# BRICK BREAKER PROJECT

MINOR PROJECT REPORT

By

**DIVJOT SINGH MANCHANDA (RA221107010131)**

**VEDASHREE E (RA221107010088)**

**ARPIT KUMAR (RA221107010079)**

**VARSHITH SUGGU (RA221107010100)**

Under the guidance of   
**Dr. Tamilmani G – Assistant Professor [CINTEL]***In partial fulfilment for the Course*

of

**21CSC203P – ADVANCED PROGRAMMING PRACTICE**

In **Artificial Intelligence – B Section**



**FACULTY OF ENGINEERING AND TECHNOLOGY**

**SCHOOL OF COMPUTING**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR**

**NOVEMBER 2023**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**(Under Section 3 of UGC Act, 1956)**

**BONAFIDE CERTIFICATE**

Certified that this minor project report for the course **21CSC203P** **ADVANCED PROGRAMMING PRACTICE** entitled in "**Brick Breaker Game**" is the bonafide work of **DIVJOT SINGH MANCHANDA (RA221107010131), VEDASHREE E (RA221107010088), ARPIT KUMAR (RA221107010079), and VARSHITH SUGGU (RA221107010100)** who carried out the work under my supervision.

# SIGNATURE SIGNATURE

# Dr. G Tamilmani Dr. Annie Uthara

# Assistant Professor Head of Department

**Computational Intelligence Computational Intelligence**

SRM Institute of Science and Technology SRM Institute of Science and Technology

Kattankulathur Kattankulathur

# ABSTRACT

The Brick Breaker Game Project is a classic arcade-style video game, offering a nostalgic yet captivating gaming experience. Players control a paddle to bounce a ball, breaking through a wall of bricks while scoring points for each successful hit. The project focuses on realistic physics simulations, multiple levels with varying challenges, and an engaging scoring system. It features a user-friendly interface with appealing graphics, sound effects, and background music. User interaction is facilitated through keyboard or touch controls. The game offers win and lose conditions, promoting player engagement and motivation. Developed by a passionate team of programmers and designers, the project aims to provide cross-platform compatibility, making it accessible to a broad audience. The Brick Breaker Game Project blends the charm of the classic game with modern gaming elements for an entertaining and immersive experience.

# ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C MUTHAMIZHCHELVAN**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy,** for his encouragement.

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V.Gopal,** for bringing out novelty in all executions.

We would like to express my heartfelt thanks to Chairperson, School of Computing **Dr. Revathi Venkataraman,** for imparting confidence to complete my course project

We wish to express my sincere thanks to **Course Audit Professors Dr. Vadivu. G , Professor, Department of Data Science and Business Systems and Dr. Sasikala. E Professor, Department of Data Science and Business Systems** and **Course Coordinators**  for their constant encouragement and support.

We are highly thankful to our my Course project Faculty **Dr. Tamilmani G, Assistant Professor, CINTEL Department,** for herassistance, timely suggestion and guidance throughout the duration of this course project.

We extend my gratitude to our **HoD Dr. Annie Uthra, CINTEL Department** and my Departmental colleagues for their Support.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

**TABLE OF CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **CHAPTER NO** | | **CONTENTS** | **PAGE NO** |
|  |  | |  |
| **1** | **INTRODUCTION**   * 1. Motivation   2. Objective   3. Problem Statement   4. Challenges | | **5** |
| **2** | **LITERATURE SURVEY** | | **6** |
| **3** | **REQUIREMENT ANALYSIS** | | **7** |
| **4** | **ARCHITECTURE & DESIGN** | | **8** |
| **5** | **IMPLEMENTATION** | | **9-15** |
| **6** | **EXPERIMENTAL RESULTS** | | **16-21** |
| **7** | **CONCLUSION** | | **22** |
| **8** | **REFERENCES** | | **23** |

1. **INTRODUCTION**

Brick Breaker is a classic arcade game where the player controls a paddle at the bottom of the screen to bounce a ball towards a wall of colored bricks. The goal of the game is to break all the bricks on the screen using the bouncing ball while preventing the ball from falling off the bottom of the screen.

The Classical Brick Breaker Game in Python is a project that pays tribute to the revered tradition of classic arcade gaming. It reignites the excitement and sentiment associated with the iconic Brick Breaker game, a title that has held a special place in the hearts of gamers for generations. With a foundation in Python programming, this project not only rekindles the nostalgia of the past but also showcases the enduring relevance and capabilities of this versatile programming language.

This Game provides the a multi-level matrix brick layout with bricks in different shades of grey depicting the different levels of hardness of the bricks, with the harder bricks providing more bricks and softer bricks giving less points, the harder bricks turn into softer bricks when hit and when the softest white bricks are hit the brick is broken and removed from the game.

The game also provides the user to choose between 3 difficulties Easy, Normal and Hard, depending on their level of experience and skills, providing exciting challenges to the user.

This game is a journey back in time, reimagining the familiar joys of Brick Breaker while leveraging modern technology to create an immersive gaming experience. It captures the essence of simplicity and challenge that has made the Brick Breaker genre timeless. The game's intuitive controls, realistic physics, and progressive levels offer an engaging and approachable experience suitable for players of all ages.

