Object-oriented programs in Python

Problem E: Maze

Geicu Emil

CEN 2.3A

Description of the problem

For this homework I had to implement the task with the **number 12**.

INDIVIDUAL TASK

12. Implement the Maze game described in Problem E, to which will be added the following facilities. Some predefined areas of the maze will be dark. The player will not see anything inside them (but not the opponents) unless they have previously taken a certain friend (flashlight,for example). The flashlight gives the player the opportunity to see ina circle around him.

MAZE: PROBLEM E

One of the most common computer games is the maze. The maze has an entrance, an exit and is composed of a number of squares (locations), which represent rooms, walls and possibly doors that can be opened.

Inside the maze, in addition to the player, there are other elements: opponents (enemies) that can catch the player (or diminish his power), one or more treasures (of different values) that the player or opponents can take, and a number of friends who can help the player (increase his power, allow him to open the doors of the maze, etc.). Students are free to choose the type and number of enemies and friends based on their experience in this game.

A game begins with the player entering the maze and ends when he exits the maze, is blocked by enemies, or has no power. The score of the game is given by the sum of the values of the accumulated treasures.

The game should be viewed in an iterative manner: at each iteration, the maze and its contents are drawn, then the user is expected to enter a command, after which the state of the maze is recalculated and a new iteration is started.

The maze is a two-dimensional array that contains squares, each square representing a room, a wall, or a door. Rooms may contain a limited number of items (for example, at most one player and another item: a treasure or a friend). Students have the freedom to set this up.

The maze knows the elements inside them, but not their position. The coordination of the maze is decentralized, so that when it is drawn, it lets each component location be drawn, and the location, knowing what elements it contains inside, can draw them.

Remark. The game can be implemented using a graphical interface (drawing is done using the graphical elements of the library), but also without a graphical interface (in which case the maze is seen as an array of characters, each character with a certain meaning. For example: ’ ’ (space) for room ,’#’ for wall, ’+’ for closed door, ’-’ for open door, different letters for player and opponents, etc.).

Communication between the different elements of the game (maze, locations, game elements) is done through messages. For example, a game item has no information about the maze, it only knows the current location. To do this, it will keep a reference to the current location. To make a move, he will query the current location (which in turn only knows its four neighbors) whether the move is valid or not.

The following commands are allowed in the game are:

- east, north, south, west - specifies the player’s movement in one of the
- four directions;
- show location - displays the contents of the location;
- fight - hits the opponent if he is in an adjacent location
- get item (get treasure) - the player takes a friend (or treasure) who is
- in the same location;
- drop item - the player leaves a friend;
- quit - the game is forced to end;
- save - the game is saved;
- load - a previously saved game is loaded;
- help - displays game controls.

After each move of the player, the opponents move according to a certain strategy. Students are free to choose their strategy to move enemies.

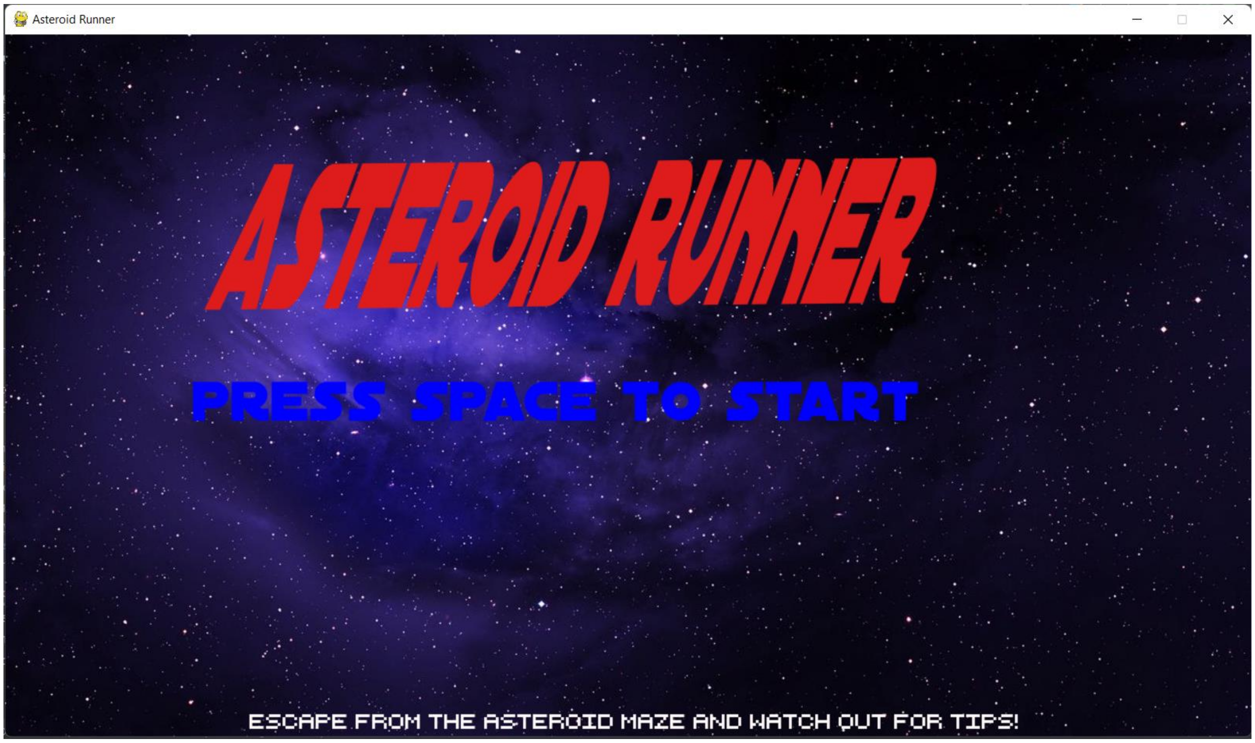
Before the game starts, the maze has to be built (interactively or the maze plan is read from a text file) by specifying its locations as well as the elements it contains.

A quick tour of the game

In order to make the game more appealing I used a graphical interface with the help of the pygame library.

In order to use pygame I made some extensive research on the web, and watched more videos on similar type of games like PacMan and Snake.

First, let's talk about the loading screen.

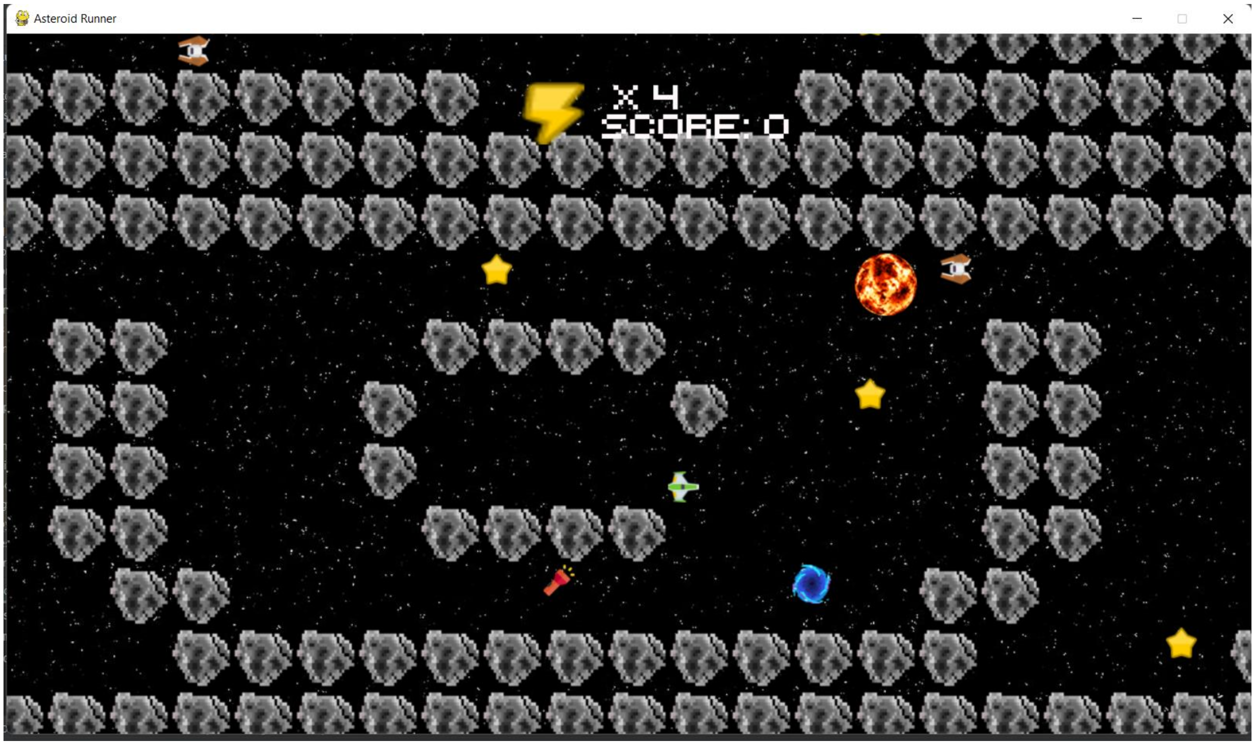


In the loading screen we can see the name of the game, which is ASTEROID RUNNER. In order to start the game we need to press the key SPACE. We can see that the challenge of the game is displayed at the bottom of the loading screen.

