جامعة طنطا

كلية الهندسة

بحث / مشروع بحثي

مقدم من

موضوع البحث : Breakout game

القسم : هندسة الحاسبات والتحكم الآلي

الفرقة : الثانية

اسم المقرر : رسومات الحاسب

كود المقرر : CCE2211

أستاذ المقرر : د. محمد عيطة

اسم الطالب	الرقم القومي	البريد الإلكتروني
أحمد محمد صالح السرسي	29901021603755	ahmed.30934270@f -eng.tanta.edu.eg
احمد نبيل محمد النواوي	29909291602751	ahmed.30934516@f-eng.tanta.edu.eg
أسماء عبد العزيز محمد السعيد	29909091601688	asmaa.30934034@f-eng.tanta.edu.eg
أمل أحمد محمد الأجدر	29807231601988	aml_EN30895197@f-eng.tanta.edu.eg

نتيجة التقييم	مقبول		غير مقبول	

Game Basics

The idea of a ***Breakout*** game is that a layer of bricks lines the top third of the screen and the goal is to destroy them all. A ball moves straight around the screen, bouncing off the top and two sides of the screen. When a brick is hit, the ball bounces back and the brick is destroyed. The player loses a turn when the ball touches the bottom of the screen; to prevent this from happening, the player has a horizontally movable paddle to bounce the ball upward, keeping it in play. The OpenGL window contains two counters:

1. *Life counter*: it decreases only if the ball hits the bottom wall.
2. *Score counter*: it increases only if the ball hits the bricks.

Getting started

The software used to implement the game is the programming language *Python* and the two libraries *OpenGL* and *Pygame*.

Implementation Steps

Step.1 Drawing the scene

• Frame

Drawing the frame is done by drawing line strips for sides of the window. Each strip has length and color. They are specified by passing vertices of one point of the line and the length of it using the parameters *firstPoint* and num to ***drawFrameStrip( )*** function. The *firstPoint* parameter is a set of three values (X of point on left side, Y of points on both sides as they are symmetric, X of point in right side). Finally, ***drawFrameStrip( )*** function is called with all parameters of all strips of frame and colors in ***drawFrame( )*** function.

```
def drawFrameStrip(firstPoint , num ):
    glBegin(GL_LINES)
    glVertex(firstPoint[0], firstPoint[1])
    glVertex(firstPoint[0], firstPoint[1]+num*brickShiftY)
    glVertex(firstPoint[2], firstPoint[1])
    glVertex(firstPoint[2], firstPoint[1]+num*brickShiftY)
    glEnd()
```

• Bricks

A class called ***Brick( )*** is declared with four variables stores the vertex data of each corner of brick, and a variable active which determines if the brick will be rendered or not as True or False value. In the ***generateLevel( )*** function, the concept of generating the bricks is specified and bricks are drawn using ***drawQuad( )*** function. All bricks generated are stored in a list called Bricks.

• **Paddle**

A class called ***Paddle( )*** is declared with variables that specifies width, height, initial position of the paddle and vertices of the four corners. The paddle is drawn by passing the corners of it to ***drawQuad( )*** function. Its motion is controlled by mouse in X direction.

```
class Paddle():
    width, height = 100, 20
    position =[0, 70]
    bottomLeft = [position[0]- width/2, position[1]+Display.frustumHeight[0]+0.5]
    bottomRight =[position[0]+ width/2, position[1]+Display.frustumHeight[0]+0.5]
    topRight =[position[0]+ width/2,
               position[1]+Display.frustumHeight[0]+height+0.5]
    topLeft = [position[0]- width/2,
               position[1]+Display.frustumHeight[0]+height+0.5]
```

• **Ball**

The ball is represented as a solid sphere and other objects in the scene deal with it as a box, its values is stored in ***boundingBox*** list, and it will move randomly in the OpenGL window.

```
class Ball():
    radius = 15
    position = [100, 40] # initial position of ball
    boundingBox = [position[0]-radius, position[1]+radius, position[0]+radius,
                   position[1]-radius] # Left, Top, Right, Bottom
```

Step.2 Coloring the Scene To add fun to the game, all objects takes the color of the window frame strip they are in its range. This is done by passing the Y value of each object to ***changeColor( )*** function and set its color after checking its Y value is in which frame strip range.

```
def changeColor(y):
    global color
    if y >= Display.frustumHeight[0]+ firstPointY+ 4*brickShiftY and y < Display.frustumHeight[0]+ firstPointY+
        8*brickShiftY:
        color = (.9647,.4314,0)       #   ORANGE
    elif y >= Display.frustumHeight[0]+firstPointY and y <= Display.frustumHeight[0]+firstPointY+ 4*brickShiftY:
        color = (0,1,0)               # GREEN
    elif y >= Display.frustumHeight[0]+firstPointY- 4*brickShiftY and y <= Display.frustumHeight[0]+firstPointY:
        color  = (1,1,0)                #YELLOW
    elif y >=  Display.frustumHeight[0] and y <=  Display.frustumHeight[0]+ 14.738* brickShiftY:
        color = (1,1,1)                   #WHITE
    elif (y >= Display.frustumHeight[0]+firstPointY +8* brickShiftY
        and y <= Display.frustumHeight[0]+firstPointY + 12* brickShiftY)
        or (y <= Display.frustumHeight[0]+firstPointY - 15.8* brickShiftY):
        color = (0,0,1)           ## BLUE
```

Step.3 Drawing Text

To implement the text we created a ***drawText( )*** function and used it for drawing game states on screen such as lives, score, high score and other texts in menu. This function accepts string to be drawn, its position, its size and thickness as string, posX, posY, scaleX, scaleY and lineWidth parameters respectively. To draw a character in window, we used a function in OpenGL called ***glutStrokeCharacter( ).***

```
def drawText(string, posX, posY, scaleX, scaleY, lineWidth):
    glLineWidth(lineWidth)
    glPushMatrix()
    glLoadIdentity()
    glTranslate(posX, posY, 0)
    glScale(scaleX, scaleY, 3)
    string = string.encode()  # conversion from Unicode string to byte string
    for c in string:
        glutStrokeCharacter(GLUT_STROKE_ROMAN, c)
    glPopMatrix()
```

Making things move and interact with player

Step.4 Collision detection

We check whether the collision has been occurred between the ball and three other objects in the scene, bricks, walls and paddle. The algorithm used for this is Axis-Aligned Bounding Box as we deal with ball as a box. In Collision class, there is a function for each face of the ball to be checked. We will describe the function that checks the top face of the ball now as an example.

checkTopFace( ) function

Collision is detected if there was overlap between the two objects in X and Y directions. First, we check that the top face of the ball is in between the bottom and the top faces of the brick because the motion is in Y direction. Then to check if there is overlap in X direction, the probabilities for that is mentioned in this figure.

