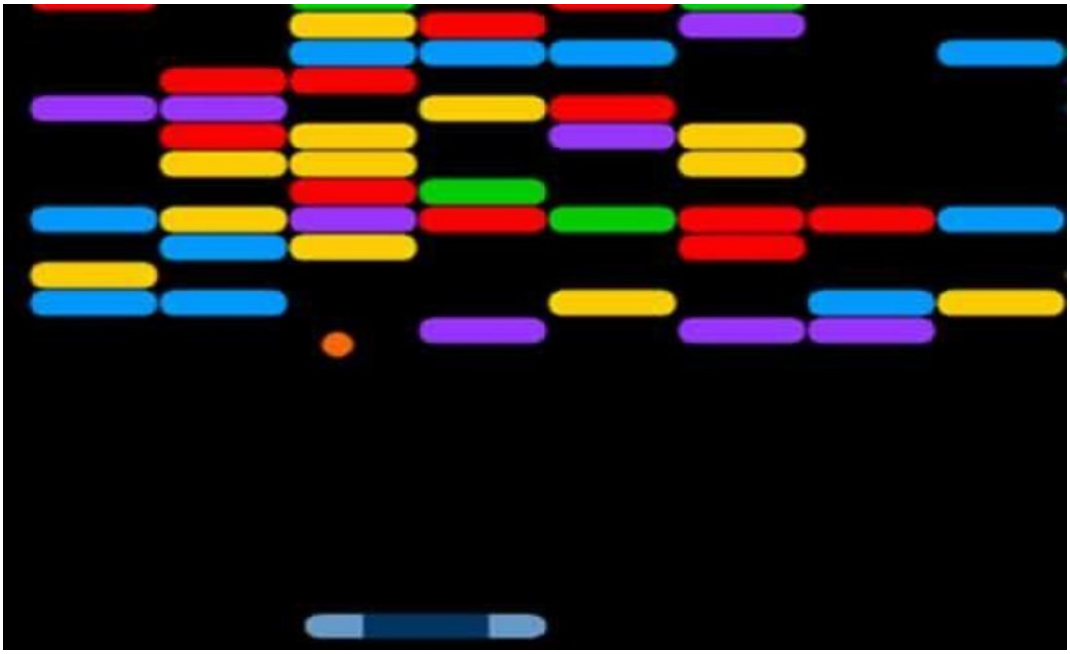
Python Project

Smokin' Ball



By:

Shivam Kumar Singh (IMT2016114)

Nishant Malpani (IMT2016095)

Introduction:

This document will fill the purpose of defining and explaining the basic construction and designing of the game '*Smokin' Ball',* in layman's terms. This game was implemented using PyGame, a cross-platfrom library designed to bring ease in writing multimedia software, such as games, in Python. Coming over to the architecture of the game, we classify it into: 'Visual features seen by the user' (*Specifications*) and 'The explanation of the underlying technical code' (*Design*).

History:

A quick, imposing history is exactly what this nostalgic game, '*Smokin' Ball'*, deserves. *'Breakout'*, '*Arkanoid'*, '*DX Ball'*, and just about every "paddle & ball vs. blocks" videogame you can name, have roots that go back as early as 1967, when Ralph Baer designed the '*Magnavox Odyssey*' game system and the paddle controller. One of the seminal games for this system included a "paddle & ball" game mechanic, making it the great-granddaddy of the *Breakout* genre. Years later, Nolan Bushnell and Steve Bristow, along with Steve Jobs and Steve Wozniak took "paddle & ball" game play a step further when they designed and developed '*Breakout'*.

Why play this game?

This game floods our brain with the reminiscence of our childhood and those countless hours in front of the white screen. So, let's dive into deep nostalgia remembering the good old days and enjoy this simple, yet fun, game!

Specifications:

Overview:

'*Smokin' Ball'* is a plain, smooth game with a solid paddle (slider), one ball and an array of bricks arranged in a complex manner waiting to be destroyed by the fiery ball. The movement of the paddle enables the user to prevent the ball from leaving the bottom part of the screen.

