Brick Breaker

Project 2 *(part 2)*

Deadline: Sunday, 27th May 2018, **11:59 PM**

Instructions

Note: You can form a group of MAX 2 students. The group must be the same as part 1. Read the following description VERY CAREFULLY.

1 Background

In the previous part you made the overall design of the game that you will now implement. The accompanied start package "`p2_starter.zip`" contains class diagram, source code and game art assets.

The "`assets`" contains various images that you will use to create the game.

The single "`Breakout.py`" source file contains all of the source code for our game. You must write your code in this file.

For your reference the class diagram is included as "`ClassDiagram.pdf`". Only the classes and their relationships are mentioned. You need to implement attributes and behaviors yourself!

Hint: Q) Why are we using Object Oriented Programming? A) Because we want every object to do its own dirty laundry. Each object should perform its own responsibilities and should maintain its own state. It should have its own logic, it should know how to update itself and how to draw itself !

2 Game Implementation

Open up "`Breakout.py`" and study the source code ***very carefully***. As a verification exercise also match it with the accompanied class diagram.

The "`GameWindow`" class opens up the game window, sets up a canvas, hooks up keyboard keys, sets up game loop and one second timer.

If you run the python file and press '`a`' or '`d`' keys, you should see appropriate messages being displayed on the screen.

Basic starter code has been written for you. You probably don't have to modify this class.

The `Game` class is the meat of the game. It draws all of the game objects and manages game logic. It is also responsible for catching key press events.

The “`Update`” method is responsible for all of the game logic. You can add your logic, collision detection between objects, updating game object states etc here.

The “`Draw`” method clears the screen and draws all of the game objects.

TIP: The order in which you draw objects matters! If you draw everything first and draw the background at the end then you won’t see anything because the background will cover-up everything.

2.1 Drawing the Player *[5 marks]*

Okay, let’s partially implement the player. Open up the `Breakout.py` file and go to the `Player` class. Since player is a moving object, it needs to store velocity apart from just the position. It also needs to store reference to the game class object and also image. Add the following code to the `__init__` method of the `Player` class:

```
class Player(GameObject):
    def __init__(self, game, pos, vel):
        #############################
        #   INSERT YOUR CODE HERE
        GameObject.__init__(self, pos)            # Call the super class constructor
        self.velocity = Vector(vel[0], vel[1])    # add velocity
        self.game = game                          # reference to the game object
        self.img = PhotoImage(file="assets/paddleBlu.gif")  # player image
        #############################
```

Now let’s write code to draw the player on the screen. Add the following lines inside the `Draw` method of the `Player` class:

```
class Player(GameObject):
    ....
    def Draw(self):
        #############################
        #   INSERT YOUR CODE HERE
        Game.canvas.create_image(self.position.x,self.position.y, image=self.img)
        #############################
```

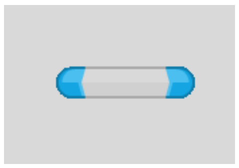
Remember that the `Game` has a class attribute “canvas” which we are accessing here to draw the player image.This method draws the player image on the screen according to the current position of the player.

We just need to do one more thing before the player image pops up on the screen. We need to add the player game object to the list of game objects in the game class. Go to the `Game` class. Add the following code to its `__init__` method:

```
class Game:
    canvas = None
    def __init__(self, canvas):
    Game.canvas = canvas           # Save canvas for future use
    self.gameObjects = []          # A list of ALL game objects in the game

    #############################
    #   INSERT YOUR CODE HERE
    self.player = Player(self, (300, 200), (0,0))      # Our Hero
    self.gameObjects.append(self.player)
    #############################
```

Run the code and you should see the player paddle on the screen (obviously you need to re-position it yourself):



Now let's move this paddle when the '`d`' key is pressed. Go to the `Game` class and add the following code in the `RightKeyPressed` method (again you need to experiment with the numbers here):

```
def RightKeyPressed(self):
    #############################
    #   INSERT YOUR CODE HERE
    self.player.position.x += 6
    #############################
```

The player paddle should move when you press the '`d`' key.

2.2 Drawing the Game Objects *[5+5+25+10 marks]*

As we did with the player object, implement background, ball, all of the bricks and all of the power-ups respective `Draw` methods. Don't forget to draw the labels as mentioned in the first part of the project. Run your game to verify that everything is drawing properly.

The game is pretty boring because nothing moves, maybe except for the player paddle, but that's OK for now.

Remember that drawing order matters !!

2.3 Moving the Ball *[10+10 marks]*

Since the ball is moving, it has a velocity attribute. You need to create the ball with a non-zero velocity and then add it to the list of game objects, just like the player. If the ball collides with any of the sides, it should deflect itself so that it remains inside the visible game area. If the ball hits the bottom you need to decrease one life of the player.

2.4 Collision Detection *[10+40 marks]*

- The paddle should not leave the game boundaries
- Implement collision between the ball and the player paddle (deflecting the ball appropriately). Try to write generic code.
- You can use the above code to detect collision between the ball and the bricks.
- Collision between power-ups and player should be very similar to ball/player or ball/bricks. You could potentially use the same code here.

You can write collision code anywhere. But you'll probably want to check the actual collision inside the `Game`'s `Update` method.

2.5 Power-ups *[30 marks]*

Implement power-ups. As mentioned in part 1 a power-up is randomly spawned after 20 seconds and its effects remain active for 5 seconds.

2.6 Exploding bricks *[20 marks]*

Each game will not have more than 3 exploding red bricks randomly placed within the bricks. Kindly refer to part 1 document for detailed rules regarding this brick.

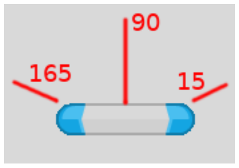
2.7 Win / Lose Conditions *[10 marks]*

Implement win / lose conditions. Update the labels accordingly. You'll probably add this logic in the `Game`'s `Update` method.

2.8 Bonus *[50 marks]*

2.8.1 Variable Ball Deflection *[20 marks]*

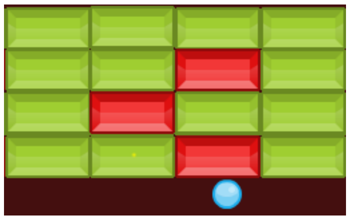
Right now when the ball hits the player paddle it is deflected with an equal angle. Change this so that the ball's angle of deflection is dependent on the position on the player paddle where the ball hits. For example if the ball hits in the exact dead center of the paddle, ball's angle of reflection will be 90 degrees. If the ball hits the edges then the reflection angles will be 15 and 165 degrees. If it hits anywhere in-between these then the angle of reflection will vary between these angles:



Hint: Treat this paddle as a rectangle with sharp corners.

2.8.2 Recursive Brick Explosion *[30 marks]*

As the ball hits any of the red bricks, the brick explodes taking down all of its (possibly 8) immediate neighbors. You must make sure that if any other red brick is in the neighborhood then it should also explode taking down its own set of immediate neighbors as well. This process should recursively continue until there are no exploding bricks in the immediate vicinity. As an example if the entire grid is composed of exploding bricks only then destroying only one of them should destroy all of them.



Submission Instructions

Compress your source file `Breakout.py` (and maybe if you are using any extra python source files) in a zip file named “`rollnumber_SECTION_proj2.zip`”. For example if your roll number is `13-10883` and section is A, then you will compress your assignment as `1310883_A_proj2.zip`. Email your assignment to fakhir.fcc@gmail.com. The subject of your email must be **“P2 SEC A Project2”**.

If you don’t follow this format then you will lose marks.

Good Luck !!!