The brick is smaller than the ball.
Left of the ball > left of the brick.
And right of the ball > right of the brick.

Right of the ball is in between right and left of the brick.

Left of the ball is in between right and left of the brick.

```
def checkTopFace():
        global Xoffset
        global Yoffset
        if (Ball.boundingBox[1] < Collision.objList[1] and Ball.boundingBox[1] > Collision.objList[3] ) and \
            ((Ball.boundingBox[0] >=  Collision.objList[0] and Ball.boundingBox[0] <= Collision.objList[2]) or \
            (Ball.boundingBox[2] >=  Collision.objList[0] and Ball.boundingBox[2] <=  Collision.objList[2]) or \
            (Ball.boundingBox[0] <  Collision.objList[0] and Ball.boundingBox[2] > Collision.objList[2])):

            Yoffset = -Yoffset
            return True
        else:
            return False
```

The bounding box of the brick min(y), max(y) and min(x), max(x) can be obtained from the corners of the brick as

- **Bottom left**: [min(x), min(y)]
- **Top right**: [max(x), max(y)]

Then stored in ***objList [minX, maxY, maxX, minY].***

Ball with Brick Detection

For each brick, there is four probabilities to detect as there is four faces. To improve calculations, we made ***getDirection( )*** function which stores the direction of ball movement as in a variable, direction, for example if direction = 0, then the ball is moving forward right, then the possible faces to check are top and right faces of the ball. The direction variable is passed to function ***detectBrick( )*** which checks the possible faces and return whether there was a collision or not.

```
def getDirection():
        if   Xoffset >= 0 and Yoffset >= 0: # going ForwardRight
            Collision.direction = 0

        elif Xoffset >= 0 and Yoffset <  0: # going BackwardRight
            Collision.direction = 3

        elif  Xoffset <  0 and Yoffset >= 0: # going ForwardLeft
            Collision.direction = 1

        elif  Xoffset <  0 and Yoffset <  0: # going BackwardLeft
            Collision.direction = 2
```

Ball with paddle Detection

It is same as the detection with bricks, but we check the bottom face only.

```
def detectPaddle(): # detect collision of ball with paddle
        Collision.objList = [Paddle.bottomLeft[0], Paddle.topRight[1], Paddle.topRight[0], Paddle.bottomLeft[1]]
        Collision.checkBottomFace()
```

Ball with walls Detection

It is similar to previous cases, as we check if the bounding box of the ball is between the scene borders.

```
def detectWall(): # detect collision of ball with wall
        global Xoffset, Yoffset, dead, life

        if Ball.boundingBox[2] >= Display.windowWidth-30 or
           Ball.boundingBox[0] <= 0+30 : # check left and right walls
            Xoffset = -Xoffset
        elif Ball.boundingBox[3] <= Display.frustumHeight[0]: # check bottom wall
            Yoffset = -Yoffset
            dead = True
            life -= 1
        elif Ball.boundingBox[1] >= Display.frustumHeight[1]-80: # check top wall
            Yoffset = -Yoffset
```

The reaction of the ball movement when a collision is detected with any object is to change the direction of ball by reversing the value of *Xoffset* and *Yoffset* variables. The new direction depends on the face of the ball that collision detected with. For example, if the ball hit an object with its top face, then the X direction will not change and the Y direction will be reversed. If that face was left face, then the X direction only will be reversed.

Step.6 Bricks Movement

The method we did that is by moving frustum in Y direction by changing its Y range in the projection function ***glOrtho( ).*** The frustum moves each 10 seconds using ***Timer( )*** function. Also the scene elements related to frustum its value had to be changed such as cameraPosition, frame and paddle.

```
frustumHeight = [0, windowHeight]
def camera():
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        glOrtho(0,Display.windowWidth, Display.frustumHeight[0],Display.frustumHeight[1], -100, 100)
        gluLookAt(Display.cameraPosition[0], Display.cameraPosition[1],Display.cameraPosition[2],
                  Display.cameraPosition[0], Display.cameraPosition[1], Display.cameraPosition[2]-1,
                  0, 1, 0)
        glMatrixMode(GL_MODELVIEW)
        glEnable(GL_DEPTH_TEST)
```

Step.7 Making Things Boom!

For sound we used Pygame as we first called the sound initialization function then we started sound effects for every ball collision with wall, paddle and bricks and when player dies. Then we called ***pygame.mixer.Sound.play(situationSound)*** in the Render function under every situation.

```
#### initializing sound ####
pygame.mixer.init()

hittingBrick = pygame.mixer.Sound("hitting brick.wav")
hittingPaddle = pygame.mixer.Sound("hitting-paddle.wav")
hittingWall = pygame.mixer.Sound("hitting-wall.wav")
lose = pygame.mixer.Sound("lose.wav")
death = pygame.mixer.Sound("death.wav")
```

Step.8 Menu

The main element in our menu is button. Button is created using Button () function. Text on button, button position, height, width, its color when mouse cursor on it, its color when mouse cursor out and its action when clicked all specified by passing them to Button () function as parameters. Button shape is a rectangle with half circles on each side. Rectangle is drawn using ***glBegin(GL_POLYGON)*** and circles are drawn using function ***drawCircle().***

Menu is rendered first and when play button is clicked, the Z value of all objects in menu changes to be out of frustum, so the menu disappears.

```
def button(text, bottomLeft, width, height, activeColor, inactiveColor, action):
    global z

    if  (bottomLeft[0]+ width+30 > Display.mouseX and Display.mouseX > bottomLeft[0]+30) and\
        (bottomLeft[1] + height > Display.mouseY and Display.mouseY > bottomLeft[1]):
        glColor4f(activeColor[0], activeColor[1], activeColor[2], activeColor[3])
        if clicked is True:
            if action == "PLAY":
                z = 200
            elif action == "QUIT":
                exit()
    else:
        glColor4f(inactiveColor[0], inactiveColor[1], inactiveColor[2], inactiveColor[3])
    # Drawing button
    glBegin(GL_POLYGON)
    glVertex3f(bottomLeft[0], bottomLeft[1], z)
    glVertex3f(bottomLeft[0]+width, bottomLeft[1], z)
    glVertex3f(bottomLeft[0]+width, bottomLeft[1]+height, z)
    glVertex3f(bottomLeft[0], bottomLeft[1]+height, z)
    glEnd()
    drawCircle(30, 285, bottomLeft[1]+30)
    drawCircle(30, 465, bottomLeft[1]+30)

    glColor(0,0,0)
    drawText(text, bottomLeft[0]+(width/2)-30, bottomLeft[1]+(height/2)-15, 0.35, 0.35, 4)
```

Step.8 Rendering Our Game

In ***Render( )*** function, the scene objects we mentioned is rendered, collision function is called, conditions of game states and objects parameters are updated.