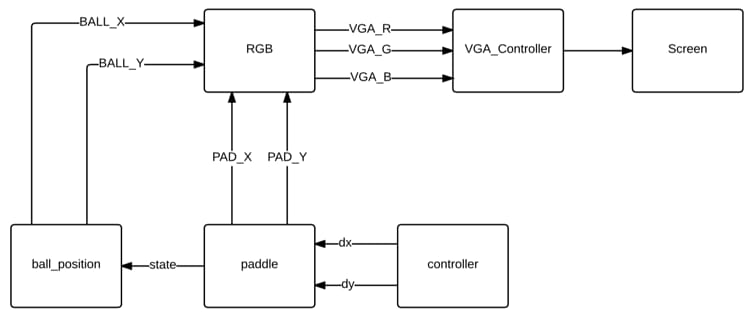
1. **LITERATURE SURVEY**

The first step is to write a code which puts the various pieces on the playing board. It probably makes sense to implement procedure to run (which drives the game) the game and that play it. The most important part of the setup consists of creating rows of bricks on the top of the game. The number of dimensions and spacing of the bricks, and the distance from the top of the window to the first line of bricks, are specified using some named constants given in class Breakout. The next step is creating a paddle. You will need to reference the paddle often. Be careful that the paddle stays completely on the board even if the mouse moves off the board. The next step is an interesting part. In order to make brick breaker or breakout into a real game, you have to be able to tell when the ball collides with another object in the window. However, the ball is not a single point. It occupies physical area, so it may collide with something on the screen even though its center does not. The easiest thing to do is typically of the simplifying assumptions made in real computer games are to check a few carefully chosen points on the outside of the ball and see whether any of those points has collided with anything. As soon as you find something at one of those points (other than the ball of course) you can declare that the ball has collided with that object

1. **REQUIREMENT ANALYSIS**
   1. **Hardware Requirements**

* Processor: Any modern processor (e.g., Intel Core i3 or equivalent)
* RAM: 2 GB or more
* Storage: At least 100 MB of free storage space
  1. **Software Requirements**
* Operating System: Windows, macOS, or Linux
* Python: Python 3.x installed
* Pygame Library: Pygame library installed (for game development)
* Code Editor: Any code editor or integrated development environment (IDE) of your choice, such as Visual Studio Code, PyCharm, or Sublime Text

1. **ARCHITECTURE/ DESIGN**

****

**Fig. Architecture Diagram**

The architectural design of the Brick Breaker game in Python adheres to the Model-View-Controller (MVC) pattern, a well-suited choice for graphical user interface (GUI) applications. In this architecture, the Model represents the game's underlying logic, encompassing features such as brick placement, ball physics, scoring, level progression, and win or loss conditions. The View serves as the user interface, rendering the game board, including the paddle, bricks, and the ball, with an emphasis on creating an engaging and visually appealing experience for players. Meanwhile, the Controller module is responsible for managing event handling, connecting user interactions (e.g., paddle movement) with the game's logic. It also oversees game-specific events, including power-up activations and level transitions, as well as the display of win status and the checking of game-over or draw conditions. This architectural approach ensures a clean separation of concerns, making it easier to maintain and expand the game's functionality while delivering an enjoyable gaming experience.

1. **IMPLEMENTATION**

import pygame

import random

# Initialize Pygame

pygame.init()

# Constants

WIDTH, HEIGHT = 800, 600 #Display Size

BALL\_SIZE = 20

PADDLE\_LIMIT = WIDTH

# Variables

paddle\_speed = 5

ball\_speed = 3

paddle\_width = 100

paddle\_height = 10

# Colors

BLACK = (0, 0, 0)

WHITE = (255, 255, 255)

GREEN = (0, 255, 0)

RED = (255, 0, 0)

GRAY = (128, 128, 128)

DGRAY = (64, 64, 64) #Dark Gray

LGRAY = (196, 196, 196) #Light Gray

CYAN = (0, 255, 255)

# Create the game window

screen = pygame.display.set\_mode((WIDTH, HEIGHT))

pygame.display.set\_caption("Brick Breaker Project")

# Game variables

play = False

score = 0

paddle = 310

#Random Ball Placement and Direction

ball\_pos = [340 + random.randint(-50,50), 350 + random.randint(-30,30)]

ball\_dir = [2\*[-1,1][random.randrange(2)], -3]

#Number of Rows and Columns

rows=5

cols=7

total\_bricks = rows\*cols

# Initialize the map (layout of bricks) - Checkered Decreasing Pattern

map = [[(i//2 + 1) if ((i + j) % 2 == 0 and i%2==1 ) else i//2 if ((i + j) % 2 == 1 and i%2==1) else i//2 - 1 if ((i + j) % 2 == 1 and i%2==0) else i//2 + 1 for j in range(cols)] for i in range(rows+2,0,-1)]

#Brick Size

brick\_width = 640 // cols

brick\_height = 40

# Pygame clock and font

clock = pygame.time.Clock()

font = pygame.font.Font(None, 36)

mode\_text = font.render("Normal Mode", True, LGRAY)

# Main game loop initialize

running = True

move\_left = False

move\_right = False

#Game Loop

while running:

for event in pygame.event.get():

#Quit Game - X button

if event.type == pygame.QUIT:

running = False

#If any Key is pressed

if event.type == pygame.KEYDOWN:

#Play/Pause - Spacebar

if event.key == pygame.K\_SPACE:

play = not play

#Move Paddle - Start

if play:

if event.key == pygame.K\_LEFT:

move\_left = True

if event.key == pygame.K\_RIGHT:

move\_right = True

# New Game

# Normal Mode - Press Enter

if event.key == pygame.K\_RETURN:

play = False

ball\_pos = [340 + random.randint(-50,50), 350 + random.randint(-30,30)]

paddle = 310

paddle\_speed = 5

paddle\_width = 100

paddle\_height = 10

ball\_speed = 3

ball\_dir = [2\*[-1,1][random.randrange(2)], -3]

score = 0

mode\_text = font.render("Normal Mode", True, LGRAY)

total\_bricks = rows\*cols

map = [[(i//2 + 1) if ((i + j) % 2 == 0 and i%2==1 ) else i//2 if ((i + j) % 2 == 1 and i%2==1) else i//2 - 1 if ((i + j) % 2 == 1 and i%2==0) else i//2 + 1 for j in range(cols)] for i in range(rows+2,0,-1)]