Visual features:

After the code is compiled and run, the game window is opened. The user can see the window titled as '*Smokin' Ball'* with a sweet little game icon.

The interactive Start Screen is displayed on the window with buttons ready to be clicked. The user sees the "Play", "Guidance" and "Exit" buttons which function exactly as they sound.

Upon clicking on the "Guidance" button, a set of instructions are advertised. The instructions are written in a basic way, solely to help an amateur. On the displayed screen, the user sees a button titled "Go Back", which brings the user back to the Start Screen.

Upon clicking on the "Play" button, the user is brought to the game arena. The user sees a slider with a ball stuck to it, waiting to be released by the 'Spacebar', as indicated by the message on the screen. Once the 'Spacebar' is pressed, the game begins with the "Score" initially set to zero and "Lives" set to three. The destruction of one brick results into the user being awarded five points. Also, if the ball touches the bottom of the screen, the "Live(s)" is decremented by one. Want to receive a call or want to have a quick bite of your cheese sandwich? With the 'Pause' feature, which gets activated upon pressing the button "P", the game pauses and a message is showcased which enables the user to continue by pressing "C" and to quit by pressing "Q". An encouraging message, along with "Game Over", is exhibited when the user loses all his lives. After completing the game, a complimentary messaged is displayed.

Design:

The code is written keeping in mind the various topics taught in the Python class, which hopefully has resulted into a robust and efficient one.

Concepts Used:

- Functions
- Object Oriented Programming (A paradigm)
- Importing Modules
- Defining Class
- Methods and Attributes
- Sprites
- Event Handling
- Collision Detection
- Movement of Objects.

Modules used:

A module is a Python object with attributes that you can bind and reference. It allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module can define functions, classes and variables. The modules used by '*Smokin' Ball'* are:

1) 'sys': This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter. The use of this module in 'Smokin' Ball' is to exit the game at any moment pressing a specific key.
2) 'pygame': This module paves way for the Python multimedia library and contains many in-built functions and classes to implement a game with great ease.
3) 'time': This module deals with the time related queries. Although, the use of this module in 'Smokin' Ball' is to control the Frames Per Second (FPS).
4) 'constants': In this module, all the constants including the RGB values of the colours, the size of the display screen, font sizes, etc. are defined.
5) 'objects': This module contains multiple classes namely: 'text_objects', 'game_objects', 'bricks' and 'buttons'. The class 'text_objects' is created to easily appoint font size, fonts, display location, colour etc. to the messages to be displayed for the user. The class 'game_objects' is created to smoothly define an object's (such as the paddle or the ball) height, width, display location, etc. 'bricks' class, again, does the same job as that of the previous class but only to a specific object, the brick. 'buttons' class helps in implementing rectangles, with attributes like colour, height, width, etc., as buttons that crave for the user's input and the act as per their tasks.

The central file:

This Python file is the one which imports all the above modules. First, the initialisation of variables like 'score', 'variables', 'FPS', 'slider speed', 'ball speed' etc. is done and then the sprites are called. Sprites are nothing but images on the display screen. The sprites are given initial display location. Now, the control flow enters the main game loop which features functions like 'detectCollisions', 'game_intro', 'buttons', 'pause', 'message_to_screen', 'events', 'score', etc. 'detectCollisions', as it sounds, helps in encountering collisions between the slider and the ball. 'message_to_screen' helps in "printing" the message on the display screen with the help of 'text_objects' class. 'events' function handles the keyboard and the mouse events by specifying tasks for any specific key pressed (like mouse clicking on a button, pressing the 'left' arrow key to move the slider). A 'Start Screen' is fashioned and the buttons are added to locations as declared by the function 'buttons'. 'pause' function helps in achieving the "Pause" feature.

Ergo, all the functions and the methods integrate to make up a sweet, little fun game!

References:

The New Boston, PyGame/docs, StackOverflow.