Step.9 Taking input from user

We take input from user mouse position to move the paddle and use the menu. This is done using ***glutPassiveMotionFunc(mouseMotion)*** to get mouse position when button is not clicked and ***glutMouseFunc(mouseClick)*** to detect if left mouse button is clicked. Also an input from keyboard's space bar is taken by ***glutKeyboardFunc(Keyboard)*** to resume the game after player dies.

Step.10 Timing function

Timer( ) function is used to loop the Render() function and update frustum position to make bricks seem going down and update game states with time. This is done using ***glutTimerFunc()*** in OpenGL.

```
def Timer(t):
    global count, scoreInc

    if z!= -10: # timer doesn't count when the player is in the menu
        count += 1
        if count == 100: # for 10 seconds
            scoreInc += 1
            count = 0

        if 90 <= count < 100: # is true for 10 times to make bricks go down gradually
            # frustum changes after every 10 seconds
            Display.frustumHeight[0] += brickShiftY/10 # Translate bottom of frustum
            Display.frustumHeight[1] += brickShiftY/10 # Translate top of frustum
            if Paddle.width >= 50 :
                Paddle.width -= 3/10

    Display.camera() # update camera
    Render()

    glutTimerFunc(100, Timer, 1)
```

Step.11 Display our game

We created a Display class where we put the initialization function and the camera function and drew the window.