#Easy Mode - Press 1

if event.key == pygame.K\_1:

play = False

ball\_pos = [340 + random.randint(-50,50), 340 + random.randint(-30,30)]

ball\_speed = 3

ball\_dir = [2\*[-1,1][random.randrange(2)], -3]

paddle\_speed = 5

paddle\_width = 200

paddle\_height = 10

paddle = 310

score = 0

mode\_text = font.render("Easy Mode", True, LGRAY)

total\_bricks = rows\*cols

map = [[(i//2 + 1) if ((i + j) % 2 == 0 and i%2==1 ) else i//2 if ((i + j) % 2 == 1 and i%2==1) else i//2 - 1 if ((i + j) % 2 == 1 and i%2==0) else i//2 + 1 for j in range(cols)] for i in range(rows+2,0,-1)]

#Hard Mode - Press 2

if event.key == pygame.K\_2:

play = False

ball\_pos = [240 + random.randint(-50,50), 350 + random.randint(-30,30)]

ball\_speed = 6

ball\_dir = [4\*[-1,1][random.randrange(2)], -6]

paddle\_speed = 7

paddle\_width = 80

paddle\_height = 20

paddle = 310

score = 0

mode\_text = font.render("Hard Mode", True, LGRAY)

total\_bricks = rows\*cols

map = [[(i//2 + 1) if ((i + j) % 2 == 0 and i%2==1 ) else i//2 if ((i + j) % 2 == 1 and i%2==1) else i//2 - 1 if ((i + j) % 2 == 1 and i%2==0) else i//2 + 1 for j in range(cols)] for i in range(rows+2,0,-1)]

#Move Paddle - Stop

if event.type == pygame.KEYUP:

if event.key == pygame.K\_LEFT:

move\_left = False

if event.key == pygame.K\_RIGHT:

move\_right = False

#Paddle Movement

if move\_left and paddle - paddle\_speed > 0:

paddle -= paddle\_speed

if move\_right and paddle + paddle\_width + paddle\_speed < PADDLE\_LIMIT:

paddle += paddle\_speed

#If Game is running

if play:

ball\_pos[0] += ball\_dir[0]

ball\_pos[1] += ball\_dir[1]

#Collision with Wall

if ball\_pos[0] < 0 or ball\_pos[0] > WIDTH - BALL\_SIZE:

ball\_dir[0] = -ball\_dir[0]

if ball\_pos[1] < 0:

ball\_dir[1] = -ball\_dir[1]

#Ball out of Bound (Game Over)

if ball\_pos[1] > HEIGHT - BALL\_SIZE:

play = False

if (paddle < ball\_pos[0] < paddle + paddle\_width and HEIGHT - BALL\_SIZE - paddle\_height <= ball\_pos[1] <= HEIGHT - BALL\_SIZE + ball\_speed):

ball\_dir[1] = -ball\_dir[1]

#Brick Layout Generator

for i in range(rows):

for j in range(cols):

if map[i][j] > 0:

brick\_x = j \* brick\_width + 80

brick\_y = i \* brick\_height + 50

brick\_rect = pygame.Rect(brick\_x, brick\_y, brick\_width, brick\_height)

ball\_rect = pygame.Rect(ball\_pos[0], ball\_pos[1], BALL\_SIZE, BALL\_SIZE)

#Collision Detection

if brick\_rect.colliderect(ball\_rect):

if ball\_pos[0] + BALL\_SIZE - 2 <= brick\_x or ball\_pos[0] + 2 >= brick\_x + brick\_width:

ball\_dir[0] = -ball\_dir[0]

else:

ball\_dir[1] = -ball\_dir[1]

#Score Increment and Brick Level Down

if map[i][j]>1:

map[i][j]-=1

score +=map[i][j]\*5

else:

#Brick Removal

map[i][j] = 0

total\_bricks -= 1

score += 5

# Clear the screen

screen.fill(BLACK)

# Draw the map (bricks)

for i in range(rows):

for j in range(cols):

if map[i][j] > 0:

brick\_x = j \* brick\_width + 80

brick\_y = i \* brick\_height + 50

brick\_rect = pygame.Rect(brick\_x, brick\_y, brick\_width, brick\_height)

#Color of Bricks depending on level

color = WHITE if map[i][j] == 1 else LGRAY if map[i][j] == 2 else GRAY if map[i][j] == 3 else DGRAY

pygame.draw.rect(screen, color, brick\_rect)

pygame.draw.rect(screen, BLACK, brick\_rect, 3)

# Draw the paddle

pygame.draw.rect(screen, GREEN, (paddle, HEIGHT - paddle\_height, paddle\_width, paddle\_height))

# Draw the ball

pygame.draw.ellipse(screen, RED, (ball\_pos[0], ball\_pos[1], BALL\_SIZE+1, BALL\_SIZE+1))

# Draw the score

score\_text = font.render("Score: " + str(score), True, WHITE)

screen.blit(score\_text, (WIDTH - 150, 10))

screen.blit(mode\_text, (60, 10))

# Displays Game Guide

project\_info\_font = pygame.font.Font(None, 25)

info\_text4 = project\_info\_font.render("Press Spacebar play/pause", True, GRAY)

info\_text5 = project\_info\_font.render("Press Enter to Restart", True, GRAY)

screen.blit(info\_text4, (80, 500))

screen.blit(info\_text5, (80, 525))