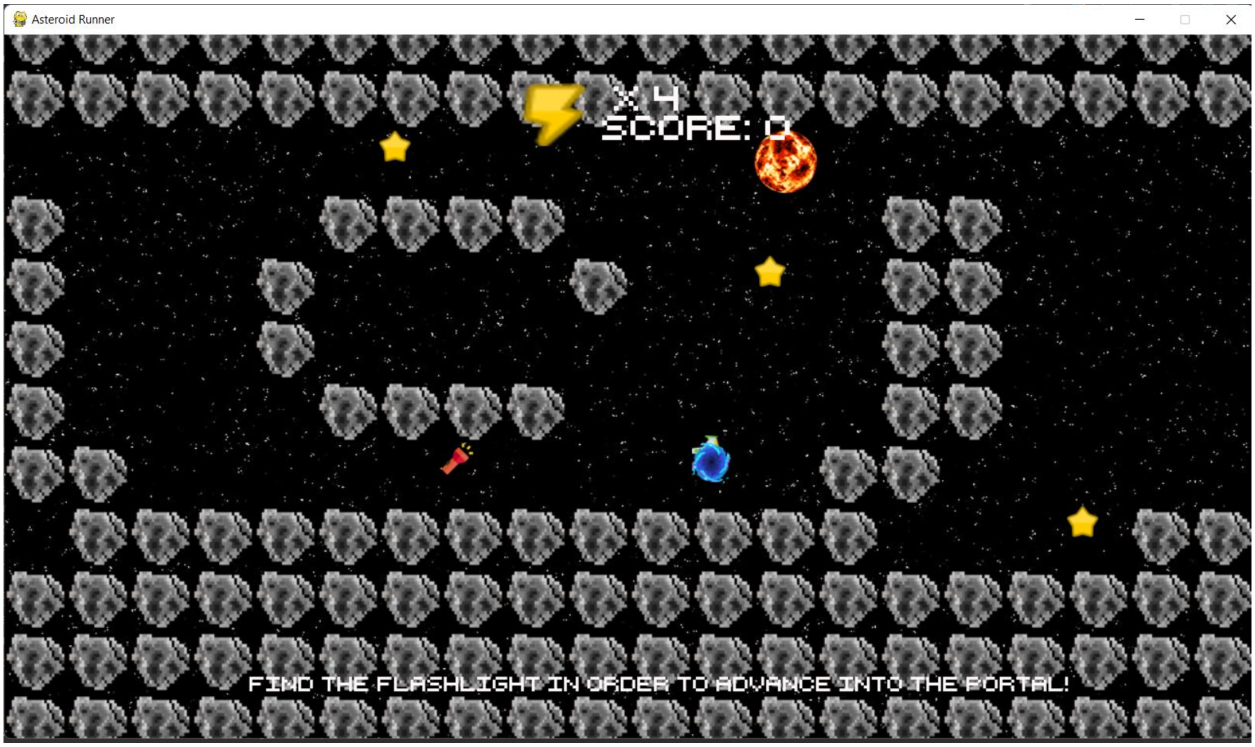
Let’s press the SPACE key.

Now we are at the first level of the game. We can see that there are multiple objects on the screen and writings. Let’s start by explaining the writing at the top. The flash refers to the lives, hearts of the player. When the player gets damaged by an enemy or a sun the lives are decreasing by 1. The score of the player is displayed below the lives. How do we gain score? We need to collect as many stars as we can. The walls of the maze are asteroids. We cannot pass through them. The enemies are represented by the red ships and their direction of moving is random. When they collide with a wall they will get a random direction, different from the previous one. When the ship gets damaged, it becomes invulnerable and takes the red color.

Now we can talk about the most interesting part. I implemented my individual task with the lantern by creating portals. The portals are the point of connection between two levels. In order to advance to the next level we need to find a portal and pass through it. But there is a catch. The portal from the first level has a special permission. Inside level 2 is complete darkness, so we will need a lantern.

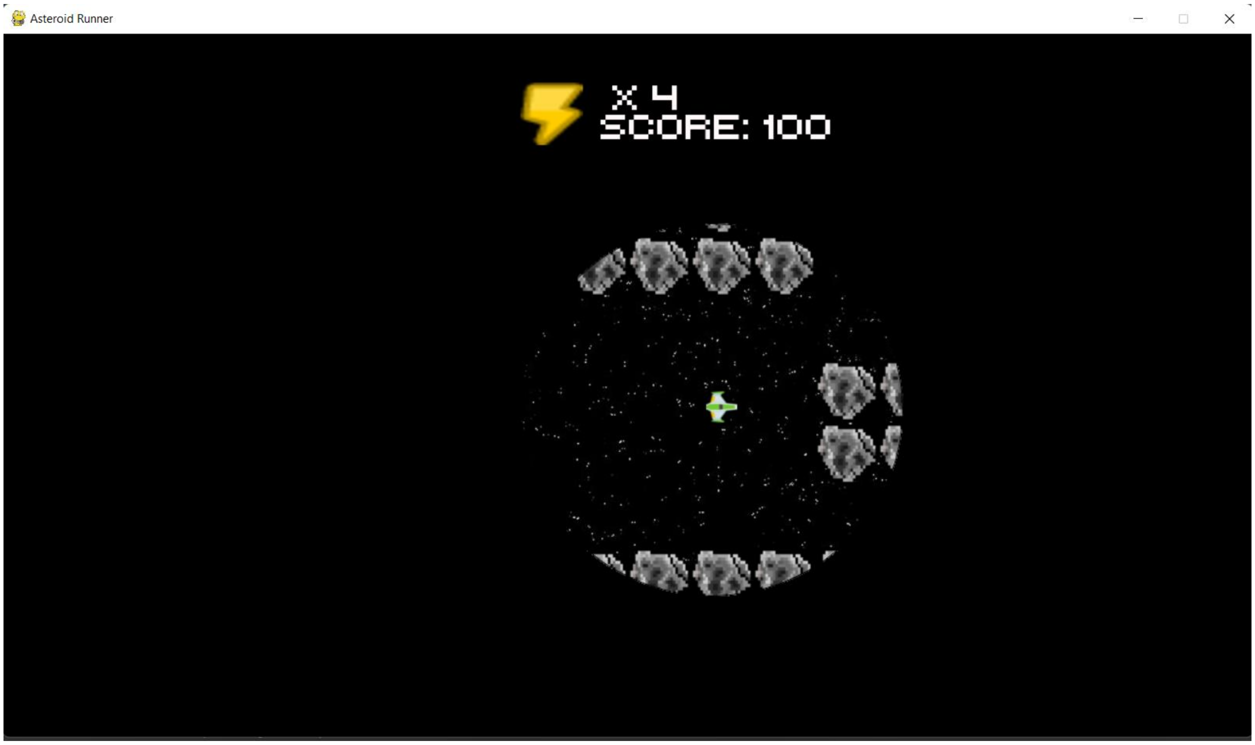


If we try to go through the portal without a lantern we will get a hint at the bottom of the screen (FIND THE FLASHLIGHT IN ORDER TO ADVANCE INTO THE PORTAL).