Code:

```
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *
import pygame
from sys import exit
import numpy as np
import math


##### initial parameters ####

Xoffset, Yoffset = 3, 3 # increment or decrement in ball position
### Bricks parameters ###
Bricks = [] # List to store all bricks, each ele-
ment of list is a brick class
brickWidth = 54
brickHeight = 26
brickShiftX= 59
brickShiftY = 32.5
firstPointX = 22 # start point of bricks
firstPointY = 583 # start point of bricks
activeBricks = 528 # number of active bricks
### Game state parameters ###
life = 3
score = 0
highestScore = 0
dead = True # if true, the player is dead
resume = False # make it true to make the player play after he is dead
### Timer parameters ###
count = 0 # temp variable to count seconds
### Coloring parameters ###
color = (1,0,0)
w = 1  # temp variable changes gradu-
ally to make color of text changes gradually
colorInc = False # if true, value of w increases
### Menu parameters ###
action = ["PLAY", "QUIT"] # actions done when buttons clicked
clicked = False # if true, the left mouse button is clicked
z = -10 # changes z of menu objects to be in or out the frustum
```

```
#### initializing sound ####
pygame.mixer.init()

hittingBrick = pygame.mixer.Sound("hitting brick.wav")
hittingPaddle = pygame.mixer.Sound("hitting-paddle.wav")
hittingWall = pygame.mixer.Sound("hitting-wall.wav")
lose = pygame.mixer.Sound("lose.wav")
death = pygame.mixer.Sound("death.wav")
mouse = pygame.mixer.Sound("Mouse_Click.wav")

############################

class Display():
    windowWidth = 750
    windowHeight = 1000
    frustumHeight = [0, windowHeight]
    mouseX = windowWidth / 2
    mouseY = windowHeight / 2
    cameraPosition = [0, frustumHeight[0], 0]  #eyeX, eyeY, eyeZ

    @staticmethod
    def init():
        glutInit()
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH)
        glutInitWindowSize(Display.windowWidth, Display.windowHeight)
        glutCreateWindow(b"Break Out game")
        Display.camera()
        glutDisplayFunc(Render)
        glutIdleFunc(Render)
        glutTimerFunc(100, Timer, 1)
        glutPassiveMotionFunc(mouseMotion)
        glutMouseFunc(mouseClick)
        glutKeyboardFunc(Keyboard)

    @staticmethod
    def camera():
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        glOrtho(0,Display.windowWidth, Display.frustumHeight[0],
                Display.frustumHeight[1], -100, 100)
        gluLookAt(Display.cameraPosition[0], Display.cameraPosition[1],
                  Display.cameraPosition[2],Display.cameraPosition[0],
                  Display.cameraPosition[1], Display.cameraPosition[2]-1,
                  0, 1, 0)
        glMatrixMode(GL_MODELVIEW)
        glEnable(GL_DEPTH_TEST)
```

```
class Ball():
    radius = 10
    position = [100, 40] # initial position of ball
    boundingBox = [position[0]-radius, position[1]+radius, posi-
tion[0]+radius, position[1]-radius] # Left, Top, Right, Bottom

class Paddle():
    width, height = 100, 20
    position = [0, 70]
    bottomLeft = [position[0]- width/2,
                  position[1]+Display.frustumHeight[0]+0.5]
    bottomRight = [position[0]+ width/2,
                   position[1]+Display.frustumHeight[0]+0.5]
    topRight = [position[0]+ width/2,
                position[1]+Display.frustumHeight[0]+height+0.5]
    topLeft =  [position[0]- width/2,
                position[1]+Display.frustumHeight[0]+height+0.5]

class Brick():
    def __init__(self, bottomLeft):
        self.bottomLeft = bottomLeft
        self.bottomRight = (bottomLeft[0]+brickWidth, bottomLeft[1])
        self.topRight = (bottomLeft[0]+brickWidth,
                         bottomLeft[1]+brickHeight)
        self.topLeft = (bottomLeft[0], bottomLeft[1]+brickHeight)

        self.active = True # if True, brick will be drawn
class Collision():

    objList = []  #minX, maxY, maxX, minY # list to store bound-
ing box of bricks
    direction = 0  # 0 --> ForwardRight, 1 --> ForwardLeft,
                     2 --> BacwardLeft, 3 --> BackwardRight

    @staticmethod
    def getDirection():
        if   Xoffset >= 0 and Yoffset >= 0: # going ForwardRight
            Collision.direction = 0

        elif Xoffset >= 0 and Yoffset <  0: # going BackwardRight
            Collision.direction = 3

        elif  Xoffset <  0 and Yoffset >= 0: # going ForwardLeft
            Collision.direction = 1

        elif  Xoffset <  0 and Yoffset <  0: # going BackwardLeft
            Collision.direction = 2
```

```
@staticmethod
    def checkTopFace(): # of the ball
        global Xoffset
        global Yoffset
        if  (Ball.boundingBox[1] < Collision.objList[1] and
             Ball.boundingBox[1] > Collision.objList[3] ) and \
            ((Ball.boundingBox[0] >=  Collision.objList[0] and
              Ball.boundingBox[0] <= Collision.objList[2]) or \
            (Ball.boundingBox[2] >=  Collision.objList[0] and
             Ball.boundingBox[2] <=  Collision.objList[2]) or \
            (Ball.boundingBox[0] <  Collision.objList[0] and
             Ball.boundingBox[2] > Collision.objList[2])):

            Yoffset = -Yoffset
            return True
        else:
            return False

    @staticmethod
    def checkBottomFace(): # of the ball
        global Xoffset
        global Yoffset
        if  (Ball.boundingBox[3] > Collision.objList[3] and
             Ball.boundingBox[3] < Collision.objList[1]) and \
            ((Ball.boundingBox[0] >=  Collision.objList[0] and
              Ball.boundingBox[0] <= Collision.objList[2]) or \
            (Ball.boundingBox[2] >=  Collision.objList[0] and
             Ball.boundingBox[2] <=  Collision.objList[2]) or \
            (Ball.boundingBox[0] <  Collision.objList[0] and
             Ball.boundingBox[2] > Collision.objList[2])):
            Yoffset = -Yoffset
            return True
        else:
            return False
```

```
    @staticmethod
    def checkRightFace(): # of the ball
        global Xoffset
        global Yoffset
        if  (Ball.boundingBox[2] > Collision.objList[0] and
             Ball.boundingBox[2] < Collision.objList[2] ) and \
            ((Ball.boundingBox[3] >=  Collision.objList[3] and
              Ball.boundingBox[3] <= Collision.objList[1]) or \
            (Ball.boundingBox[1] >=  Collision.objList[3] and
             Ball.boundingBox[1] <=  Collision.objList[1]) or \
            (Ball.boundingBox[3] <  Collision.objList[3] and
             Ball.boundingBox[1] > Collision.objList[1])):
            Xoffset = -Xoffset
            return True
        else:
            return False

    @staticmethod
    def checkLeftFace(): # of the ball
        global Xoffset
        global Yoffset
        if (Ball.boundingBox[0] < Collision.objList[2] and
            Ball.boundingBox[0] > Collision.objList[0]) and \
            ((Ball.boundingBox[3] >=  Collision.objList[3] and
              Ball.boundingBox[3] <= Collision.objList[1]) or \
            (Ball.boundingBox[1] >=  Collision.objList[3] and
             Ball.boundingBox[1] <=  Collision.objList[1]) or \
            (Ball.boundingBox[3] <  Collision.objList[3] and
             Ball.boundingBox[1] > Collision.objList[1])):
            Xoffset = -Xoffset
            return True
        else:
            return False
```

```
@staticmethod
    def detectBrick(): # detect collision of ball with brick
        Collision.getDirection()
        if   Collision.direction == 0 : # ForwardRight
            if Collision.checkTopFace() or Collision.checkRightFace():
                return True
            else:
                return False

        elif Collision.direction == 1 : #ForwardLeft
            if Collision.checkTopFace() or Collision.checkLeftFace():
                return True
            else:
                return False

        elif Collision.direction == 2 : # BackwardLeft
            if Collision.checkBottomFace() or Collision.checkLeftFace():
                return True
            else:
                return False

        elif Collision.direction == 3 : # BackwardRight
            if Collision.checkBottomFace() or Collision.checkRightFace():
                return True
            else:
                return False

    @staticmethod
    def detectWall(): # detect collision of ball with wall
        global Xoffset, Yoffset, dead, life

        if Ball.boundingBox[2] >= Display.windowWidth-30 or
           Ball.boundingBox[0] <= 0+30 : # check left and right walls
            Xoffset = -Xoffset
            pygame.mixer.Sound.play(hittingWall)
        elif Ball.boundingBox[3] <= Display.frustu-Height[0]:
            # check bottom wall
            Yoffset = -Yoffset
            dead = True
            life -= 1
            if life > 0:
                pygame.mixer.Sound.play(lose)
        elif Ball.boundingBox[1] >= Display.frustumHeight[1]-80:
            # check top wall
            Yoffset = -Yoffset
            pygame.mixer.Sound.play(hittingWall)
```

```
 @staticmethod
    def detectPaddle(): # detect collision of ball with paddle
        Collision.objList = [Paddle.bottomLeft[0], Paddle.topRight[1],
        Paddle.topRight[0], Paddle.bottomLeft[1]]
        if Collision.checkBottomFace() :
            pygame.mixer.Sound.play(hittingWall)

def resetGame():
    global Xoffset, Yoffset, Bricks, firstPointX, firstPointY,
           activeBricks, life, score, dead, resume, count, color

    Xoffset = 3
    Yoffset = 3
    Bricks = []
    firstPointX = 22
    firstPointY = 583
    activeBricks = 528
    life = 3
    score = 0
    dead = True
    resume = False
    count = 0
    color = (1,0,0)
    Display.frustumHeight = [0, Display.windowHeight]
    Paddle.width = 100
    generateLevel()

def generateLevel():
    global Bricks

    ### STRAIGHT LINES UP & DOWN DRAWING ###
    XstartPoint = firstPointX
    YstartPoint= firstPointY

    for Lines in range (0,44,1):
        for N in range (1,12,1):
            if Lines/4 == N:
             YstartPoint += 4*brickShiftY

        for Rows in range (0,12,1):
            Bricks.append(Brick((XstartPoint,YstartPoint)))
            XstartPoint += brickShiftX
        YstartPoint +=  brickShiftY
        XstartPoint = 22

generateLevel()
```

```
def button(text, bottomLeft, width, height, activeColor, inactiveColor,
           action):
    global z

    if  (bottomLeft[0]+ width+30 > Display.mouseX and
         Display.mouseX > bottomLeft[0]-30) and
        (bottomLeft[1] + height > Display.windowHeight-Display.mouseY
         and Display.windowHeight-Display.mouseY > bottomLeft[1]):
        glColor4f(activeColor[0], activeColor[1], activeColor[2],
                  activeColor[3])
        if clicked is True:
            if action == "PLAY":
                pygame.mixer.Sound.play(mouse)
                z = 200
            elif action == "QUIT":
                pygame.mixer.Sound.play(mouse)
                exit()
    else:
        glColor4f(inactiveColor[0], inactiveColor[1],
                  inactiveColor[2], inactiveColor[3])
    # Drawing button
    glBegin(GL_POLYGON)
    glVertex3f(bottomLeft[0], bottomLeft[1], z)
    glVertex3f(bottomLeft[0]+width, bottomLeft[1], z)
    glVertex3f(bottomLeft[0]+width, bottomLeft[1]+height, z)
    glVertex3f(bottomLeft[0], bottomLeft[1]+height, z)
    glEnd()
    drawCircle(30, 285, bottomLeft[1]+30)
    drawCircle(30, 465, bottomLeft[1]+30)

    glColor(0,0,0)
    drawText(text, bottomLeft[0]+(width/2)-30, bottomLeft[1]+(height/2)-
15, 0.35, 0.35, 4)

def drawCircle(r=1, centerX=0, centerY=0):
    glBegin(GL_POLYGON)
    for theta in np.arange(0, 360, 1):
        x = r * math.cos(theta * math.pi / 180) + centerX
        y = r * math.sin(theta * math.pi / 180) + centerY
        glVertex3f(x, y, z)
    glEnd()
```

```
def changeColor(y): # change color of ob-
jects to the frame color they are in its range
    global color
    if y >= Display.frustumHeight[0]+ firstPointY+ 4* brickShiftY and
       y < Display.frustumHeight[0]+ firstPointY+ 8* brickShiftY:
        color = (.9647,.4314,0)       #   ORANGE
    elif y >= Display.frustumHeight[0]+firstPointY and
         y <= Display.frustumHeight[0]+firstPointY+ 4* brickShiftY:
        color = (0,1,0)                # GREEN
    elif y >= Display.frustumHeight[0]+ firstPointY- 4* brickShiftY and
         y <= Display.frustumHeight[0]+ firstPointY:
        color  = (1,1,0)                #YELLOW
    elif y >=  Display.frustumHeight[0] and
         y <=  Display.frustumHeight[0]+ 14.738* brickShiftY:
        color = (1,1,1)                   #WHITE
    elif (y >= Display.frustumHeight[0]+firstPointY +8* brickShiftY and
          y <= Display.frustumHeight[0]+firstPointY + 12* brickShiftY)
       or(y <= Display.frustumHeight[0]+firstPointY - 15.8* brickShiftY):
        color = (0,0,1)            ## BLUE

def drawQuad(bottomLeft, bottomRight, topRight, topLeft):
    glBegin(GL_QUADS)
    glVertex(bottomLeft[0], bottomLeft[1], 0)
    glVertex(bottomRight[0], bottomRight[1], 0)
    glVertex(topRight[0], topRight[1], 0)
    glVertex(topLeft[0], topLeft[1], 0)
    glEnd()

def drawFrameStrip(firstPoint , num ): # draw frame strips
    glBegin(GL_LINES)
    # firstpoint[0] : x of the line strip of frame on left
    # firstpoint[1] : y of the line strip of frame of both sides
    # firstpoint[2] : x of the line strip of frame on right

    # Left side
    glVertex(firstPoint[0], firstPoint[1])
    glVertex(firstPoint[0], firstPoint[1]+num*brickShiftY)
    # Right side
    glVertex(firstPoint[2], firstPoint[1])
    glVertex(firstPoint[2], firstPoint[1]+num*brickShiftY)
    glEnd()
```

```
def drawFrame():
    glColor3f(.9647,.4314,0)    #   ORANGE
    glLineWidth(80)
    drawFrameStrip([Display.windowWidth-745 ,
                    Display.frustumHeight[0]+firstPointY +4* brickShiftY,
                    Display.windowWidth - 5], 4)

    glColor3f(0,1,0)     # GREEN
    drawFrameStrip([Display.windowWidth -745 ,
                    Display.frustumHeight[0]+firstPointY,
                    Display.windowWidth - 5], 4)

    glColor3f(1,1,0)        #YELLOW
    drawFrameStrip([Display.windowWidth -745 ,
                    Display.frustumHeight[0]+firstPointY -4* brickShiftY,
                    Display.windowWidth - 5], 4)

    glColor3f(0,0,1)            ## BLUE
    drawFrameStrip([Display.windowWidth -745 ,
                    Display.frustumHeight[0]+firstPointY +8* brickShiftY,
                    Display.windowWidth - 5], 4)
    drawFrameStrip([Display.windowWidth -745 ,
                 Display.frustumHeight[0]+ firstPointY -15.8*brickShiftY,
                 Display.windowWidth - 5], .6)
    drawline(Display.windowWidth, Display.frustumHeight[1]-7)

    glColor3f(1,1,1)        #WHITE
    drawFrameStrip([Display.windowWidth -745 ,
                    Display.frustumHeight[0]+0 ,
                    Display.windowWidth - 5], 14.738)

def drawline(first,second):
    glBegin(GL_LINES)
    glVertex(0,second -3)
    glVertex(first , second-3)
    glEnd()
```

```
def mouseClick(button, state, x, y):
    global clicked

    if button == GLUT_LEFT_BUTTON:
        if state == GLUT_DOWN:
            clicked = True
        else:
            clicked = False

def mouseMotion(x, y):
    Display.mouseX = x
    Display.mouseY = y

def Keyboard(key, x, y):
    global resume
    if key == b" ": # press space bar to resume when player dies
        resume = True
    if ord(key) == 27 : # press Esc button to exit
        exit()

def drawText(string, posX, posY, scaleX, scaleY, lineWidth):
    glLineWidth(lineWidth)
    glPushMatrix()
    glLoadIdentity()
    glTranslate(posX, posY, 0)
    glScale(scaleX, scaleY, 3)
    string = string.encode()  # conver-
sion from Unicode string to byte string
    for c in string:
        glutStrokeCharacter(GLUT_STROKE_ROMAN, c)
    glPopMatrix()
```

```
def Timer(t):
    global count

    if z!= -10: # timer doesn't count when the player is in the menu
        count += 1
        if count == 100: # for 10 seconds
            count = 0

        if 90 <= count < 100:
            # is true for 10 times to make bricks go down gradually
            # frustum changes after every 10 seconds
            Display.frustumHeight[0] += brickShiftY/10
            # Translate bottom of frustum
            Display.frustumHeight[1] += brickShiftY/10
            # Translate top of frustum
            if Paddle.width >= 50:
                Paddle.width -= 3/10

    Display.camera() # update camera
    Render()

    glutTimerFunc(100, Timer, 1)
```

