CSE423

Section: 01

Group - 01
OpenGL_Pong

Submitted by

Abir Ahammed Bhuiyan	20101197
Mirza Abyaz Awsaf	20101146
Allama Bakhtiyar Nafis	18201085

Please visit any of these links to download the whole program

https://github.com/eniac00/OpenGL_Pong

https://0x0.st/H3eU.zip

Folder Structure:

```
.
├── Components.py
├── Game.py
├── main.py
├── Modules
│   ├── CircleModule.py
│   ├── ImportGL.py
│   ├── __init__.py
│   ├── LineModule.py
│   └── RectangleModule.py
├── README.md
├── ScreenShots
│   ├── demo.gif
│   ├── pong_1.png
│   ├── pong_2.png
│   ├── pong_3.png
│   └── pong_4.png
├── Sound
│   ├── boink2.wav
│   ├── heehee.mp3
│   ├── hello.mp3
│   ├── paddleBreak.wav
│   └── twang.wav
├── TextUtils.py
└── Utils.py
```

4 directories, 21 files

File path: ./OpenGL_Pong/main.py

```
#!/usr/bin/env python3


from Utils import *

class Menu:
    def __init__(self):
        self.width = 500  # screen width
        self.height = 700 # screen height

        self.hover = None
```

```
    def drawTitle(self):
        Rectangle(50, 630, 70, 70, fill=True)
        Rectangle(50, 560, 20, 50, fill=True)

        Rectangle(151, 590, 60, 60, fill=True)

        Rectangle(255, 601, 15, 70, fill=True)
        Rectangle(270, 600, 50, 15, fill=True)
        Rectangle(320, 601, 15, 70, fill=True)

        Rectangle(370, 630, 15, 100, fill=True)
        Rectangle(385, 630, 60, 20, fill=True)
        Rectangle(370, 530, 80, 15, fill=True)
        Rectangle(450, 580, 15, 50, fill=True)


    def drawComponents(self):
        self.drawTitle()

        if self.hover:
            if self.hover == 1:
                Rectangle(90, 440, 320, 80, fill=True)
                Rectangle(90, 320, 320, 80)
                Rectangle(90, 200, 320, 80)

                drawText(200, 390, "P1 vs P2", color=(0, 0, 0))
                drawText(200, 270, "P2 vs AI")
                drawText(220, 150, "Quit")

            if self.hover == 2:
                Rectangle(90, 440, 320, 80)
                Rectangle(90, 320, 320, 80, fill=True)
                Rectangle(90, 200, 320, 80)

                drawText(200, 390, "P1 vs P2")
                drawText(200, 270, "P2 vs AI", color=(0, 0, 0))
                drawText(220, 150, "Quit")

            if self.hover == 3:
                Rectangle(90, 440, 320, 80)
                Rectangle(90, 320, 320, 80)
                Rectangle(90, 200, 320, 80, fill=True, color=(1, 0,
0))
```

```
            drawText(200, 390, "P1 vs P2")
            drawText(200, 270, "P2 vs AI")
            drawText(220, 150, "Quit")
    else:
        Rectangle(90, 440, 320, 80)
        drawText(200, 390, "P1 vs P2")
        Rectangle(90, 320, 320, 80)
        drawText(200, 270, "P2 vs AI")
        Rectangle(90, 200, 320, 80)
        drawText(220, 150, "Quit")

    drawTextSmall(170, 40, "created by eniac00")


def iterate(self):
    glViewport(0, 0, self.width, self.height)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    glOrtho(0.0, self.width, 0.0, self.height, 0.0, 1.0)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()


def showScreen(self):
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLoadIdentity()
    self.iterate()
    glClearColor(0, 0, 0, 1.0)
    self.drawComponents()
    glutSwapBuffers()


def checkButtonClick(self, x, y):
    if self.height-y >= 360 and self.height-y <= 440:
        if x >= 90 and x <= 410:
            subprocess.Popen(["python", "Game.py"])
            glutLeaveMainLoop()
    elif self.height-y >= 240 and self.height-y <= 320:
        if x >= 90 and x <= 410:
            print("Coming soon ...")
    elif self.height-y >= 120 and self.height-y <= 200:
        if x >= 90 and x <= 410:
            glutLeaveMainLoop()
    else:
        pass
```

```
    glutPostRedisplay()


def checkButtonHover(self, x, y):
    if self.height-y >= 360 and self.height-y <= 440:
        if x >= 90 and x <= 410:
            self.hover = 1
    elif self.height-y >= 240 and self.height-y <= 320:
        if x >= 90 and x <= 410:
            self.hover = 2
    elif self.height-y >= 120 and self.height-y <= 200:
        if x >= 90 and x <= 410:
            self.hover = 3
    else:
        self.hover = None

    glutPostRedisplay()


def animate(self):
    glutPostRedisplay()


def keyboardListener(self, key, x, y):
    if key == b"w":
        pass
    glutPostRedisplay()


def mouseListener(self, button, state, x, y):
    if button == GLUT_LEFT_BUTTON and state == GLUT_DOWN:
        self.checkButtonClick(x, y)
    glutPostRedisplay()


def specialKeyListener(self, key, x, y):
    if key == GLUT_KEY_UP:
        if not self.hover:
            self.hover = 1
        else:
            if self.hover == 1:
                self.hover = 3
            else:
                self.hover = (abs(self.hover-2) % 3) + 1
```

```
        if key == GLUT_KEY_DOWN:
            if not self.hover:
                self.hover = 3
            else:
                self.hover = ((self.hover) % 3)  + 1
        glutPostRedisplay()


    def run(self):
        glutInit()
        glutInitDisplayMode(GLUT_RGBA)
        glutInitWindowSize(self.width, self.height)
        glutInitWindowPosition(0, 0)
        glutCreateWindow(b"OpenGL Pong")
        glutDisplayFunc(self.showScreen)
        glutIdleFunc(self.animate)
        glutKeyboardFunc(self.keyboardListener)
        glutPassiveMotionFunc(self.checkButtonHover)
        glutMouseFunc(self.mouseListener)
        glutSpecialFunc(self.specialKeyListener)
        glutMainLoop()



if __name__ == "__main__":
    Menu().run()
```