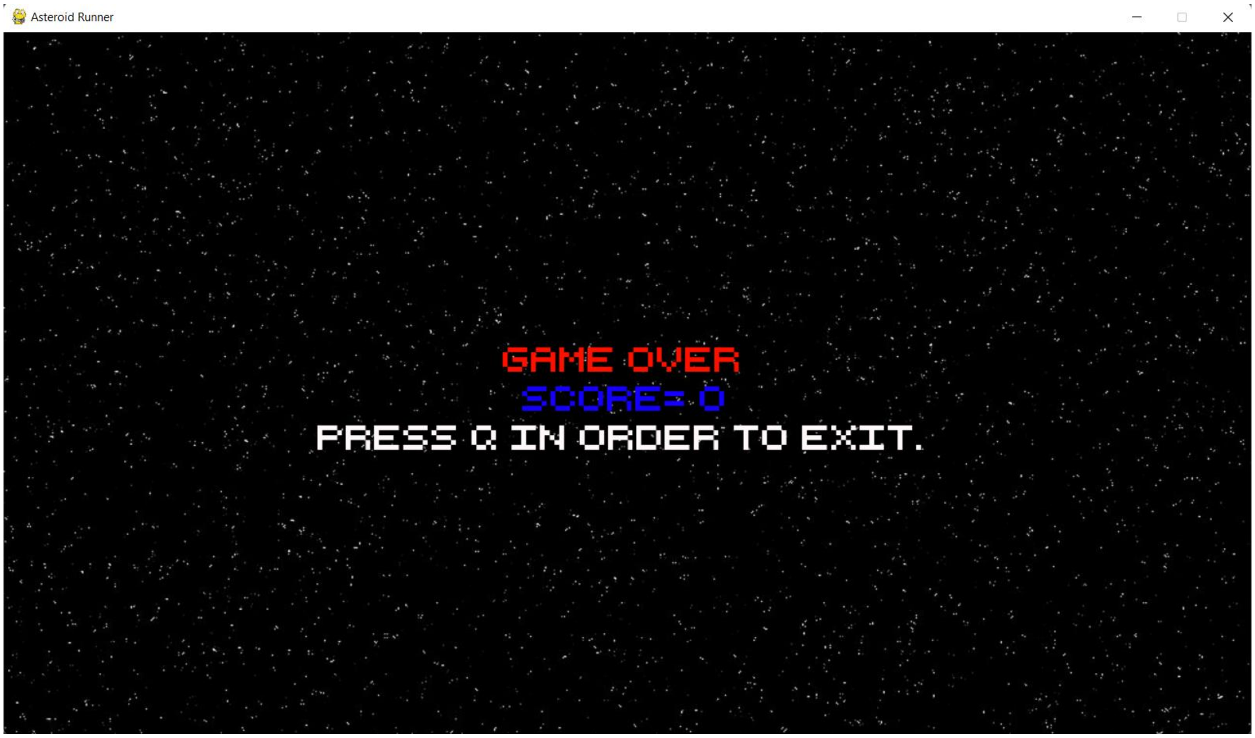
FIND THE FLASHLIGHT IN ORDER TO ADVANCE INTO THE PORTAL!

After we get the flashlight and we get inside the portal we will see the world around us only in a circle.



How the lantern is implemented? I took an image which was cropped in the middle in a form of a circle. Then I fixed the center of this picture to the movement of the player. And now we have a lantern that is following us.

What about the game over screen? The game over screen is triggered only if we die.

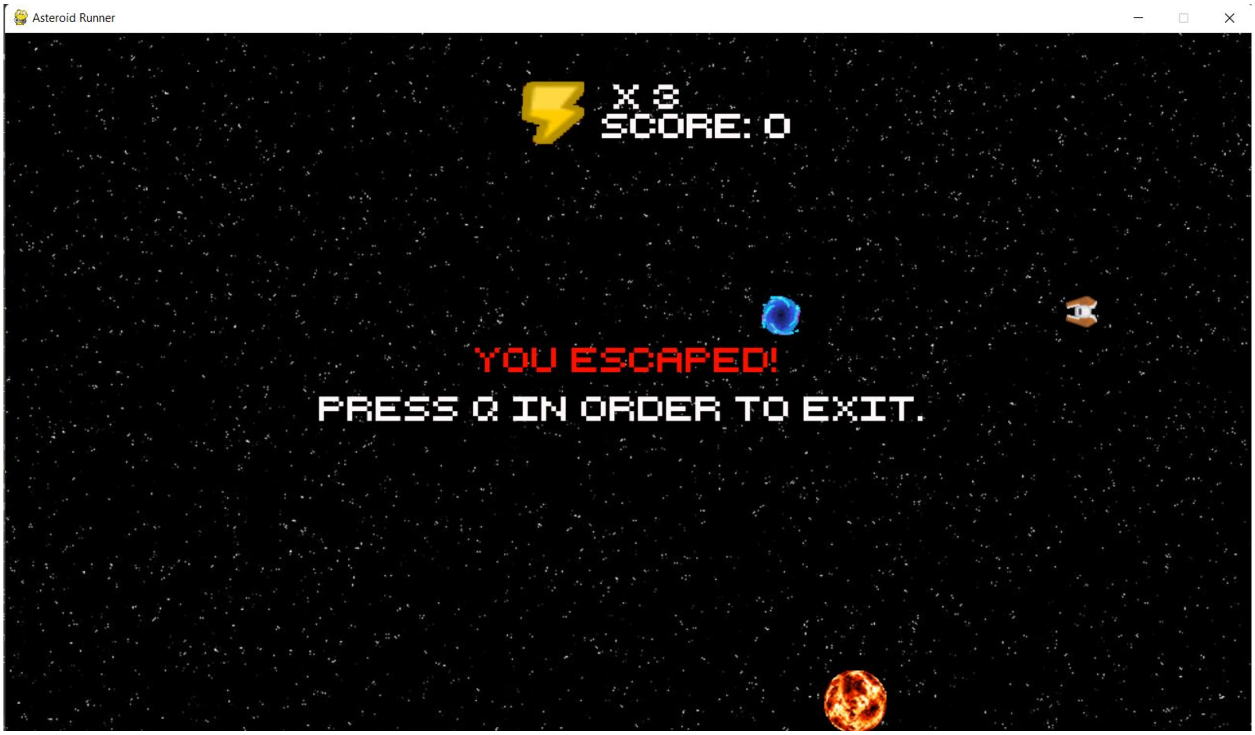


GAME OVER
SCORE= 0
PRESS Q IN ORDER TO EXIT.

It is a simple screen which displays only 3 messages. Game over, the score that we died with and a hint about how to exit the game.

What about the win game scree? The win game screen is triggered only if we escape all the levels.

The win game screen is as simple as the first one, but this one has other information. It displays the score and the lives you escaped with, the message of win game and a hint that tells us how to exit the game.



YOU ESCAPED!

PRESS Q IN ORDER TO EXIT.

