade.tilemap.process_layer
               (my_map, ladder_layer_name, TILE_SCALING, use_spatial_hash=True)
        self.wall_list = arcade.tilemap.process_layer(map_object=my_map,
                                                      layer_name=platform_layer_name,
                                                      scaling=TILE_SCALING,
                                                      use_spatial_hash=True)


        # Set the background color
        if my_map.background_color:
            arcade.set_background_color(my_map.background_color)

        # Create "physic engine"
        self.physics_engine = arcade.PhysicsEnginePlatformer
              (self.player_sprite, self.wall_list, GRAVITY, self.ladder_list)

        # Timers
        self.total_time = 0.0
    # =======================
```

```
# Setup the game
    # ====================
    def setup(self, level):
        """ Set up the game here. Call this function to restart the game. """

        # Used to keep track of our scrolling
        self.view_bottom = 0
        self.view_left = 0

        # Reset the initial value at beginning of every new level
        self.score = 0
        self.life = 5

        # Create the Sprite lists
        self.player_list = arcade.SpriteList()
        self.trampoline_list = arcade.SpriteList()
        self.wall_list = arcade.SpriteList()
        self.enemy_list = arcade.SpriteList()
        self.coin_list = arcade.SpriteList()
        self.foreground_list = arcade.SpriteList()
        self.background_list = arcade.SpriteList()
        self.bullet_list = arcade.SpriteList()
        self.explosion_list = arcade.SpriteList()
        self.bullet_enemy_list = arcade.SpriteList()
        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)

        # Name of the layer from tiled
        platform_layer_name = "ground"
        coins_layer_name = "Coins" # items you can collect
        foreground_layer_name = "Foreground"
        background_layer_name = "Background"
        dont_touch_layer_name = "Don't touch" # will kill you if you touch th
        trampoline_layer_name = "Trampoline" # will make you jump higher
        enemy_layer_name = "Enemies"
        ladder_layer_name = "Ladder"
```

```
# Draw sprites and information
    # ================
    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.foreground_list.draw()
        self.dont_touch_list.draw()
        self.trampoline_list.draw()
        self.player_list.draw()
        self.enemy_list.draw()
        self.ladder_list.draw()
        self.coin_list.draw()
        self.explosion_list.draw()
        self.bullet_list.draw()
        self.bullet_enemy_list.draw()

        # Calculates minutes and seconds
        minutes = int(self.total_time) // 60
        seconds = int(self.total_time) % 60

        # Draw Timer, Score and Life on the screen
        #(don't put it before "draw our sprites")
        time_text = f"Time: {minutes:02d}:{seconds:02d}"
        arcade.draw_text(time_text, 10 + self.view_left,
                         30 + self.view_bottom, arcade.color.BLACK, 20)

        score_text = f"Score: {self.score}"
        arcade.draw_text(score_text, 10 + self.view_left,
                         10 + self.view_bottom, arcade.csscolor.WHITE, 18)

        life_text = f"Life: {self.life}"
        arcade.draw_text(life_text, 10 + self.view_left,
                         600 + self.view_bottom, arcade.csscolor.WHITE, 18)
    # ================================================
# When we use the mouse
    # ================================================
    def on_mouse_press(self, x, y, button, modifiers):
        """ Called whenever the mouse button is clicked
        """

        # create a bullet with sound
        bullet = arcade.Sprite(":resources:images/space_shooter/laserBlue01.p
                               SPRITE_LASER_SCALING)
        arcade.sound.play_sound(self.gun_sound)

        # POSITION THE BULLET
        start_x = self.player_sprite.center_x
        start_y = self.player_sprite.center_y
        bullet.center_x = start_x
        bullet.center_y = start_y

        # mouse destination
        dest_x = x + self.view_left
        dest_y = y + self.view_bottom

        # Compute the correct angle
        x_diff = dest_x - start_x
        y_diff = dest_y - start_y
        angle = math.atan2(y_diff, x_diff)

        # Angle the bullet sprite
        bullet.angle = math.degrees(angle)
        print(f"Bullet angle: {bullet.angle: .2f}")

        bullet.change_x = math.cos(angle)*BULLET_SPEED
        bullet.change_y = math.sin(angle)*BULLET_SPEED

        # Add the bullet to the appropriate list
        self.bullet_list.append(bullet)
    # =====================================
```

```
def on_key_press(self, key, modifiers):
        """Called whenever a key is pressed. """

