DASS_Assignment2

February 20, 2021

1 DX Ball Lite

1.0.1 Tanishq Goel - 2019114015

1.1 Controls

- A: move backward
- D: move forward
- E: shoot the ball
- Q: quit

1.2 Overview

An arcade game in Python3 (terminal-based), heavily inspired by DX Ball where the user controls the paddle, and can move it forward and backward to break the bricks. Also watch out for the power ups.

1.3 Rules of the Game

- You control paddle
- You have 3 lives for your paddle, missing ball 3 times will result in a **GAME OVER**.
- Destroying each brick adds 1 points to the score.
- Score maxiumum in minimum amount of time

1.4 Description of Classes Created

Board: The board class creates a 40*100 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. Paddle and Ball are subclasses of Object class.

Paddle: The Paddle class has all the variables and functionality of a paddle, this includes the size and movement It is inherited from Object class and has additional functionality.

Brick: The Brick is a new class seperate from Object class and has it's own different funcitonalities. It has different subclasses for different types of bricks. It also represents polymorphism as the check_collison() function has been changed.

Ball: Inherited from Object class, it has functions that check for collisions with any brick/ paddle. It also represents polymorphism as the render() function and clear() function has been changed.

Power-ups: The Brick is a new class seperate from Object class and has it's own different funcitonalities. It has different subclasses for different types of bricks. It also represents polymorphism as the start_pow() function has been changed. It overrides the same function present in the parent class. _____

1.5 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
```

A base class `Brick` has been declared from which multiple elements are inherited.

```
class Brick():
    def __init__(self,x,y):
        self.height=1
        self.width=9
        self._posx=x
        self._posy=y
        self.isthere = 0
```

A base class `Powerups` has been declared from which multiple elements are inherited.

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

```
        self._shape = character
        self.FLAG=0
        self.speed=1
        self.samay=0
        self.check=0
```

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

```
class Brick():
    ...
        def check_collision_brick(self):
            for i in range(self.width):
                for j in range(self.height):
                    if i + self._posx==global_var.ball.xget() + config.Ball_speedx and self._po
                        if(config.thru_flag==1):
                            self.strength=0
                            self.isthere=1
                            config.score+=1
                            for i in range(self.width):
                                for j in range(self.height):
                                    global_var.mp.matrix[j+self._posy][i+self._posx ] = " "


class Magenta_Bricks(Object):
    def check_collision_brick(self):
            for i in range(self.width):
                for j in range(self.height):
                    if i + self._posx==global_var.ball.xget() +  config.Ball_speedx and self._
                        if(config.thru_flag==1):
                            self.strength=0
                            config.blastflag=1
                            config.blasty=self._posy
                            if(self.strength==0):
                                self.isthere=1
                                for i in range(self.width):
                                    for j in range(self.height):
                                        global_var.mp.matrix[j+self._posy][i+self._posx ] = "
```

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()

class Ball(Object):
    ...
    def check_collision(self):
        if(self.yget()==global_var.paddle._posy and config.Ball_speedy>0):
            for i in range (0,global_var.paddle._width):
                    ...
                if(self.xget() ==global_var.paddle.xget()+i):

                    if(i==0 or i==1):
                        config.Ball_speedx-=j
                        config.Ball_speedy*=-1

                    elif(i==global_var.paddle._width -1 or i==global_var.paddle._width -2):
                        config.Ball_speedx+=j
                        config.Ball_speedy*=-1

                    else:
                        config.Ball_speedx+=j
                        config.Ball_speedy*=-1
```

.check_collision() is an abstraction

1.6 How To Play:

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

```
python3 main.py
```

1.7 Reqiurements:

- Python3

For mac:

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

For Linux: “‘ sudo apt-get update sudo apt-get install python3