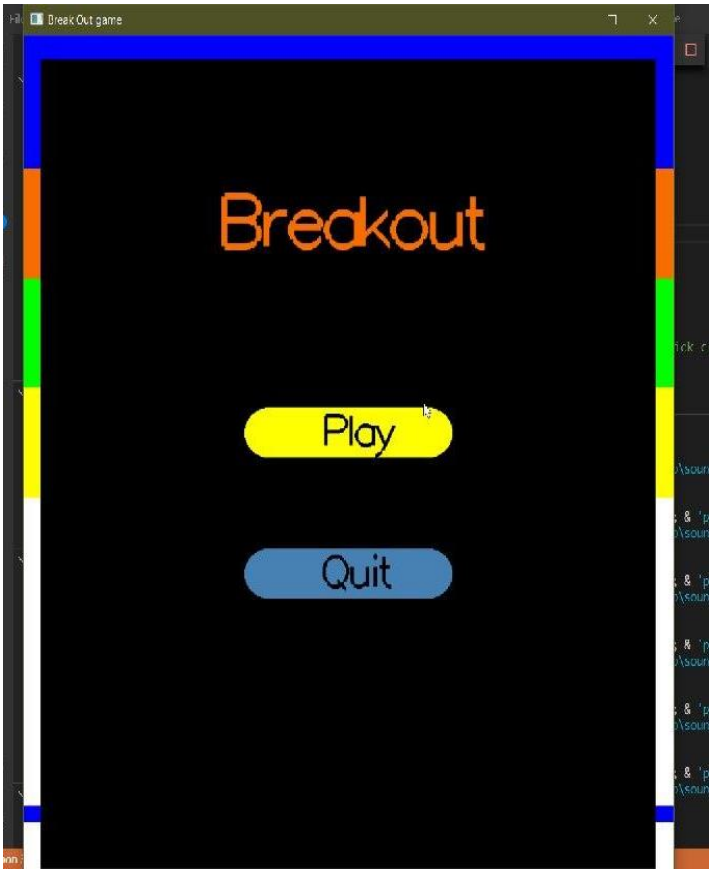
```
def Render():
    global Xoffset, Yoffset, activeBricks, dead, life,\
           score, resume, color, highestScore, colorInc, w

    glClearColor(0,0,0,0)
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

    drawFrame()

    if z != 200: # render menu only
        button("Play", (285,(Display.frustumHeight[1])/2),
               180,60, (1,1,0,0.5), (0.2745,0.5089,0.7588,0.5), "PLAY")
        button("Quit", (285,(Display.frustu-Height[1])/3),
               180,60, (1,1,1,0.5), (0.28,0.5,0.7,0.5), "QUIT")
        glColor(.9647,.4314,0)
        drawText("Breakout", Display.windowWidth-525,
                 Display.windowHeight-250, 0.6, 0.6, 6.5)

    else: # render game without menu

        #### conditions to change text color ####
        if colorInc :
            if w < 1:
                w += 0.03
            else:
                colorInc = False
        else:
            if w > 0:
                w -= 0.03
            else:
                colorInc = True

        if dead is True :
            Ball.position[0] = Paddle.position[0]
            # update ball position to stick on paddle
            Ball.position[1] = Display.frustumHeight[0]+
            Paddle.position[1]+ Paddle.height+ Ball.radius+ 2
            Xoffset, Yoffset = 0, 0 # stop the ball until resume

            if resume :
                dead = False
                resume = False
                Xoffset = 3
                Yoffset = 3
```

