Brick-Breaker-Terminal-Based-Game

By Ayush Sharma - 2019101004

Overview

A classic arcade Brick Breaker terminal based game written in Python3. The player will be using a paddle with a bouncing ball to smash a wall of bricks scores! The objective of the game is to break all the bricks as fast as possible. You lose a life when the ball touches the ground below the paddle.

```
* The code is modular and extensible to new features.
* Concepts of object oriented programming is present in the code
* The game is a basic simulator of brick breaker
* Libraries(external) used are colorama.
* Note: The development has been done on Ubuntu 18.04 OS.
* Note: For better experience use builtin terminal to play.
```

GUIDE

Reqiurements

- Python3
- Colorama

For mac:

```
brew cask update
sudo brew cask install python3
pip install colorama
```

For Linux:

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

Run the following code to start the game.

```
python3 main.py
```

Note: Input file i.e. `input.py` has been made & checked only for UBUNTU OS.

GAME RULES

- Player will have total `3` lives.
- One life is lost when ball goes below the paddle i.e. bottom of the frame
- Increment of `+5` will be made once a brick got broke completely
- Timer start from the start of the game, with no time limit
- Further the ball hits from the center, more the deflection in that direction.
- Once all lives are lost, game will get ended.
- Once all bricks are broken, stage up will be done with new brick layout.
- Maximum stage a player can reach is `3`

FUNCTIONALITIES

Controls

- Press `Spacebar` to release the ball
- Press `a` or `A` to move paddle left
- Press `d` or `D` to move paddle right
- Press `p` or `P` to change your Ball
- To quit, press `q`

Features

- Total `5` different types of bricks. Three breakable, One Unbreakable & One Chain reaction initializer.
- Chain reaction initializer brick on breaking with the ball would explode resulting in the destruction of all the bricks adjacent to it(diagonally, vertically and horizontally)
- Total `7` ball shapes
- Implementation of different level
- Colors for Objects
- Well commented code.
- Some degree of randomness in forming Brick-Layout for a stage.

Description of Classes Created

Status: The Status class has been made for storing status information like lives remaining, score, time, stage during the game.

Manager: The Manager class will combine all the other class & their functionalities to start game.

Frame: The Frame creates a board(basically matrix) of width `150 unit` & height `45 unit` for gameplay, with boundaries, walls and empty spaces. It also comprises of a `display` function to print that matrix on terminal.

Paddle: The Paddle class as the name suggest is made of all properties & methods suitable for paddle like horizontal movement and it's shape change.

Ball: The Ball class as thge name suggest is for the ball. It comprises all the functionalities & attributes that ball should posses like collision detection, it's shape, movement on the 2-D matrix & stickyness.

SingleBrick: The class SingleBrick is parent class for all the bricks in the game. It renders the brick on the matrix(print on terminal) & have break & remove brick function.

OneUnitBrick,TwoUnitBrick,ThreeUnitBrick: Inherited from SingleBrick class, these are special bricks with strength of `1 unit`,`2 unit` & `3 unit` respectively.

UnbrekableBrick: Inherited from SignleBrick class, these are unbrekable brick other than case when come under chain reaction.

ExplodingBrick: Inherited from SingleBrick class, special type of bricks that inhibits chain reaction. On colliding with ball it gets destroys along with other that are adjacent to it.

BrickLayout: Parent class for different stages that are LayoutStage1, LayoutStage2 & so on class. It posses logic for generating brick-layout.

Used OOPs Concepts Examples

Inheritance: Inheritance allows us to define a class that inherits all the methods and properties from another class. A base class `SingleBrick` has been declared from which different brick types are inherited.

```
class SingleBrick:
    '''
    class for single brick
    '''
    def __init__(self,point: Point,frame: Frame):
        '''
        constructor of a single brick
        '''
        self.break_brick_time = 1
        self.point = point
        self.dimension = Dimension(BRICKWIDTH,BRICKHEIGHT)
        self.shape = self.initial_shape(Back.WHITE,Style.DIM)
        self.frame = frame
    ...
```

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

```
class SingleBrick:
    ...

    def break_brick(self):
        '''
        This will make bricks break basically clear the frame.
        '''
        if self.break_brick_time>0:
            self.break_brick_time -= 1
            self.frame.clear_frame_area(self.point,self.dimension)
            self.frame.status.add_score(BASICSCOREINCREMENT)
    ...

class TwoUnitBrick(OneUnitBrick):
    ...

    def break_brick(self):
        '''
        This will make bricks break basically clear the frame.
        '''
```

```
        if self.break_brick_time==2:
            self.break_brick_time -= 1
            self.shape = self.initial_shape(Back.WHITE,Style.DIM)
            self.draw()
        elif self.break_brick_time==1:
            self.break_brick_time -= 1
            self.frame.clear_frame_area(self.point,self.dimension)
            self.frame.status.add_score(BASICSCOREINCREMENT)
    ...
```

Encapsulation The idea of wrapping data and the methods that work on data within one unit. It prevents accidental modification of data. Implemented many classes and objects for the same. Decribed in above section.

Abstraction Abstraction means hiding the complexity and only showing the essential features of the object & showing away inner details from the end user.

```
class Ball:
    ...
    def move_with_paddle(self):
        '''
        Logic for moving ball with paddle horizontally when stick.
        '''
        if not self.stick:
            return False

        new_point = Point(
            self.paddle.point.x + self.paddle_offset,
            self.point.y
        )
        self.re_draw(new_point,self.shape,self.dimension)
        return True
    ...
```

.move_with_paddle() is an abstraction

```
class Status:
    ...
    def stage_up(self):
        '''
        Staging up the player
        '''
```

```
            self._stage += 1
            if self._stage > MAXSTAGE:
                show_result(self.ret_status())
            self.start_game(1)
    ...
```

.stage_up() is an abstraction