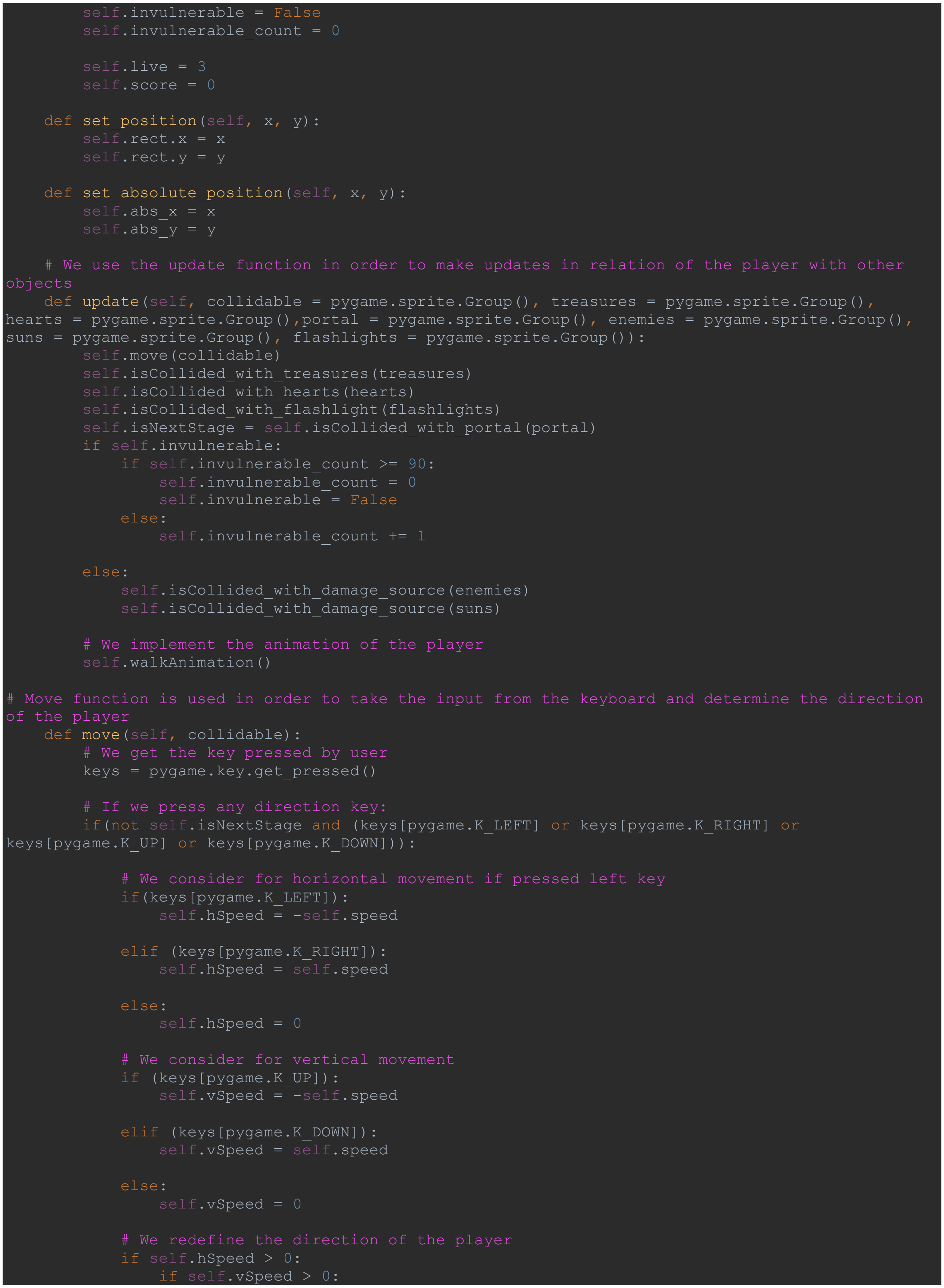
Implementation

In order to make the game I created objects for everything (player, enemy, wall, portal, treasure, sun, heart, flashlight, character, darkness).

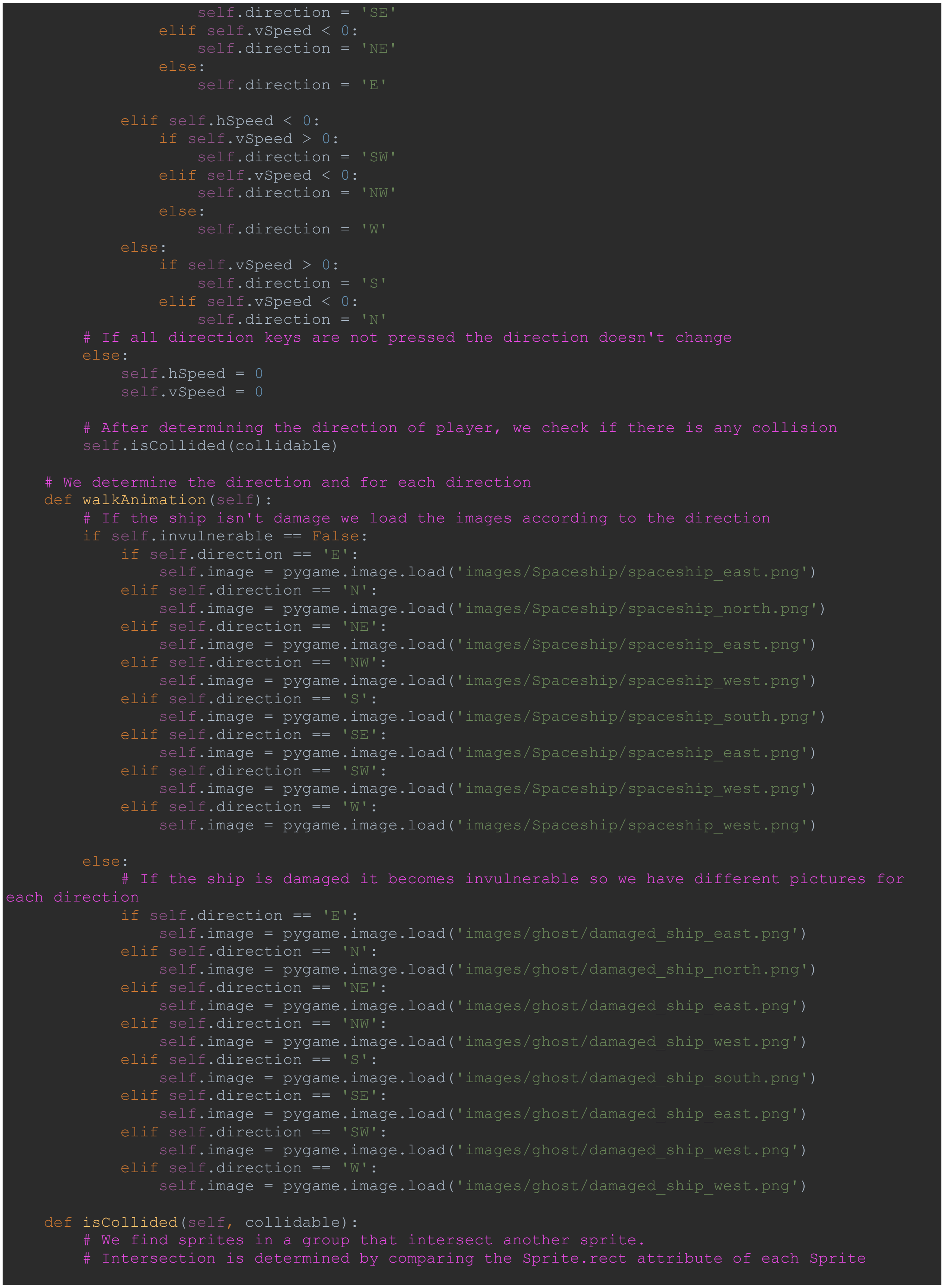
I will present the code for every object with suggestive comments.