File path: ./OpenGL_Pong/Utils.py

```
#!/usr/bin/env python3

from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *

from Modules.LineModule import Line
from Modules.CircleModule import Circle
from Modules.RectangleModule import Rectangle

from Components import *
from TextUtils import *

from random import randint, sample
```

```
import pygame
import subprocess
import sys
import time
```

File path: ./OpenGL_Pong/Components.py

```
#!/usr/bin/env python3


from Utils import *


class Pad:
    def __init__(self, x, y, radius, width=1):
        self.x = x
        self.y = y
        self.width = width
        self.radius = radius


    def draw(self, color=(1, 1, 1)):
        Line(x0=self.x, y0=self.y+self.radius, x1=self.x,
y1=self.y-self.radius, size=self.width, color=color)

    def reset(self, x, y):
        self.x = x
        self.y = y


class Ball:
    def __init__(self, x, y):
        self.x = x
        self.y = y


    def draw(self):
        Circle(x=self.x, y=self.y, radius=4, weight=6)

    def reset(self, x, y):
        self.x = x
        self.y = y
```

```
class Arrow:
    def __init__(self, x=320, y=650, d=25, width=3, color=(0, 0.705,
0.705)):
        self.x = x
        self.y = y
        self.d = d
        self.width = width
        self.color = color
        self.vertex_x =  self.x - self.d
        self.vertex_y = self.y


    def draw(self):
        Line(self.x, self.y, self.x + self.d, self.y, self.width,
self.color)
        Line(self.x, self.y, self.vertex_x, self.vertex_y,
self.width, self.color)
        Line(self.vertex_x, self.vertex_y, self.x, self.y + self.d,
self.width, self.color)
        Line(self.vertex_x, self.vertex_y, self.x, self.y - self.d,
self.width, self.color)



class Pause:
    def __init__(self, x=445, y=650, d=25, width=3, color=(0.125,
0.660, 0.286)):
        self.x = x
        self.y = y
        self.d = d
        self.width = width
        self.space = 15
        self.color = color


    def draw(self):
        Line(self.x + self.space, self.y + self.d, self.x +
self.space, self.y - self.d, self.width, self.color)
        Line(self.x - self.space, self.y + self.d, self.x -
self.space, self.y - self.d, self.width, self.color)
```

```
class Play:
    def __init__(self, x=435, y=650, d=25, width=3, color=(0.125,
0.660, 0.286)):
        self.x = x
        self.y = y
        self.d = d
        self.width = width
        self.color = color
        self.vertex_x =  self.x + self.d
        self.vertex_y = self.y


    def draw(self):
        Line(self.x, self.y + self.d, self.x, self.y - self.d,
self.width, self.color)
        Line(self.vertex_x, self.y, self.x, self.y + self.d,
self.width, self.color)
        Line(self.vertex_x, self.y, self.x, self.y - self.d,
self.width, self.color)



class Cross:
    def __init__(self, x=580, y=650, d=25, width=3, color=(0.840,
0.0924, 0.0924)):
        self.x = x
        self.y = y
        self.d = d
        self.width = width
        self.color = color


    def draw(self):
        Line(self.x, self.y, self.x + self.d, self.y + self.d,
self.width, self.color)
        Line(self.x, self.y, self.x + self.d, self.y - self.d,
self.width, self.color)
        Line(self.x, self.y, self.x - self.d, self.y + self.d,
self.width, self.color)
        Line(self.x, self.y, self.x - self.d, self.y - self.d,
self.width, self.color)
```