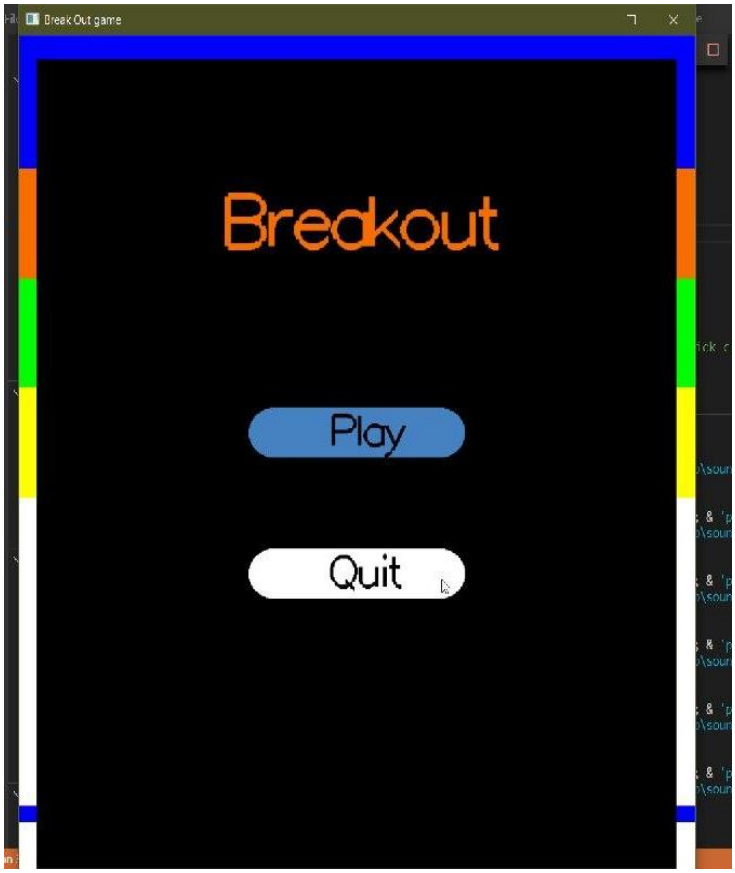
```
if activeBricks > 0 and life > 0 :
    # draw game states on screen
    glColor(w, w, w)  # to make text apper and disapper with time
    drawText( str(score), Display.windowWidth/3-20,
    Display.frustumHeight[0]+20, 0.3, 0.3, 3)
    glColor(1,1,1)
    drawText( str(life), Display.windowWidth*2/3-20,
    Display.frustumHeight[0]+20, 0.3, 0.3, 3)

    # Draw paddle
    glColor3f(0,0,1)            ## BLUE
    glPushMatrix()
    glLoadIdentity()
    if not (Display.mouseX < Paddle.width/2 or
            Display.mouseX > Display.windowWidth-Paddle.width/2):
    # to make paddle between left and right borders of window
        Paddle.position[0] = Display.mouseX # updating paddle position
        Paddle.bottomLeft = [Paddle.position[0]- Paddle.width/2,
        Paddle.position[1]+Display.frustumHeight[0]]
        Paddle.bottomRight = [Paddle.position[0]+ Paddle.width/2,
        Paddle.position[1]+Display.frustumHeight[0]]
        Paddle.topRight = [Paddle.position[0]+ Paddle.width/2,
        Paddle.position[1]+Display.frustumHeight[0]+Paddle.height]
        Paddle.topLeft =  [Paddle.position[0]- Paddle.width/2,
        Paddle.position[1]+Display.frustumHeight[0]+Paddle.height]
    drawQuad(Paddle.bottomLeft, Paddle.bottomRight, Paddle.topRight,
             Paddle.topLeft)
    glPopMatrix()

    # Draw ball
    changeColor(Ball.position[1])
    glColor(color[0], color[1], color[2],0)
    glPushMatrix()
    glLoadIdentity()
    glTranslate(Ball.position[0], Ball.position[1],0)
    glutSolidSphere(Ball.radius, 50, 50)
    glPopMatrix()
```