Let’s have a look on the Player class:

```
# Player class refers to the ship that we are controlling a
class Player(pygame.sprite.Sprite,Character):
    # set image to be 32 x 32
    def __init__(self):

        # The super function allows the use of pygame.Rect object
        super().__init__()
        Character.__init__(self)

        # We set the image of the ship
        self.image = pygame.image.load("images/Spaceship/spaceship_east.png")

        # We fetch the rectangle dimenstion which has the same dimension as the ship's image
        self.rect = self.image.get_rect()

        self.hSpeed = 0
        self.vSpeed = 0
        self.speed=8
        self.isNextStage = False
        self.walkCount = 0
        self.hasFlashlight = False
        self.direction = 'S'
        self.portalNot = False
        self.ghostWalkCount = 0
```



```
        self.invulnerable = False
        self.invulnerable_count = 0

        self.live = 3
        self.score = 0

    def set_position(self, x, y):
        self.rect.x = x
        self.rect.y = y

    def set_absolute_position(self, x, y):
        self.abs_x = x
        self.abs_y = y

    # We use the update function in order to make updates in relation of the player with other
objects
    def update(self, collidable = pygame.sprite.Group(), treasures = pygame.sprite.Group(),
hearts = pygame.sprite.Group(),portal = pygame.sprite.Group(), enemies = pygame.sprite.Group(),
suns = pygame.sprite.Group(), flashlights = pygame.sprite.Group()):
        self.move(collidable)
        self.isCollided_with_treasures(treasures)
        self.isCollided_with_hearts(hearts)
        self.isCollided_with_flashlight(flashlights)
        self.isNextStage = self.isCollided_with_portal(portal)
        if self.invulnerable:
            if self.invulnerable_count >= 90:
                self.invulnerable_count = 0
                self.invulnerable = False
            else:
                self.invulnerable_count += 1

        else:
            self.isCollided_with_damage_source(enemies)
            self.isCollided_with_damage_source(suns)

        # We implement the animation of the player
        self.walkAnimation()

# Move function is used in order to take the input from the keyboard and determine the direction
of the player
    def move(self, collidable):
        # We get the key pressed by user
        keys = pygame.key.get_pressed()

        # If we press any direction key:
        if(not self.isNextStage and (keys[pygame.K_LEFT] or keys[pygame.K_RIGHT] or
keys[pygame.K_UP] or keys[pygame.K_DOWN])):

            # We consider for horizontal movement if pressed left key
            if(keys[pygame.K_LEFT]):
                self.hSpeed = -self.speed

            elif (keys[pygame.K_RIGHT]):
                self.hSpeed = self.speed

            else:
                self.hSpeed = 0

            # We consider for vertical movement
            if (keys[pygame.K_UP]):
                self.vSpeed = -self.speed

            elif (keys[pygame.K_DOWN]):
                self.vSpeed = self.speed

            else:
                self.vSpeed = 0

            # We redefine the direction of the player
            if self.hSpeed > 0:
                if self.vSpeed > 0:
```



```
                    self.direction = 'SE'
                elif self.vSpeed < 0:
                    self.direction = 'NE'
                else:
                    self.direction = 'E'

            elif self.hSpeed < 0:
                if self.vSpeed > 0:
                    self.direction = 'SW'
                elif self.vSpeed < 0:
                    self.direction = 'NW'
                else:
                    self.direction = 'W'
            else:
                if self.vSpeed > 0:
                    self.direction = 'S'
                elif self.vSpeed < 0:
                    self.direction = 'N'
        # If all direction keys are not pressed the direction doesn't change
        else:
            self.hSpeed = 0
            self.vSpeed = 0

        # After determining the direction of player, we check if there is any collision
        self.isCollided(collidable)

    # We determine the direction and for each direction
    def walkAnimation(self):
        # If the ship isn't damage we load the images according to the direction
        if self.invulnerable == False:
            if self.direction == 'E':
                self.image = pygame.image.load('images/Spaceship/spaceship_east.png')
            elif self.direction == 'N':
                self.image = pygame.image.load('images/Spaceship/spaceship_north.png')
            elif self.direction == 'NE':
                self.image = pygame.image.load('images/Spaceship/spaceship_east.png')
            elif self.direction == 'NW':
                self.image = pygame.image.load('images/Spaceship/spaceship_west.png')
            elif self.direction == 'S':
                self.image = pygame.image.load('images/Spaceship/spaceship_south.png')
            elif self.direction == 'SE':
                self.image = pygame.image.load('images/Spaceship/spaceship_east.png')
            elif self.direction == 'SW':
                self.image = pygame.image.load('images/Spaceship/spaceship_west.png')
            elif self.direction == 'W':
                self.image = pygame.image.load('images/Spaceship/spaceship_west.png')

        else:
            # If the ship is damaged it becomes invulnerable so we have different pictures for
each direction
            if self.direction == 'E':
                self.image = pygame.image.load('images/ghost/damaged_ship_east.png')
            elif self.direction == 'N':
                self.image = pygame.image.load('images/ghost/damaged_ship_north.png')
            elif self.direction == 'NE':
                self.image = pygame.image.load('images/ghost/damaged_ship_east.png')
            elif self.direction == 'NW':
                self.image = pygame.image.load('images/ghost/damaged_ship_west.png')
            elif self.direction == 'S':
                self.image = pygame.image.load('images/ghost/damaged_ship_south.png')
            elif self.direction == 'SE':
                self.image = pygame.image.load('images/ghost/damaged_ship_east.png')
            elif self.direction == 'SW':
                self.image = pygame.image.load('images/ghost/damaged_ship_west.png')
            elif self.direction == 'W':
                self.image = pygame.image.load('images/ghost/damaged_ship_west.png')

    def isCollided(self, collidable):
        # We find sprites in a group that intersect another sprite.
        # Intersection is determined by comparing the Sprite.rect attribute of each Sprite
        self.rect.x += self.hSpeed
        self.abs_x += self.hSpeed

        collision_list = pygame.sprite.spritecollide(self, collidable, False)

        # If intersection with collidable object in collision_list ( horizontal x direction )
        for collided_object in collision_list:
            if (self.hSpeed > 0):
                # We update the Absoulte position
                hDiff = collided_object.rect.left - self.rect.right
                self.abs_x += hDiff
                # We update the relative position
                self.rect.right = collided_object.rect.left
                self.hSpeed = 0

            elif (self.hSpeed < 0):
                # We update the Absoulte position
                hDiff = collided_object.rect.right - self.rect.left
                self.abs_x += hDiff
                # W update the relative position
                self.rect.left = collided_object.rect.right
                self.hSpeed = 0

        self.rect.y += self.vSpeed
        self.abs_y += self.vSpeed
        # If intersection is with collidable object in y direction
        collision_list = pygame.sprite.spritecollide(self, collidable, False)
        for collided_object in collision_list:
            # Moving down
            if (self. vSpeed > 0):
                # We update the Absoulte position
                vDiff = collided_object.rect.top - self.rect.bottom
                self.abs_y += vDiff

                # We update the relative position
                self.rect.bottom= collided_object.rect.top
                self.vSpeed = 0
            # Moving up
            elif (self. vSpeed < 0):
                # We update the Absoulte position
                vDiff = collided_object.rect.bottom - self.rect.top
                self.abs_y += vDiff

                # We update the relative position
                self.rect.top = collided_object.rect.bottom
                self.vSpeed = 0


    # The next part is represented by methods that describe the behaviour of the player with the
collision with every other object

    def isCollided_with_treasures(self, treasures):
        if (pygame.sprite.spritecollide(self, treasures, True)):
            self.score += 100
            star_collision.play()


    def isCollided_with_hearts(self, hearts):
        if (pygame.sprite.spritecollide(self, hearts, True)):
            self.live += 1
            star_collision.play()

    def isCollided_with_portal(self, portals):

        collision_list = pygame.sprite.spritecollide(self, portals, False)
        for portal in collision_list:
            if (self.rect.collidepoint(portal.rect.centerx, portal.rect.centery)):
                if(self.hasFlashlight == True):
                    portal_collision.play()
                    self.portalNot = False
                    return True
                else:
                    self.portalNot = True



    def isCollided_with_damage_source(self, damage_source):
        collision_list = pygame.sprite.spritecollide(self, damage_source, False)
        for item in collision_list:
            if (item.rect.collidepoint(self.rect.centerx, self.rect.centery)):
                enemy_collision.play()
                self.invulnerable = True
                self.live -= 1
                return True

    def isCollided_with_flashlight(self, flashlights):
        if (pygame.sprite.spritecollide(self, flashlights, True)):
            self.hasFlashlight = True
```