# Check for win condition

if total\_bricks <= 0:

play = False

win\_text = font.render("YOU WON: " + str(score), True, CYAN)

screen.blit(win\_text, (WIDTH // 2 - 120, HEIGHT // 2 - 30))

# Check for game over

if not play and ball\_pos[1] > HEIGHT - BALL\_SIZE:

# Displaying Game Score and Restart Option

game\_over\_text = font.render("Game Over, Score: " + str(score), True, CYAN)

restart\_text = font.render("Press Enter To Restart", True, WHITE)

easy\_text = font.render("Press 1 for Easy Mode", True, LGRAY)

hard\_text = font.render("Press 2 for Hard Mode", True, LGRAY)

screen.blit(game\_over\_text, (WIDTH // 2 - 150, HEIGHT // 2 - 30))

screen.blit(restart\_text, (WIDTH // 2 - 150, HEIGHT // 2 ))

screen.blit(easy\_text, (WIDTH // 2 - 150, HEIGHT // 2 + 30))

screen.blit(hard\_text, (WIDTH // 2 - 150, HEIGHT // 2 + 60))

# Displaying Project Info

info\_text1 = project\_info\_font.render("Project by -", True, WHITE)

info\_text2 = project\_info\_font.render("Divjot, Varshit, Arpit, Vedashree", True, WHITE)

info\_text3 = project\_info\_font.render("131, 100, 079, 088", True, WHITE)

screen.blit(info\_text1, (180, 400))

screen.blit(info\_text2, (180, 425))

screen.blit(info\_text3, (180, 450))

#Next Frame

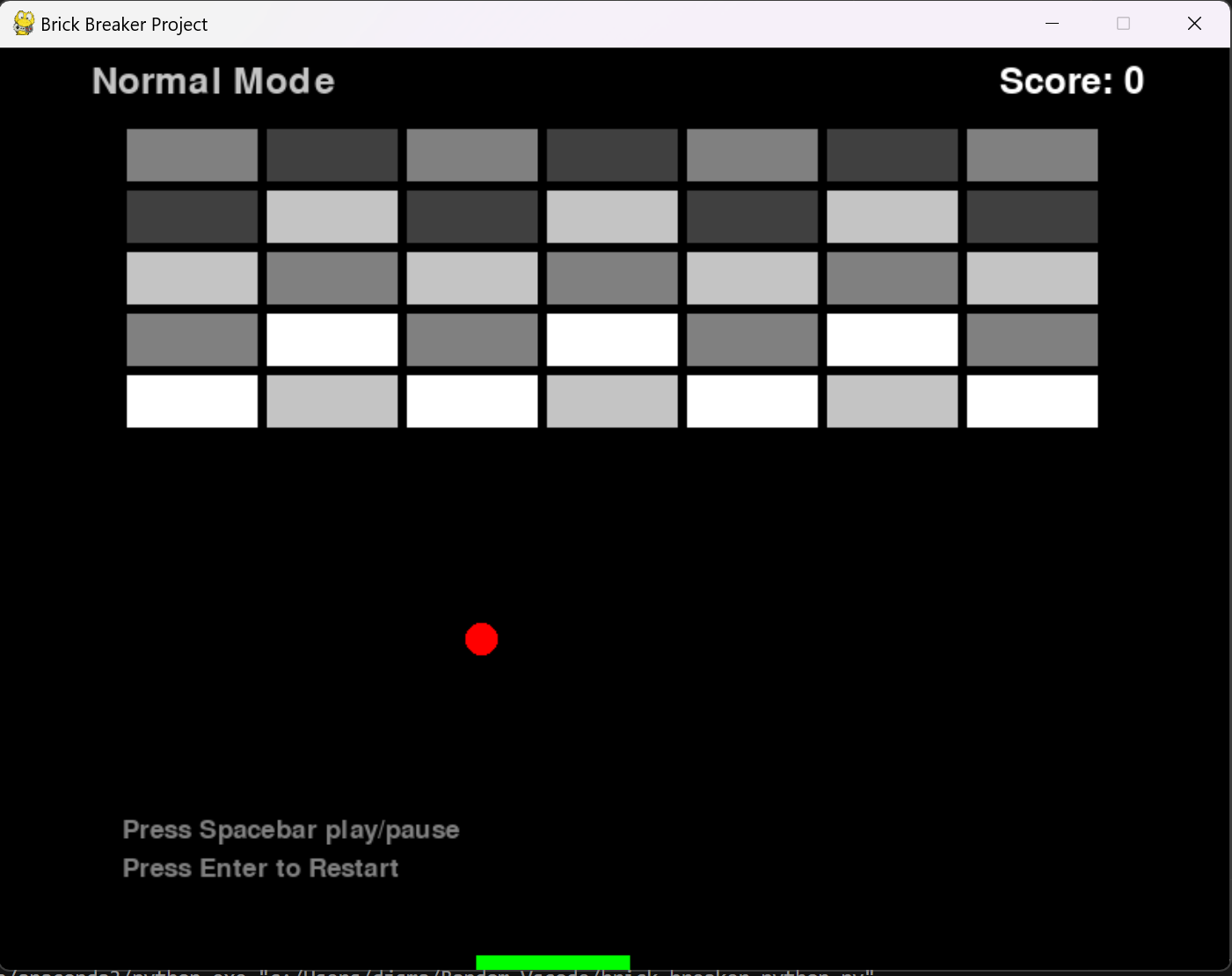
pygame.display.update()

clock.tick(60)