File path: ./OpenGL_Pong/Game.py

```
#!/usr/bin/env python3

from Utils import *

class Game:
    def __init__(self):
        self.width = 900  # screen width
        self.height = 700 # screen height
        self.upper_bound = self.height-100   # real height in which
ball can move
                                             # leaving 100 pixel from
the top side for showing buttons(pause, exit, retry, point)

        self.pad_height = 60 # pad length
        self.pad1 = Pad(30, 300, self.pad_height//2, 8) #
initializing left pad
        self.pad2 = Pad(870, 300, self.pad_height//2, 8) #
initializing right pad
        self.pad_y_vel = 80  # pad vertical movement

        self.ball = Ball(self.width//2, self.height//2) #
initializing the ball
        self.MAX_VEL = 10  # ball max velocity possible
        self.ball_x_vel = self.MAX_VEL * int(sample([1, -1], 1)[0])
# ball x direction velocity
        self.ball_y_vel = 0  # ball y direction velocity

        self.p1_point = 0   # player_1/left_pad point
        self.p2_point = 0   # player_2/right_pad point

        self.freeze = True     # tracking if the pause button is been
pressed or not
        self.gameOver = False   # tracking if gameOver happened (if
any of the paddle misses 3 ball this means gameover)
        self.winner = None      # after gameOver who is the winner

        pygame.init()
        pygame.mixer.init()
```

```
        self.boink_sound = pygame.mixer.Sound("./Sound/boink2.wav")
# sound when ball hits the pad
        self.gameover_sound =
pygame.mixer.Sound("./Sound/heehee.mp3") # game over sound jardinains
laughing
        self.pad_break_sound =
pygame.mixer.Sound("./Sound/paddleBreak.wav") # paddle missing the
incoming ball
        self.hello_sound = pygame.mixer.Sound("./Sound/hello.mp3")  #
game start sound
        self.boundary_sound = pygame.mixer.Sound("./Sound/twang.wav")
# ball hits the boundary

        self.hello_sound.play()  # game starts so playing

        self.prev_time = time.time()  # necessary for calculating
time delta
        self.dt = 0      # necessary for calculating time delta
        self.fps = 30    # necessary for calculating time delta



    def calcDeltaTime(self):
        self.dt = time.time() - self.prev_time
        self.prev_time = time.time()
        self.dt *= self.fps


    def drawComponents(self):
        Line(250, 700, 250, 600, size=1)
        Line(380, 700, 380, 600, size=1)
        Line(510, 700, 510, 600, size=1)
        Line(650, 700, 650, 600, size=1)

        Line(0, self.upper_bound, self.width, self.upper_bound,
size=2) # Boundary Line

        drawText(70, 645, f"Player_1: {self.p1_point:02}")
        drawText(720, 645, f"Player_2: {self.p2_point:02}")

        self.pad1.draw()
        self.pad2.draw()
        self.ball.draw()

        Arrow().draw()
```

```
        Pause().draw() if not self.freeze else Play().draw()
        Cross().draw()

        drawPauseText() if self.freeze and not self.gameOver else
None
        drawGOText(winner=self.winner) if self.gameOver else None


    def iterate(self):
        glViewport(0, 0, self.width, self.height)
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        glOrtho(0.0, self.width, 0.0, self.height, 0.0, 1.0)