Let’s have a look on the Enemy class:

```
# Enemy class refers to the enemy ships of the game
class Enemy(pygame.sprite.Sprite,Character):
    def __init__(self, x, y):

        super().__init__()
        Character.__init__(self)
        self.image = pygame.image.load('images/Enemy/enemy_east.png')
        self.rect = self.image.get_rect()

        self.rect.x = x
        self.rect.y = y


        self.direction = random.choice(["up", "down", "left", "right"])


    def set_position(self, x, y):
        self.rect.x = x
        self.rect.y = y

    def update(self, collidable = pygame.sprite.Group(), collidable_2 = pygame.sprite.Group()):
        self.move()
        self.isCollided(collidable)
        self.isCollided(collidable_2)

# For the move function we use directions in order to make the direction more clear
    def move(self):
        if self.direction == "up":
            self.dx = 0
            self.dy = -self.speed

        elif self.direction == "down":
            self.dx = 0
            self.dy = self.speed

        elif self.direction == "left":
            self.dx = -self.speed
            self.dy = 0

        elif self.direction == "right":
            self.dx = self.speed
            self.dy = 0

        else:
            self.dx = 0
            self.dy = 0

        self.walkAnimation()
        self.rect.x += self.dx
        self.rect.y += self.dy

# The walk animation is make checking the direction of the enemy ship and loading the image
corresponding to that direction
    def walkAnimation(self):
        if self.direction == 'up':
            self.image = pygame.image.load('images/Enemy/enemy_north.png')
        elif self.direction == 'down':
            self.image = pygame.image.load('images/Enemy/enemy_south.png')
        elif self.direction == 'left':
            self.image = pygame.image.load('images/Enemy/enemy_west.png')
        elif self.direction == 'right':
            self.image = pygame.image.load('images/Enemy/enemy_east.png')

# We use this function in order to see if we collide with something
    def isCollided(self, collidable):
        # We check for any enemy collision with walls_Group, if there is a collision, we set the
enemy ship to move in a random direction
        collision_list = pygame.sprite.spritecollide(self, collidable, False)
        for collided_object in collision_list:
            # If the ship is moving right
            if (self.dx > 0):
                self.rect.right = collided_object.rect.left
                self.dx = 0
                self.direction = random.choice(["up", "down", "left"])

            # If the ship is moving  Left
            if (self.dx < 0):
                self.rect.left = collided_object.rect.right
                self.dx = 0
                self.direction = random.choice(["up", "down", "right"])

            # If the ship is moving  down
            if (self.dy > 0):
                self.rect.bottom= collided_object.rect.top
                self.dy = 0
                self.direction = random.choice(["up", "left", "right"])
            # If the ship is moving  up
            if (self.dy < 0):
                self.dx = 0
                self.rect.top = collided_object.rect.bottom
                self.direction = random.choice(["down", "left", "right"])

    def shift_world(self, shift_x, shift_y):
        self.rect.x += shift_x
        self.rect.y += shift_y
```

Let’s have a look on the Wall class:

```
# Wall class refers to the walls of the maze which in our case are some asteroids
class Wall(pygame.sprite.Sprite):

    def __init__(self, x, y):

        super().__init__()
        self.image = pygame.image.load('images/others/wall.png').convert_alpha()
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y

    def shift_world(self, shift_x, shift_y):
        self.rect.x += shift_x
        self.rect.y += shift_y

    def draw(self, window):

        window.blit(self.image,(self.rect.x, self.rect.y))
```

Let’s have a look on the Flashlight class:

```
# Flashlight class refers to the flashlight that lets us advance to the levels that are in the
dark
class Flashlight(pygame.sprite.Sprite):

    def __init__(self, x, y):
        super().__init__()
        self.image = pygame.image.load('images/Flashlight/Flashlight.png')
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y

    def shift_world(self, shift_x, shift_y):
        self.rect.x += shift_x
        self.rect.y += shift_y
```

Let’s have a look on the Treasure class:

```
# Treasure class refers to little stars that are made as collectable objects. If we pe pick them
out our score increases
class Treasure(pygame.sprite.Sprite):

    def __init__(self, x, y):

        super().__init__()
        self.image = pygame.image.load('images/others/foodA.png').convert_alpha()

        self.rect = self.image.get_rect()

        self.rect.x = x
        self.rect.y = y

    def shift_world(self, shift_x, shift_y):
        self.rect.x += shift_x
        self.rect.y += shift_y
```

Let’s have a look on the Heart class:

```
# Heart class refers to our energy or lives of the ship
class Heart(pygame.sprite.Sprite):

    def __init__(self, x, y):

        super().__init__()
        self.image = pygame.image.load('images/features/heart.png').convert_alpha()

        self.rect = self.image.get_rect()

        self.rect.x = x
        self.rect.y = y

    def shift_world(self, shift_x, shift_y):
        self.rect.x += shift_x
        self.rect.y += shift_y
```

Let’s have a look on the Sun class:

```
# Sun class makes dangerous objects for our ship. If we collide with a sun on heart is depleted
class Sun(pygame.sprite.Sprite):

    def __init__(self, x, y):

        super().__init__()

        self.image = pygame.image.load('images/Sun/star.png')

        self.rect = self.image.get_rect()

        self.rect.x = x
        self.rect.y = y

        self.count = 0

    def shift_world(self, shift_x, shift_y):
        self.rect.x += shift_x
        self.rect.y += shift_y
```

Let’s have a look on the Portal class:

```
# Portal class is used for making portals that help us advance to the next levels
class Portal(pygame.sprite.Sprite):

    def __init__(self, x, y):

        super().__init__()
        self.image = pygame.image.load('images/portal/portal.png').convert_alpha()


        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y
        self.count = 0



    def shift_world(self, shift_x, shift_y):
        self.rect.x += shift_x
        self.rect.y += shift_y
```

Let’s have a look on the Darkness class:

```
# Darkness class is used to make the ilusion of a lantern in the dark from level 2 onwards
class Darkness(pygame.sprite.Sprite):
    def __init__(self):

        super().__init__()
        self.image = pygame.image.load('images/others/fog.png').convert_alpha()

        self.rect = self.image.get_rect()

    def update(self, player_x, player_y):
        self.rect.centerx = player_x + 16
        self.rect.centery = player_y + 16
```

Let’s have a look on the Character class:

```
# In the upcoming classes we wil make use of the inheritance
# We keep in mind that pygame.sprite.Sprite is the base class for visible game objects
# Class character is used for the player and the enemy in order to demonstrate polymorphism on
function overwriting
class Character:
    def __init__(self):
        self.speed = 4
    def walkAnimation(self):
        print("walk")
```

Let’s have a look on the ‘settings’ of the game:

```
# The size of the game window
screen_width, screen_height = 1280, 720
screen_size = screen_width, screen_height
```

```
window = pygame.display.set_mode(screen_size)
fade = pygame.Surface((screen_width, screen_height))
fade.fill((0,0,0))


# We make the name of the game window Asteroid Runner
pygame.display.set_caption("Asteroid Runner")

# We create an object to keep track of time
clock = pygame.time.Clock()

# We will consider 60 frames per second
fps = 60
```

We have a function named create_instances() which initializes all the objects relevant to the game:

```
# Here we initialize all the objects relevant to the game.
def create_instances():
    global current_level, running, player, player_group, darkness_group, flashlight_group
    global walls_group, enemies_group, treasures_group, hearts_group, portal_group, suns_group
    global screen_width, screen_height
    global winGame

    winGame = False
    current_level = 0

    # We make our player instance
    player = Player()
    player_group = pygame.sprite.Group()
    player_group.add(player)

    # We make groups for the other objects aswell
    walls_group = pygame.sprite.Group()
    enemies_group = pygame.sprite.Group()
    treasures_group = pygame.sprite.Group()
    portal_group = pygame.sprite.Group()
    suns_group = pygame.sprite.Group()
    hearts_group = pygame.sprite.Group()
    flashlight_group = pygame.sprite.Group()
    darkness_group = pygame.sprite.Group()
    darkness_group.add(Darkness())
```

We have a function named run_viewbox() which keeps the player in the focus of the game:

```
# Here we do the managing of our camera
def run_viewbox(player_x, player_y):

    left_viewbox = screen_width / 2 - screen_width / 8
    right_viewbox = screen_width / 2 + screen_width / 8
    top_viewbox = screen_height / 2 - screen_height / 8
    bottom_viewbox = screen_height / 2 + screen_height / 8
    dx, dy = 0, 0

    if(player_x <= left_viewbox):
        dx = left_viewbox - player_x
        player.set_position(left_viewbox, player.rect.y)

    elif(player_x >= right_viewbox):
        dx = right_viewbox - player_x
        player.set_position(right_viewbox, player.rect.y)

    if(player_y <= top_viewbox):
        dy = top_viewbox - player_y
        player.set_position(player.rect.x, top_viewbox)

    elif(player_y >= bottom_viewbox):
        dy = bottom_viewbox - player_y
        player.set_position(player.rect.x, bottom_viewbox)
    if (dx != 0 or dy != 0):
        for wall in walls_group:
            wall.shift_world(dx, dy)

        for enemy in enemies_group:
            enemy.shift_world(dx, dy)

        for treasure in treasures_group:
            treasure.shift_world(dx, dy)

        for heart in hearts_group:
            heart.shift_world(dx, dy)

        for portal in portal_group:
            portal.shift_world(dx, dy)

        for sun in suns_group:
            sun.shift_world(dx, dy)

        for flashlight in flashlight_group:
            flashlight.shift_world(dx,dy)
```