# End of Program

pygame.quit()

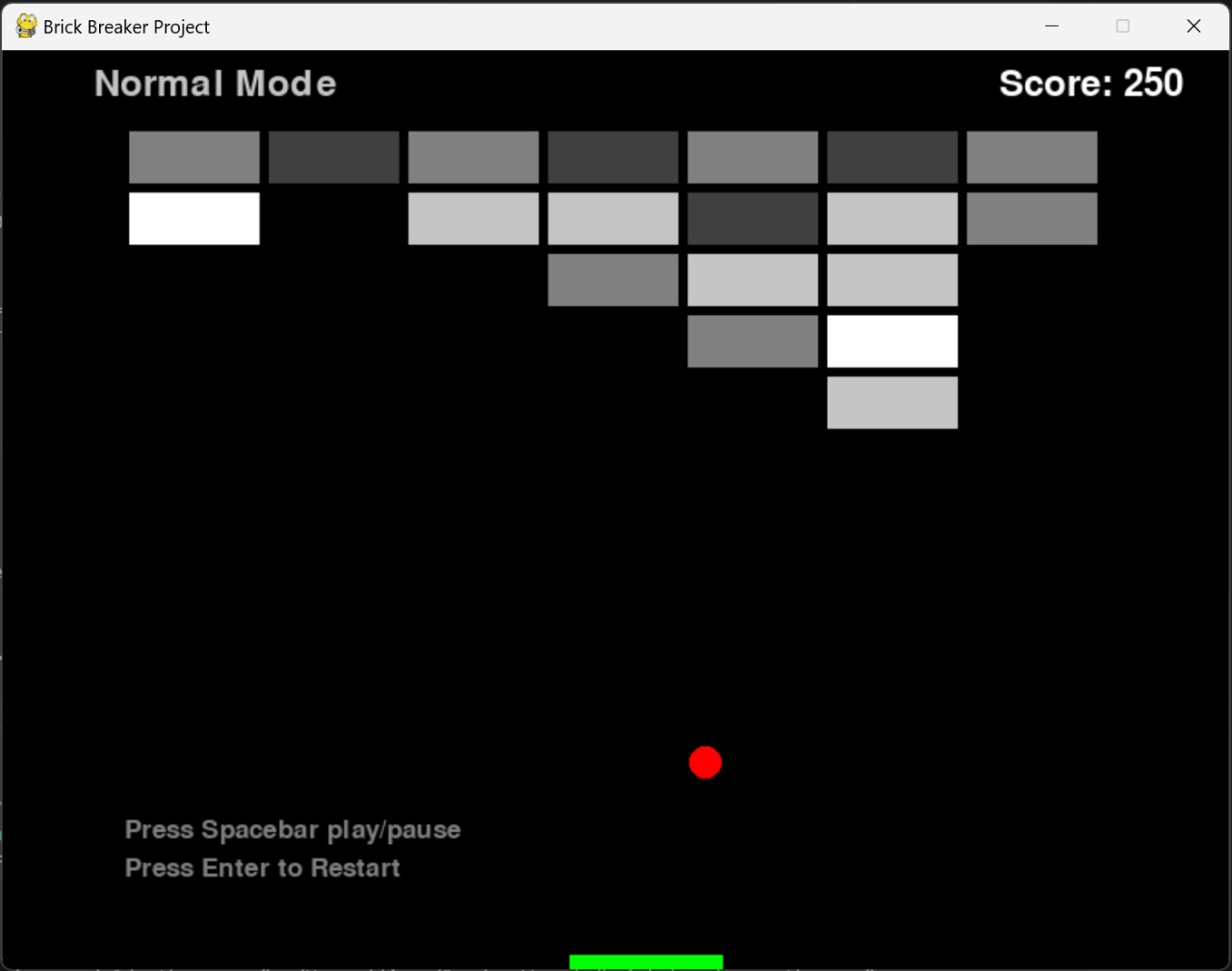
1. **EXPERIMENTAL RESULTS**
   1. **Start Screen**

****

**Fig.6.1**

Figure 6.1 presents the start screen of the Brick Breaker game, featuring a visually engaging display with white and grey blocks, a vibrant red ball, and a green paddle. The objective is simple: players must use the green paddle to control the red ball's movement, directing it to break all the blocks. This screen introduces players to the game's core mechanics and sets the stage for an exciting and visually captivating gaming experience.