        # Add jump and climb up the ladder
        if key == arcade.key.UP or key == arcade.key.W or key == arcade.key.SPACE:
            if self.physics_engine.is_on_ladder():
                self.player_sprite.change_y = PLAYER_MOVEMENT_SPEED
            elif self.physics_engine.can_jump():
                self.player_sprite.change_y = PLAYER_JUMP_SPEED
                arcade.play_sound(self.jump_sound)
        # Climb dow the ladder
        elif key == arcade.key.DOWN or key == arcade.key.S:
            if self.physics_engine.is_on_ladder():
                self.player_sprite.change_y = -PLAYER_MOVEMENT_SPEED
        # Add horizontal movement
        elif key == arcade.key.LEFT or key == arcade.key.A:
            self.player_sprite.change_x = -PLAYER_MOVEMENT_SPEED
        elif key == arcade.key.RIGHT or key == arcade.key.D:
            self.player_sprite.change_x = PLAYER_MOVEMENT_SPEED
    # =========================================================
```

```
# When we release the keyboard
    # =========================================================
    def on_key_release(self, key, modifiers):
        """Called when the user releases a key. """
        if key == arcade.key.LEFT or key == arcade.key.A:
            self.player_sprite.change_x = 0
        elif key == arcade.key.RIGHT or key == arcade.key.D:
            self.player_sprite.change_x = 0
        elif key == arcade.key.UP or key == arcade.key.W:
            if self.physics_engine.is_on_ladder():
                self.player_sprite.change_y = 0
        elif key == arcade.key.DOWN or key == arcade.key.S:
            if self.physics_engine.is_on_ladder():
                self.player_sprite.change_y = 0
    # =========================================
```

```
def on_update(self, delta_time):
        """ Movement and game logic """

        # update time
        self.total_time += delta_time

        # Move the player with the physics engine
        self.physics_engine.update()
        self.explosion_list.update()
        self.frame_count +=1

        for enemy in self.enemy_list:
            # Position to start at the enemy current location
            start_xx = enemy.center_x
            start_yy = enemy.center_y

            # Get the destination location for the bullet
            dest_xx = self.player_sprite.center_x
            dest_yy = self.player_sprite.center_y

            # Math to calculate how to get bullet to the destination
            xx_diff = dest_xx - start_xx
            yy_diff = dest_yy - start_yy
            angle = math.atan2(yy_diff, xx_diff)

            # Set the enemy to the face of the player
            enemy.angle = math.degrees(angle) - 180

            # Shoot every 90 frames changes and only if close to the player
            if abs(xx_diff) < 600 and abs(yy_diff) < 450:
                if self.frame_count % 90 == 0:
                    bullet_enemy = arcade.Sprite(
                        ":resources:images/space_shooter/laserBlue01.png")
                    bullet_enemy.center_x = start_xx
                    bullet_enemy.center_y = start_yy

                    # Angle the bullet sprite
                    bullet_enemy.angle = math.degrees(angle)

                    # Take into account the angle, calculate our change_x and y
                    bullet_enemy.change_x = math.cos(angle) * BULLET_SPEED
                    bullet_enemy.change_y = math.sin(angle) * BULLET_SPEED

                    self.bullet_enemy_list.append(bullet_enemy)
```

```
# See if we reach a coin
coin_hit_list = arcade.check_for_collision_with_list(self.player_sprite,
                                                     self.coin_list)

# Loop through each coin we hit (if any) and remove it
for coin in coin_hit_list:
    coin.remove_from_sprite_lists()
    arcade.play_sound(self.collect_coin_sound)
    # Increase the score
    self.score += 1

# Bullet
self.bullet_list.update()

for bullet in self.bullet_enemy_list:
    hit_wall_enemy = arcade.check_for_collision_with_list(bullet,
                                                          self.wall_list)
    if len(hit_wall_enemy) > 0:
        arcade.play_sound(self.gun_sound)
        bullet.remove_from_sprite_lists()

for bullet in self.bullet_list:
    hit_list = arcade.check_for_collision_with_list(bullet, self.coin_list)
    hit_enemy = arcade.check_for_collision_with_list(bullet, self.enemy_list)
    #hit_trampoline = arcade.check_for_collision_with_list(bullet,
    #                                               self.trampoline_list)
    hit_wall = arcade.check_for_collision_with_list(bullet, self.wall_list)