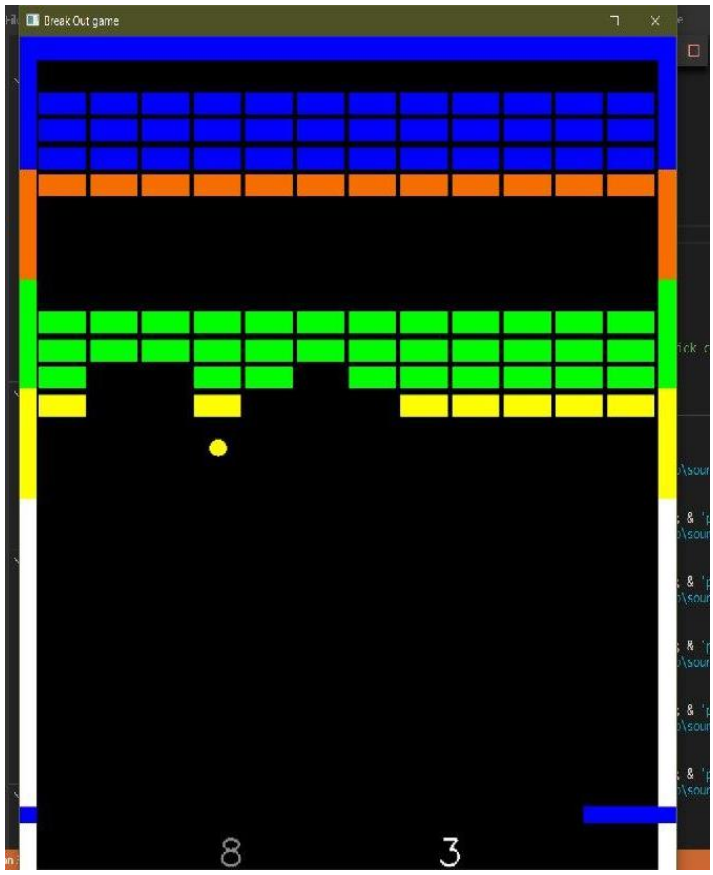
```
    # Draw bricks
        for i in range(len(Bricks)):
            if Bricks[i].active is True:
                if Bricks[i].bottomLeft[1] <= Paddle.topLeft[1]+2*Ball.radius+1:
                   # player dies when bricks go down
                    life = 0
                    break
                changeColor(Bricks[i].bottomLeft[1])
                glColor3f(color[0], color[1], color[2])
                drawQuad(Bricks[i].bottomLeft, Bricks[i].bottomRight,
                         Bricks[i].topRight, Bricks[i].topLeft)

            if not dead: # doen't check collision when player is dead # saves computations
                Collision.objList = [Bricks[i].bottomLeft[0], Bricks[i].topRight[1],
                Bricks[i].topRight[0], Bricks[i].bottomLeft[1]] # updating objlist
                if Bricks[i].active is True and Collision.detectBrick() :
                # if true, this brick will not be drawn
                    Bricks[i].active = False
                    activeBricks -= 1
                    score += 1
                    pygame.mixer.Sound.play(hittingBrick)

        if not dead: # doen't check collision when player is dead # saves computations
            Collision.detectWall()
            Collision.detectPaddle()
            Ball.position[0] += Xoffset # updating ball position
            Ball.position[1] += Yoffset
            Ball.boundingBox = [Ball.position[0]-Ball.radius,
                                Ball.position[1]+Ball.radius, Ball.position[0]+Ball.radius,
                                Ball.position[1]-Ball.radius] # Left, Top, Right, Bottom
        if life == 0 :
            highestScore = max(highestScore,score)
            pygame.mixer.Sound.play(death)   # Lose

    if life == 0  :
        glColor(0.5, 0.5, w)
        drawText("Press Space bar to try again", 80,
                (Display.frustumHeight[0]+Display.frustumHeight[1])/2+80, 0.3, 0.3, 3)
        drawText("Press Esc to quit", 210,
                (Display.frustumHeight[0]+Display.frustumHeight[1])/2-80, 0.3, 0.3, 2.5)
        drawText("HIGH SCORE: "+str(highestScore), 220, Display.frustumHeight[0]+20, 0.3, 0.3, 3)

        if resume is True:
            resetGame()

glutSwapBuffers()
```

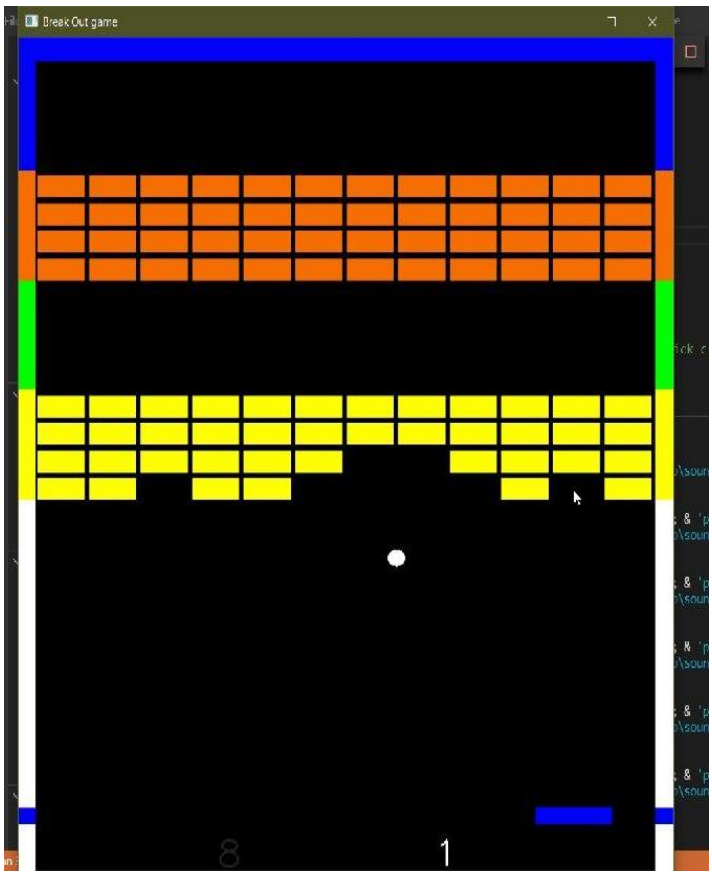
```
def main():
    Display.init()
    pygame.init()
    glutMainLoop()

main()
```