We have a function which does the setup of our maze:

```
# Here we setup our maze
def setup_maze(current_level):

    # We take every character of the matrix and we decode it
    for y in range(len(levels[current_level])):
        for x in range(len(levels[current_level][y])):
            character = levels[current_level][y][x]
            pos_x = (x*64)
            pos_y = (y*64)

            if character == "X":
                # We update wall coordinates
                walls_group.add(Wall(pos_x, pos_y))

            elif character == "F":
                # We update flashlight coordinates
                flashlight_group.add(Flashlight(pos_x, pos_y))

            elif character == "P":
                # We update player coordinates
                player.set_position(pos_x, pos_y)
                player.set_absolute_position(pos_x, pos_y)

            elif character == "E":
                # We update  enemy coordinates
                enemies_group.add(Enemy(pos_x, pos_y))

            elif character == "T":
                # We update  treasure coordinates
                treasures_group.add(Treasure(pos_x, pos_y))

            elif character == "H":
                # We update hearts coordinates
                hearts_group.add(Heart(pos_x, pos_y))

            elif character == "U":
                # We update  portal coordinates
                portal_group.add(Portal(pos_x, pos_y))

            elif character == "S":
                #Update suns coordinates
                suns_group.add(Sun(pos_x, pos_y))
```

We have a function which clears the maze:

```
# Here we clear our current level
def clear_maze():
    walls_group.empty()
    enemies_group.empty()
    treasures_group.empty()
    hearts_group.empty()
    suns_group.empty()
    portal_group.empty()
    flashlight_group.empty()
    player.isNextStage = False
```

We have a function named nextStage() which verifies the state of the player (whether we need or not to advance to the next level of if we have won the game).

```
# Here we make the transition between levels
def nextStage(isNextStage):
    global current_level, isGameOver
    global winGame
    # If we got to a new level we have 2 possibilities
    if isNextStage:
        current_level += 1

        # The first possibiliy is that we conquered all the levels and escaped the maze
        # In this case we will display the message YOU ESCAPED along with the score and lives of the player
        if current_level >= 4:
            winGame = True
            window.blit(background,(0,0))
            gameovertext = font1.render("YOU ESCAPED!", 1, (255, 17, 0))
            tip = font1.render("Press Q in order to exit.", 1, (255, 250, 250))
            gameoverScore = font1.render("Score= " + str(player.score), 1, (21, 0, 255))
            window.blit(gameovertext, (screen_width // 2 - 160, screen_height // 2 - 50))
            window.blit(tip, (screen_width // 2 - 320, screen_height // 2))
            run_viewbox(screen_width // 2 - 200, screen_height // 2)

        # The second possibilty is that we still have levels to escape so we clear the current level and we setup the next one
        elif winGame == False:
            clear_maze()
            setup_maze(current_level)
```

Let’s have a look on some setting related to the backgrounds and audio of the game:

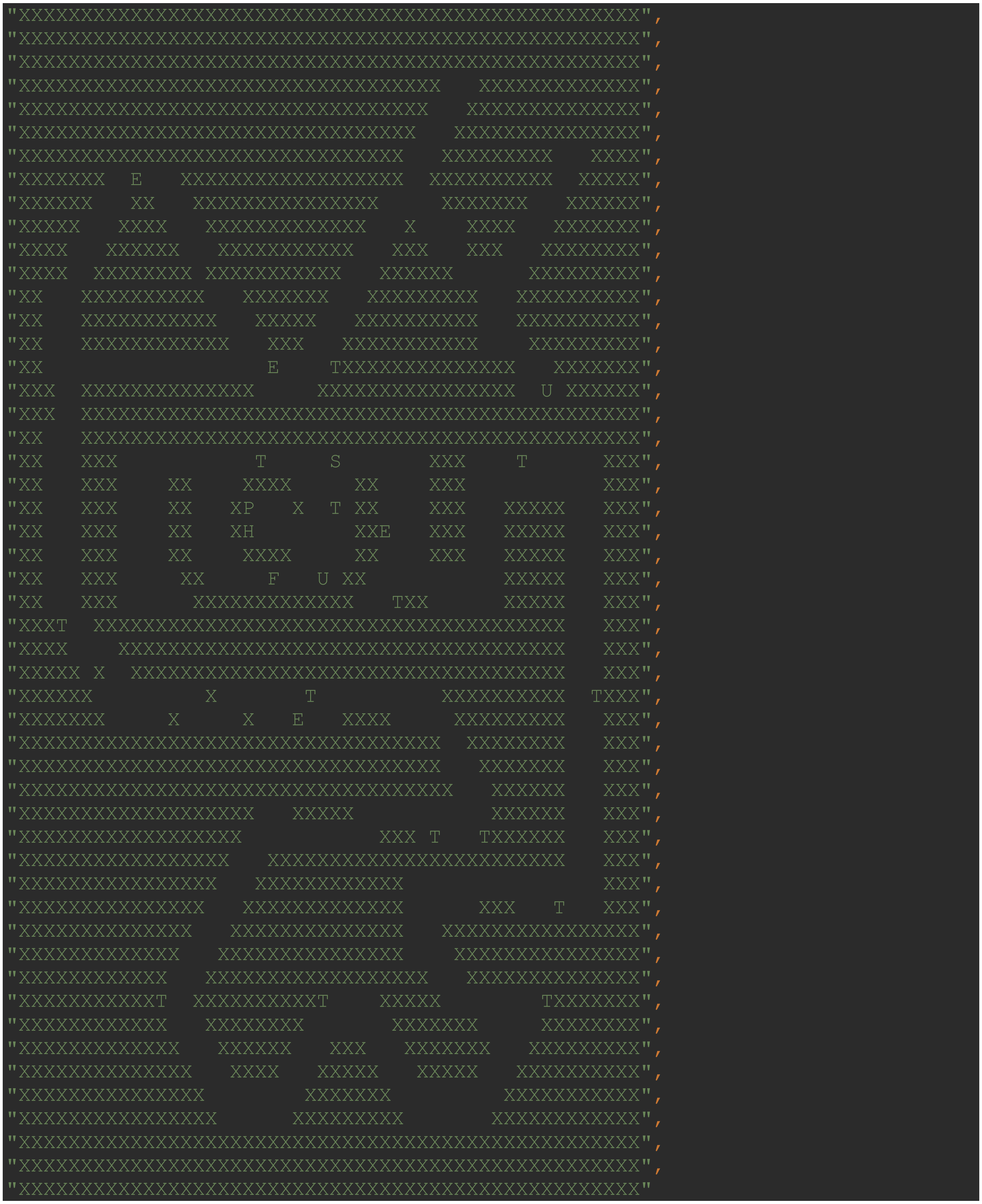
```
# We load the background and the loading screen of the game

background = pygame.image.load('images/others/background.jpg')
loadingScreen = pygame.image.load('images/others/loadingscreen.png')
heartShape = pygame.image.load('images/features/heart.png')

# We load the audio of each interaction of the player with the objects
# We adjust the volume also, because the default one is too loud

music = pygame.mixer.music.load(os.path.join('audios','Background_Music.mp3'))
pygame.mixer.music.play(-1)
pygame.mixer.music.set_volume(0.1)
star_collision = pygame.mixer.Sound(os.path.join('audios', 'Star_Collision.wav'))
star_collision.set_volume(0.1)
enemy_collision = pygame.mixer.Sound(os.path.join('audios','Enemy_Collision.wav'))
enemy_collision.set_volume(0.1)
portal_collision = pygame.mixer.Sound(os.path.join('audios','Portal_Collision.wav'))
portal_collision.set_volume(0.1)

# We load the same custom font but with different dimensions
```

```
font1 = pygame.font.Font('images/Font/retrofont.ttf',35)
font2 = pygame.font.Font('images/Font/retrofont.ttf',20)
```