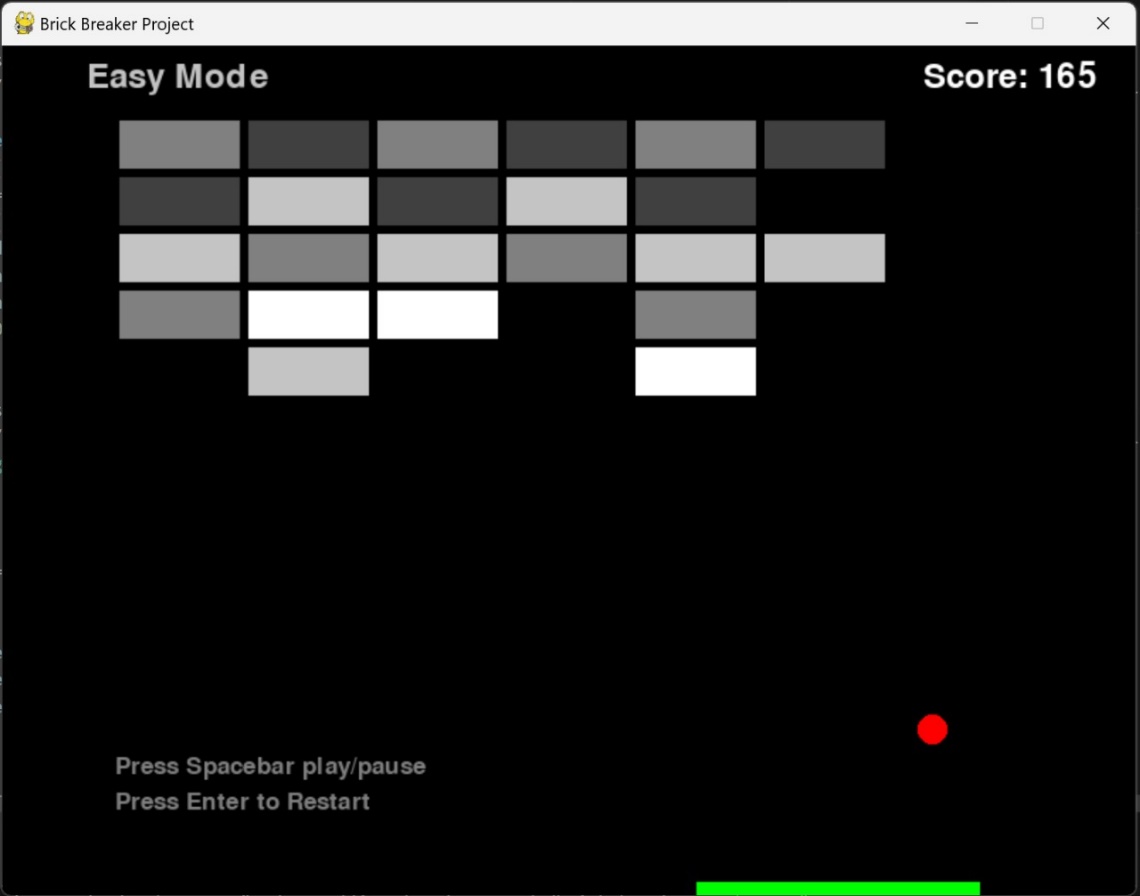
* 1. **In-game Screen – Normal Mode**

****

**Fig. 6.2**

Figure 6.2 illustrates the in-game screen during active gameplay. The game mechanics are straightforward: when the ball makes contact with a block the ball drops 1 level in strength and colour and points get added, Dark Brown Bricks give 20 points and White give 5 points, after hitting a white Brick, the Brick Breaks and Disappears. This dynamic element adds a layer of strategy and excitement to the gameplay, as players strive to maximize their score by strategically targeting different block types.

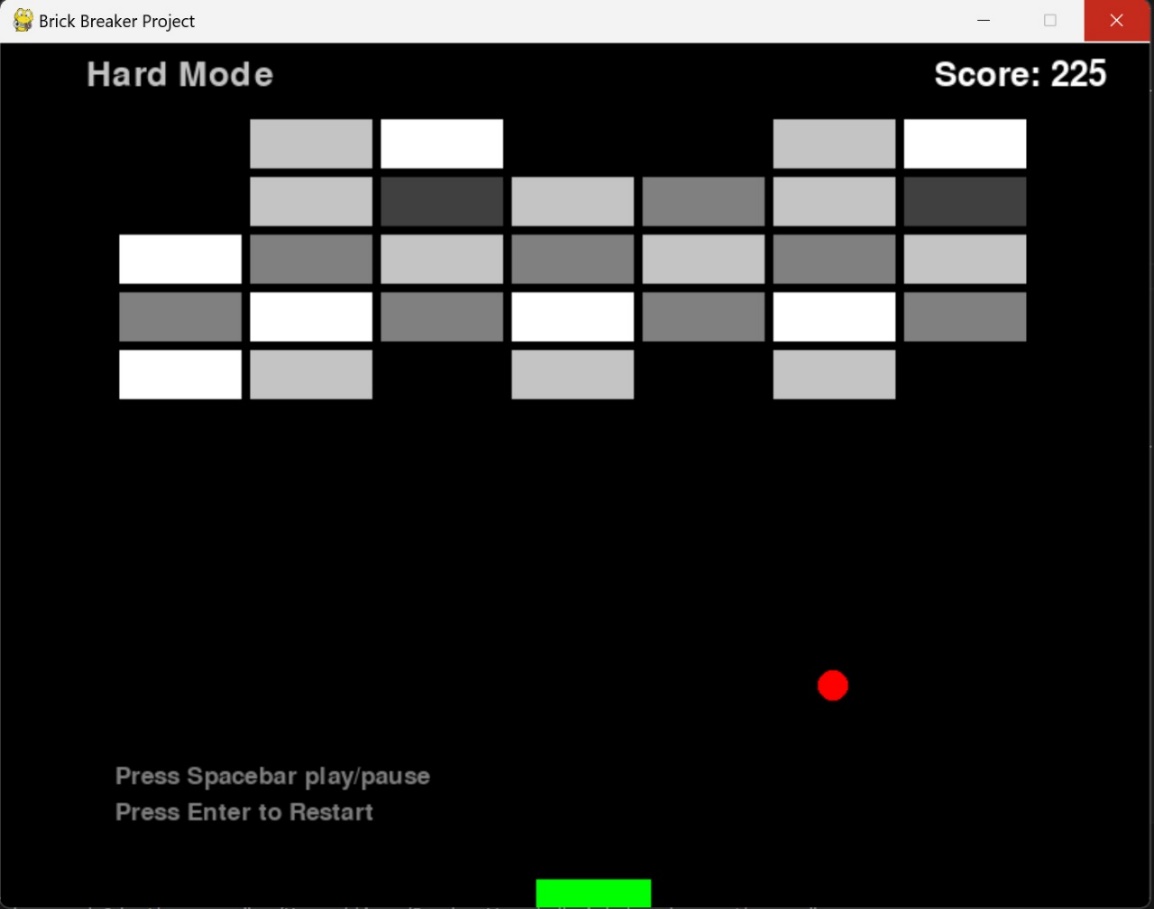
* 1. **In-game Screen – Easy Mode**

****

**Fig.6.3**

Figure 6.3 showcases the Easy Mode of the Game, in this mode the ball becomes slower and the paddle becomes wider, this is especially beneficial for beginners as it provides then with extra time and liberty to hit the ball with the paddle