    # Collision with the wall for player bullet
    if len(hit_wall) > 0:
        arcade.play_sound(self.gun_sound)
        bullet.remove_from_sprite_lists()

    if len(hit_list) > 0:
        arcade.play_sound(self.gun_sound)
        bullet.remove_from_sprite_lists()

    for coin in hit_list:
        coin.remove_from_sprite_lists()
        arcade.play_sound(self.collect_coin_sound)
        self.score += 1
    if len(hit_enemy) > 0:
        # Make an explosion when the enemy is destroyed
        explosion = Explosion(self.explosion_texture_list)
        # Move it to the location of the enemy
        explosion.center_x = hit_enemy[0].center_x
        explosion.center_y = hit_enemy[0].center_y
        # Call update
        explosion.update()
        self.explosion_list.append(explosion)

        arcade.play_sound(self.gun_sound)
        bullet.remove_from_sprite_lists()

    for enemy in hit_enemy:
        arcade.sound.play_sound(self.hit_sound)
        enemy.remove_from_sprite_lists()

    # if the bullet flies off screen, remove it
    if bullet.bottom > self.width+self.view_bottom or bullet.top <0
       or bullet.right <0 or bullet.left > self.width+self.view_left:
        bullet.remove_from_sprite_lists()

# Manage Scrolling
# Track if we need to change the viewport
changed = False

# Kill the player if he falls out of the map
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 to the start
    self.view_left = 0
    self.view_bottom = 0
    changed = True
    arcade.play_sound(self.game_over)
```

```
# Kill the player if he is touched by an enemy bullet
self.bullet_enemy_list.update()
for bullet_enemy in self.bullet_enemy_list:

    hit_enemy_player = arcade.check_for_collision_with_list(bullet_enemy,
                                                            self.player_list)

    if len(hit_enemy_player) > 0:
        arcade.play_sound(self.gun_sound)
        bullet_enemy.remove_from_sprite_lists()

        if bullet_enemy.bottom > self.width + self.view_bottom or
           bullet_enemy.top < 0 or bullet_enemy.right < 0 or
           bullet_enemy.left > self.width + self.view_left:
            bullet_enemy.remove_from_sprite_lists()

        self.player_sprite.center_x = PLAYER_START_X
        self.player_sprite.center_y = PLAYER_START_Y

        # Set the camera to the start
        self.view_left = 0
        self.view_bottom = 0
        changed = True
        arcade.play_sound(self.game_over)
        if self.score > 2:
            self.score -= 3
        else: self.score = 0
        if self.life > 1:
            self.life -= 1
        else:
            self.life = 0

# If you reach a score of 10, consume it to give an extra life point
if self.score == 10:
    self.life += 1
    self.score = 0
```

```
    # Lose 3 points if you die (no negative value)
    if self.score > 2:
        self.score -= 3
    else: self.score = 0

    if self.life > 1:
        self.life -= 1
    else: self.life = 0

# Kill the player if he touches something dangerous
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

    # Set the camera to the start
    self.view_left = 0
    self.view_bottom = 0
    changed = True
    arcade.play_sound(self.game_over)

    if self.score > 2:
        self.score -= 3
    else: self.score = 0

    if self.life > 1:
        self.life -= 1
    else: self.life = 0

if arcade.check_for_collision_with_list(self.player_sprite,
                                        self.trampoline_list):
    self.player_sprite.change_x = 0
    self.player_sprite.change_y = 20
```

```
# Restart the game when you have no more life
if self.life == 0:
    # Restart to the first level
    self.level = 1
    # load the first level
    self.setup(self.level)
    # Set the camera to the start
    self.view_left = 0
    self.view_bottom = 0
    changed = True

# See if the user got 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 = True

self.player_list.update()
self.player_list.update_animation()

# 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 = 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 = 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 = 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 = True

        if changed:
            # Only scroll to integers. Otherwise we end up with pixels
            #that don't line up on the screen
            self.view_bottom = int(self.view_bottom)
            self.view_left = int(self.view_left)

            # Do the scrolling
            arcade.set_viewport(self.view_left,
                                SCREEN_WIDTH + self.view_left,
                                self.view_bottom,
                                SCREEN_HEIGHT + self.view_bottom)
```

```
# Main
# ===========================================
def main():
    """ Main method """
    window = MyGame()
    window.setup(window.level)
    arcade.run()

if _name_ == "_main_":
    main()
# ===========================================
```