Jetpack Joyride

By Shradha Sehgal - 2018101071

Overview

An arcade game in Python3 (terminal-based), heavily inspired by Jetpack Joyride where the user controls the mandalorian, and can move it up, forward and backward,while collecting coins and fighting/dodging its enemies on the way.

Rules of the Game

> - You control Mandalorian throughout his efforts (to save Yoda) by avoiding / killing enemy beams, collecting points, and dodging magnets.
> - *Contact* with the enemy can kill your Mandalorian.
> - You have 5 lives for your Mando, getting killed 5 times will result in a **GAME OVER**.
> - The Mandalorian can shoot bullets at the enemy beams.
> - Destroying each beam adds 5 points to the score.
> - The game lasts for 150 seconds, you have to kill all enemies (including the boss enemy dragon) in that time or you lose.

Description of Classes Created

Board:

The board class creates a 50*1200 board for gameplay, with boundaries, walls and empty spaces. It also comprises of a print_board function to take a print of the board.

Object:

The Object class is the base class based on which all other entities of the game are inherited.

Mando:

The Mando class has all the variables and functionality of Mandalorian, this includes the generation, movement and firing of bullets. It is inherited from Object class and has additional functionality. It also represents polymorphism as the render() function has been changed.

Dragon:

The Dragon class is inherited from the Object class and has functionality to move alongside the Mando, and shoot bullets at it. It also has additional private variables such as lives_remaining etc.

Bullet:

Inherited from Object class, it has functions that check for collisions with any item / dragon.

Dragon_Bullet:

Inherited from Object class, it has functions to check collisions with Mando.

Concepts used

Inheritance:

Inheritance allows us to define a class that inherits all the methods and properties from another class. A base class `Object` has been declared from which multiple elements are inherited.

```
class Object():

    def __init__(self, character, x, y):
        self._posx = x
        self._posy = y
        self._width = len(character[0])
        self._height = len(character)
        self._shape = character
```

Polymorphism

Polymorphism allows us to define methods in the child class with the same name as defined in their parent class. eg.

```
class Object():
    ...
    def render(self):
        for i in range(self._width):
            for j in range(self._height):
                # print(j+self._posy, i+self._posx)
                global_var.mp.matrix[j+self._posy][i+self._posx] = self._shape
```

```
class Mando(Object):
    def render(self):
        if self.__shield == 1:
            for i in range(self._width):
                for j in range(self._height):
                    global_var.mp.matrix[j+self._posy][i+self._posx] = self._s

        else:
            for i in range(self._width):
                for j in range(self._height):
                    global_var.mp.matrix[j+self._posy][i+self._posx] = self._s
```

Encapsulation

The idea of wrapping data and the methods that work on data within one unit. Prevents accidental modification of data. Implemented many classes and objects for the same.

Abstraction

Abstraction means hiding the complexity and only showing the essential features of the object.

```
def My_Dragon(Object):
    ...
    def move(self):
        self.clear()
        self._posx += 1
        self.change_shape()
        self.render()
```

.move() is an abstraction

How To Play:

> - Run `pip3 install requirements.txt`
> - Run the following code to start the game.

```
python3 main.py
```

> - 'w, a, d' use these controls for up, left, and right.
> - use 'e' to shoot.
> - use ' ' to activate shield.
> - press 'q' to quit.

Reqiurements:

- Python3

For mac:

```
brew cask update
sudo brew cask install python3
```

For Linux:

```
sudo apt-get update
sudo apt-get install python3
```