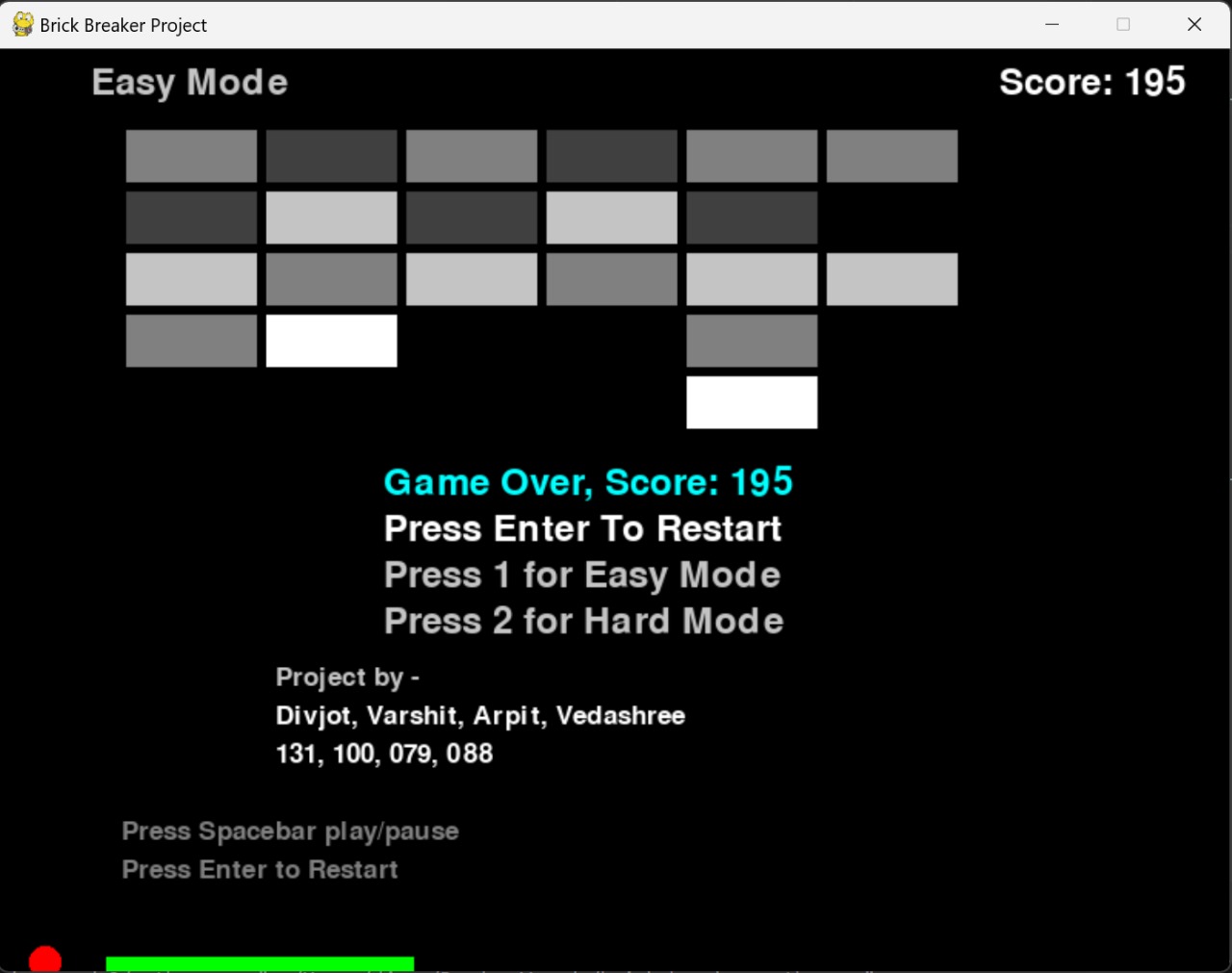
* 1. **In-game Screen – Hard Mode**

****

**Fig.6.4**

Figure 6.4 showcases the Hard Mode of the Game, in this mode the ball becomes much faster, along with that even the paddle becomes narrower, this creates an interesting challenge for experienced players as the games becomes far more difficult to play.