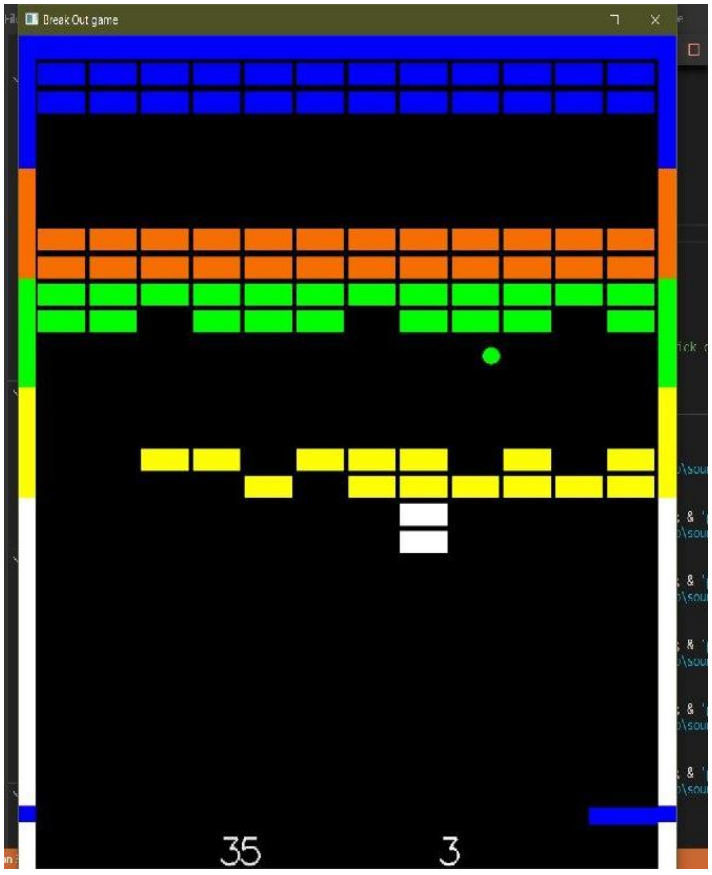
Our Game

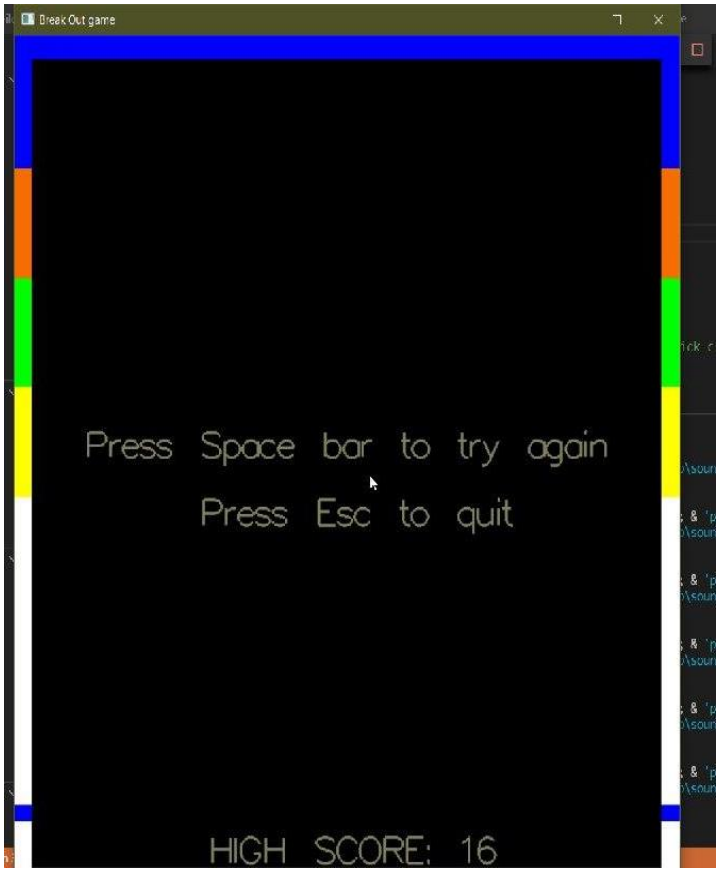


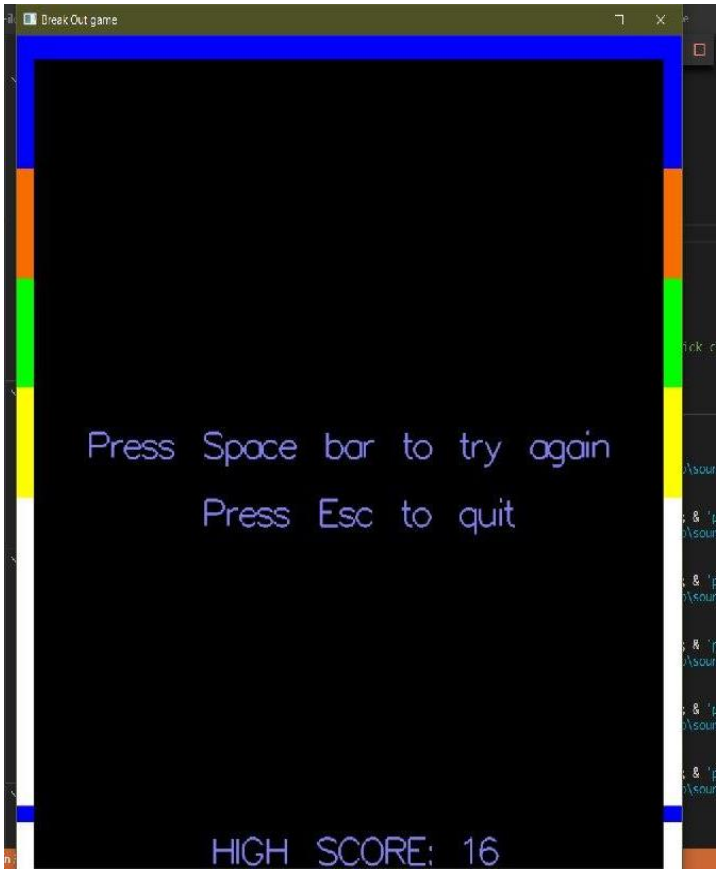



Break Out game
8
3


Break Out game
8
1


Break Out game
35
3


Break Out game
Press Space bar to try again
Press Esc to quit
HIGH SCORE: 16


Break Out game
Press Space bar to try again
Press Esc to quit
HIGH SCORE: 16

References

- **Books**

  - Dr. Mohamed Aita. *Lectures Notes, 2020*

  - Shreiner, Dave, et al. *OpenGL programming guide: The Official guide to learning OpenGL.*

  - Sweigart, Al. *Making Games with Python & Pygame.*

  - McGugan, Will. *Beginning game development with Python and Pygame: from novice to professional*.

- **Websites**

  - "Python Tutorial." https://www.tutorialspoint.com/python/index.htm

  - *LearnOpenGL*, https://learnopengl.com/In-Practice/2D-Game/Collisions/Collision-detection

  - "PyOpenGL 3.1.0." *PyOpenGL 3.1.0 Function Reference*, http://pyopengl.sourceforge.net/documentation/manual-3.0/index.html