DASS Assignment 4

Team Number: 10
Team Members: Yash Bhatia(2020101007), George Paul(2021121006), Anantha Shayana Reddy(2020101028), Praneeth Varma(2020101040)

Contribution

UML, Class Responsibilities , Updated design- George and Praneeth
Code Smells, Bugs,Proposed Changes - Yash and Ananth

Overview

A terminal based game similar to brick breaker.
Features:
Extensive use of Object Oriented Programming.

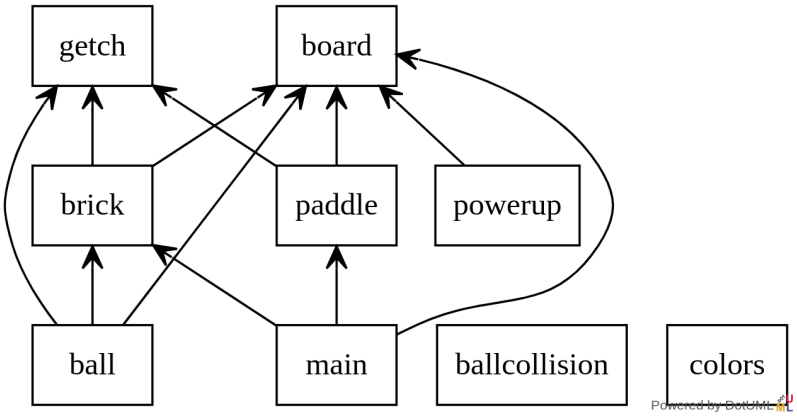
- Encapsulation: Functions like ball_movement(), draw_brick() are included within the class.
- Polymorphism: Overriding polymorphism is used for the function assign_color().
- Inheritance: Bricks of different types are inherited from the brick class.

- Abstraction: There are various functions like createball(), createpaddle(), which are called through the object of the class.

Original Design Analysis

The original design includes classes for the paddle, ball, ball collisions and bricks mainly. The classes serve their purposes of keeping track of these things and maintaining them. The original design is a little complex since it relies a lot on hard-coded values that cannot be changed in the future for, say, game design purposes. Imports are all given in headers.py and imported in all the other files. The screen is rendered every time there is input or whenever a certain timeout is reached without input.

UML class diagrams



Ball
flag : int lives score x xv y yv
collision() createball() explodable() get_distance(x, y) set_x(x) set_xv(x) set_y(y) set_yv(y) show_x() show_xv() show_y() show_yv()

Ballcollision
lives score x xv y yv
decreaselives() get_lives() hit_paddle(xv, yv, xv1, yv1, flag) hit_wall(xv, yv, xv1, yv1) move_ball(xv, yv, xv1, yv1) moveball()

Board
board
printboard()

Brick
cols : int power : list rows : int
assign_color() black_brick(y, x) cyan_brick(y, x) green_brick(y, x) red_brick(y, x) yellow_brick(y, x)

Expand
x y yv

Exploadbrick
assign_color()

Paddle
ballx bally flag : int x y
createpaddle() expandpaddle(x, y) livesdecreased() move_paddle() set_x(x) set_y(y) show_x() show_y()

PowerN
assign_color()

Powerup
x y yv
decidepowerup() getx() gety() setx(x) sety(y)

Responsibilities of Major Classes

Class	Description
Paddle	Deals with the paddle's mechanics such as creation, movement with user input and the expansion powerup
Ball	Tracks the position and movement of the ball and checks for collisions
BallCollision	Is instantiated whenever a collision occurs, whether it's the wall, paddle or brick.
Board	Deals with rendering the board at each moment and tracking the positions of the walls.
Powerup	Powerup class has no use in the game
Expand	This is pretty much a method which should have been included in the Powerup class

Code Smells

Code Smell Number	Description
1	ballcollision.py:L15 - Method doesn't have self as a parameter
2	ballcollision.py:L61 - Method is unreadable and overcomplicated
3	ball.py:L69 - Method is unreadable and overcomplicated
4	Multiple unused variables and parameters such as:

	- ball.py:L37 - x,y - brick.py:L18 - valrand
5	Multiple functions were defined that aren't used such as: - ballcollision.py:L15 - get_lives() - powerup.py:L9 - setx(), sety(), getx(), gety()
6	powerup.py:L22 - Expand class constructor is used only as a method. Expand's data members aren't used elsewhere.

Bugs in the game

Bug Number	Description
1	The score gets frozen at certain point in every game
2	The ball at times gets stuck after it hits the left wall
3	After hitting the explodable block, there are many blocks which do not take any damage, regardless of how many times they get hit.
4	The ball always initiates its flight from a fixed position, (mid of bottom row) , regardless of the position of the paddle.
5	The speed of the ball increases if we keep a particular key pressed. Key presses shouldn't have any effect on the dynamics of the ball.
6	Constant flickering
7	The ball doesn't really bounce after it hits the top of a brick. It more or less passes through and destroys them

8	The ball dynamics are not so well designed. It seems like the ball always follows the same path.
9	When the game ends, it should show: “lives remaining : 0”. It rather shows “lives remaining: 1”

Proposed Changes

- The original design has some issues like there have been cases where there is much hardcoding like in brick.py which can be easily avoided by storing the list indices in a list and iterating through them. In the ball class there have been many unnecessary methods like there have been some methods which are just returning the public variables etc.
- There are some cases where there are multiple nested if conditions (ballcollision.py etc) which increase the complexity of the code and they can be avoided by iterating through lists.
- There has been a Class Named Expand in powerup.py which has no methods and in constructor there has been some computation now instead of creating a new Class Expand and increasing the complexity of the code all the functionality can be done by just a normal function call. These are some of the changes which can be implemented to decrease the complexity of the code.

UML Diagram for the Updated design