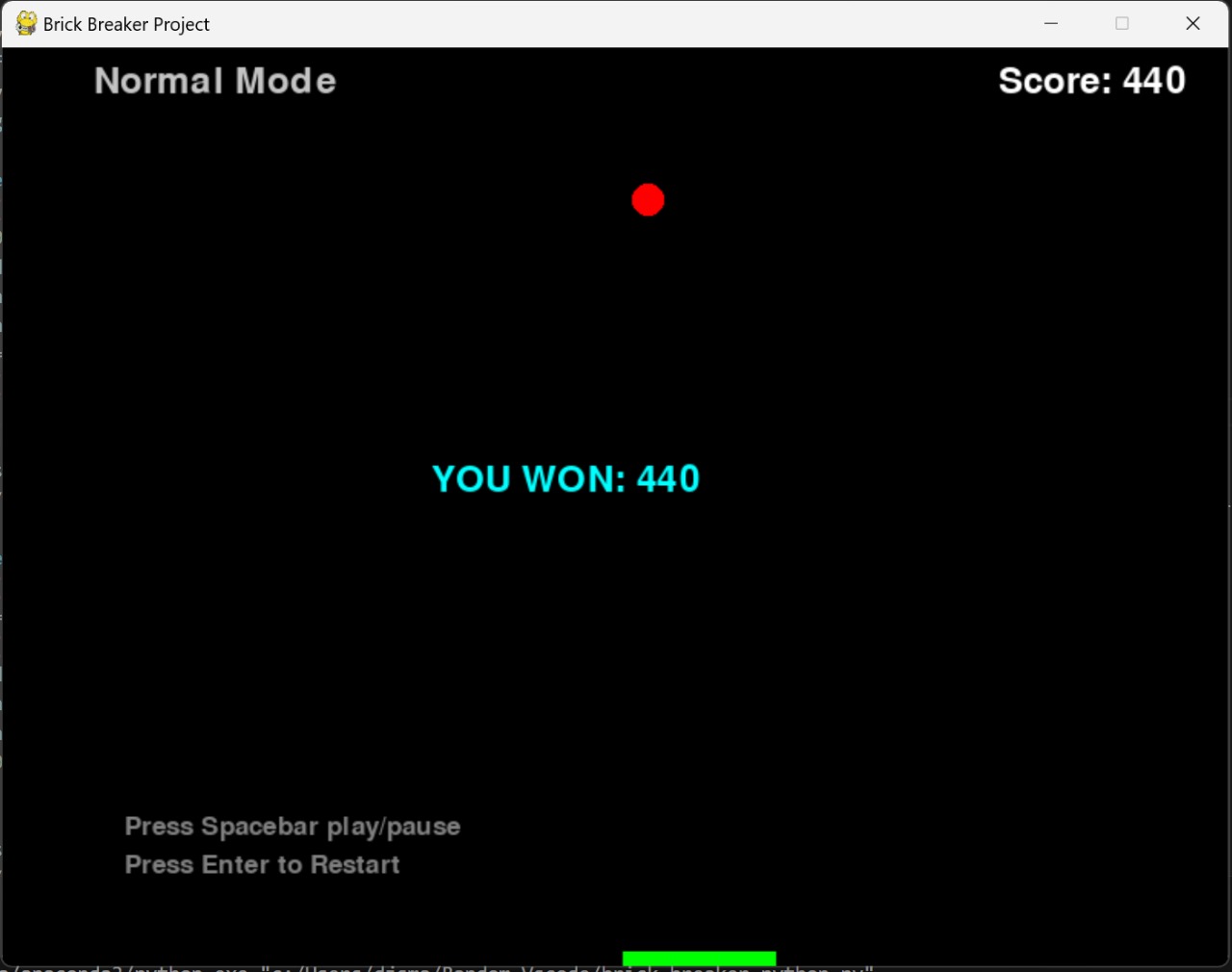
* 1. **Game Over (Lost)**

****

**Fig.6.5**

Figure 6.5 showcases the game over screen, signaling the end of the game when the ball fails to make contact with the paddle. On this screen, players are presented with their final score, and they have the option to restart the game. It serves as both the conclusion of the game and a new beginning, offering players the chance to improve their performance and aim for higher scores in subsequent attempts while giving an option to try different modes.

* 1. **Game Won**

****

**Fig.6.6**

Figure 6.6 illustrates the victorious moment in the game when the player emerges as the winner. The screen proudly displays the player's score, celebrating their achievement. This screen also provides the option to restart the game, allowing players to continue their Brick Breaker journey or aim for an even more impressive score in the next round. It encapsulates the thrill of success while inviting players to keep testing their skills.

1. **CONCLUSION**

We successfully created a Python-based Classical Brick Breaker Game as a part of our university project. This endeavour rekindled the timeless charm of the classic Brick Breaker arcade game while showcasing the enduring power of Python programming.

Our game boasts straightforward yet engaging gameplay with intuitive controls and realistic physics, appealing to a diverse audience. The inclusion of a scoring system and competitive multi-level brick elements and multiple difficulty modes adds excitement and encourages players to strive for higher scores and conquer challenging levels.

Moreover, our project's cross-platform compatibility ensures accessibility, enabling players to enjoy the game on various devices.

The dedication of our development team is evident in the polished and enjoyable gaming experience we've crafted.

This project stands as a testament to the versatility of Python in bringing classic gaming experiences to life, bridging the gap between the past and the present.

1. **REFERENCES**

* Columbia University brick breaker game http://www.cs.columbia.edu/~sedwards/classes/2015/4840/designs/breakout.pdf&ved=2ahUKEwicz8Gn9JiCAxUC1TgGHarTDCsQFnoECAEQAQ&usg=AOvVaw0ROVvmX-pmjbScxsU\_06sT
* <https://www.cs.columbia.edu/~sedwards/classes/2015/4840/designs/breakout.pdf>
* Brick breaker - Global journal of Engineering and science http://www.gjesr.com/Issues%2520PDF/ICITAIC-2019/24.pdf
* Brick breaker game design

Columbia University

* <http://www.cs.columbia.edu/~sedwards/classes/2019/4840-spring/designs/BrickBreaker.pdf>
* Brick breaker final report

<http://web.mit.edu/6.111/www/f2013/projects/jabbott_Project_Final_Report.pdf>

* https://en.m.wikipedia.org/wiki/Brick\_Breaker