Finally, we have the main function in which all the other elements of the game combine.

```
# Here in main(), we manage the states of the game

def main():

# We initialize the states of the game (loading state, game is over state, running state)
    loading = True
    isGameOver = False
    running = True

# We create the instances of the game and we make the correspondence between every character of
each level matrix with its coresponding object
    create_instances()
    setup_maze(current_level)


# We initialize i with 0 and we will need it when we will make the transition between levels
    i = 0

# We make updates while the game is running
    while running:

        # While the game is running and we press the exit button of the window or the key q we
terminate and exit the game
        for event in pygame.event.get():
            if(event.type == pygame.QUIT) or (event.type == pygame.KEYDOWN and (event.key ==
pygame.K_ESCAPE or event.key == pygame.K_q )):
             running = False



        # The first state is the loading state where we see the title screen and at the bottom
the main objective of the game
        if loading:
            breakLoop = True
            while breakLoop:
                window.blit(loadingScreen, (0,0))
                tip = font2.render("Escape from the asteroid maze and watch out for tips!", 1,
(255, 250, 250))
                window.blit(tip, (250, screen_height - 30))
                pygame.display.flip()

                # If we press the space key down we will end the loading screen and we will move
to the actual playing state of the game
                for event in pygame.event.get():
                    if event.type == pygame.KEYDOWN and event.key == pygame.K_SPACE:
                        breakLoop = False
                        loading = False

        # If we run out of lives (hearts, energy, etc.) The program will display the message game
over along with the current score
        elif isGameOver:
            window.blit(background,(0,0))
            gameovertext = font1.render("GAME OVER",1,(255, 17, 0))
            tip = font1.render("Press Q in order to exit.",1,(255,250,250))
            gameoverScore = font1.render("Score= " + str(player.score),1,(21, 0, 255))
            window.blit(gameovertext, (screen_width // 2 - 130, screen_height // 2 - 50))
            window.blit(gameoverScore, (screen_width // 2 - 110, screen_height // 2 - 10))
            window.blit(tip, (screen_width // 2 - 320, screen_height // 2 + 30))

        # If the player is still in the current level (stage) we make updates for every object
that we see on our screen
        else:
            if (player.isNextStage != True):
                player_group.update(walls_group, treasures_group, hearts_group, portal_group,
enemies_group, suns_group, flashlight_group)
                enemies_group.update(walls_group)
                darkness_group.update(player.rect.x, player.rect.y)


                # We update out view camera
                run_viewbox(player.rect.x, player.rect.y)

                # We make the background of the level to an environement simlar to space (The
background is an image with a lot of stars)
                window.blit(background,(0,0))

                # We draw the walls
                for wall in walls_group:
                    if (wall.rect.x < screen_width) and (wall.rect.y < screen_height):
                        wall.draw(window)

            # We draw the remaining objects

            if player.isNextStage != True:
                player_group.draw(window)

            flashlight_group.draw(window)
            portal_group.draw(window)
            treasures_group.draw(window)
            hearts_group.draw(window)
            enemies_group.draw(window)

            suns_group.draw(window)

            # Implement fog from level 2 onwards
            if current_level >= 1 and winGame == False:
                darkness_group.draw(window)

            if player.portalNot == True:
                text = font2.render("Find the flashlight in order to advance into the
portal!",1,(255,250,250))
                window.blit(text,(250,650))

            lifeLeftText = font1.render(' X '+ str(player.live),1,(255,250,250))
            scoreText = font1.render('Score: ' + str(player.score), 1, (255,250,250))
            window.blit(heartShape,(530,50))
            window.blit(lifeLeftText,(610,40))
            window.blit(scoreText, (610, 70))

            # If we advance to the next stage and we have not yet won the game we make a fading
animation in order to make a
            #smooth transition between the completed level and the new level

            if player.isNextStage ==  True and winGame == False:
                fade.set_alpha(i)
                window.blit(fade, (0,0))
                pygame.display.update()
                pygame.time.delay(2)
                i += 15
                if i == 255:
                    i = 0
                    nextStage(player.isNextStage)
                    continue

            # If all the lives (energy, hearts) of the player run out the game is over
            if player.live == 0:
                isGameOver = True

        # We delay and update out screen
        pygame.display.flip()
        clock.tick_busy_loop(fps)
    pygame.quit()
```

Let’s have a look at what’s behind a maze:



```
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   XXXXXXXXXXXXX",
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   XXXXXXXXXXXXXX",
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXX",
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   XXXXXXXXX   XXXX",
"XXXXXXX  E   XXXXXXXXXXXXXXXXXX  XXXXXXXXXX  XXXXX",
"XXXXXX   XX   XXXXXXXXXXXXXXX     XXXXXXX   XXXXXX",
"XXXXX   XXXX   XXXXXXXXXXXXX   X    XXXX   XXXXXXX",
"XXXX   XXXXXX   XXXXXXXXXXX   XXX   XXX   XXXXXXXX",
"XXXX  XXXXXXXX XXXXXXXXXXX   XXXXXX      XXXXXXXXX",
"XX   XXXXXXXXXX   XXXXXXX   XXXXXXXXX   XXXXXXXXXX",
"XX   XXXXXXXXXXX   XXXXX   XXXXXXXXXX   XXXXXXXXXX",
"XX   XXXXXXXXXXXX   XXX   XXXXXXXXXXX    XXXXXXXXX",
"XX                  E    TXXXXXXXXXXXXXX   XXXXXXX",
"XXX  XXXXXXXXXXXXXX     XXXXXXXXXXXXXXXX  U XXXXXX",
"XXX  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
"XX   XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
"XX   XXX           T     S       XXX    T      XXX",
"XX   XXX    XX    XXXX     XX    XXX           XXX",
"XX   XXX    XX   XP   X  T XX    XXX   XXXXX   XXX",
"XX   XXX    XX   XH        XXE   XXX   XXXXX   XXX",
"XX   XXX    XX    XXXX     XX    XXX   XXXXX   XXX",
"XX   XXX     XX     F   U XX           XXXXX   XXX",
"XX   XXX      XXXXXXXXXXXXX   TXX      XXXXX   XXX",
"XXXT  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   XXX",
"XXXX    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   XXX",
"XXXXX X  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   XXX",
"XXXXXX         X       T          XXXXXXXXXX  TXXX",
"XXXXXXX     X     X   E   XXXX     XXXXXXXXX   XXX",
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  XXXXXXXX   XXX",
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   XXXXXXX   XXX",
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX   XXXXXX   XXX",
"XXXXXXXXXXXXXXXXXXX   XXXXX           XXXXXX   XXX",
"XXXXXXXXXXXXXXXXXX           XXX T   TXXXXXX   XXX",
"XXXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXXXXXXXXXX   XXX",
"XXXXXXXXXXXXXXXX   XXXXXXXXXXXX                XXX",
"XXXXXXXXXXXXXXX   XXXXXXXXXXXXX      XXX   T   XXX",
"XXXXXXXXXXXXXX   XXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX",
"XXXXXXXXXXXXX   XXXXXXXXXXXXXXX    XXXXXXXXXXXXXXX",
"XXXXXXXXXXXX   XXXXXXXXXXXXXXXXXX   XXXXXXXXXXXXXX",
"XXXXXXXXXXXT  XXXXXXXXXXT    XXXXX        TXXXXXXX",
"XXXXXXXXXXXX   XXXXXXXX       XXXXXXX     XXXXXXXX",
"XXXXXXXXXXXXX   XXXXXX   XXX   XXXXXXX   XXXXXXXXX",
"XXXXXXXXXXXXXX   XXXX   XXXXX   XXXXX   XXXXXXXXXX",
"XXXXXXXXXXXXXXX        XXXXXXX         XXXXXXXXXXX",
"XXXXXXXXXXXXXXXX      XXXXXXXXX       XXXXXXXXXXXX",
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
```

Conclusion

I consider that this project helped me develop in the field of very basic game programming making use of the Oop features learned at the course and the laboratory. The ones that I implemented in my code are inheritance and polymorphism. With this project I learned how to be patient and how to search pieces of information in different kinds of sources.