        glMatrixMode(GL_MODELVIEW)
        glLoadIdentity()


    def showScreen(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        self.iterate()
        glClearColor(0, 0, 0, 1.0)
        self.calcDeltaTime()
        self.drawComponents()
        self.checkBallBoundaryCollision()
        self.checkBallPadCollision()
        glutSwapBuffers()


    def checkBallPadCollision(self):
        if self.ball.x < -20 or self.ball.x > self.width+20:
            if self.ball.x < 0:
                if self.p2_point == 2:
                    self.gameOver = True
                    self.gameover_sound.play()
                    self.p2_point += 1
                    self.winner = 2
                else:
                    self.p2_point += 1
                    self.pad_break_sound.play()
                self.ball.reset(self.width//2+randint(10, 50),
self.height//2+randint(20, 100))
                self.ball_x_vel *= 1
                self.ball_y_vel *= int(sample([1, -1], 1)[0])
```

```
            if self.ball.x > self.width:
                if self.p1_point == 2:
                    self.gameOver = True
                    self.gameover_sound.play()
                    self.p1_point += 1
                    self.winner = 1
                else:
                    self.p1_point += 1
                    self.pad_break_sound.play()
                self.ball.reset(self.width//2+randint(10, 50),
self.height//2+randint(20, 100))
                self.ball_x_vel *= -1
                self.ball_y_vel *= int(sample([1, -1], 1)[0])

        if self.ball.y <= self.pad1.y+30 and self.ball.y >=
self.pad1.y-30:
            if self.ball.x-8 <= self.pad1.x:
                self.boink_sound.play()
                self.ball_x_vel *= -1

                difference_in_y = self.pad1.y - self.ball.y
                reduction_factor = (self.pad_height/2)/self.MAX_VEL
                y_vel = difference_in_y / reduction_factor
                self.ball_y_vel = -1 * y_vel

        if self.ball.y <= self.pad2.y+30 and self.ball.y >=
self.pad2.y-30:
            if self.ball.x+8 >= self.pad2.x:
                self.boink_sound.play()
                self.ball_x_vel *= -1

                difference_in_y = self.pad2.y - self.ball.y
                reduction_factor = (self.pad_height/2)/self.MAX_VEL
                y_vel = difference_in_y / reduction_factor
                self.ball_y_vel = -1 * y_vel


    def checkBallBoundaryCollision(self):
        if  self.ball.y+5 >= self.upper_bound:
            self.ball_y_vel *= -1
            self.boundary_sound.play()
        elif self.ball.y-5 <= 0:
            self.ball_y_vel *= -1
            self.boundary_sound.play()
```

```
def resetEverything(self):
    self.p1_point = 0
    self.p2_point = 0
    self.ball.reset(self.width//2, self.height//2)
    self.pad1.reset(30, 300)
    self.pad2.reset(870, 300)
    self.gameOver = False
    self.ball_x_vel = self.MAX_VEL * int(sample([1, -1], 1)[0])
    self.ball_y_vel = 0
    self.hello_sound.play()


def checkButton(self, x, y):
    if y <= self.height and y >= self.upper_bound:
        if x >= 250 and x <= 380:
            self.resetEverything()
        elif x >= 380 and x <= 510:
            self.freeze = True if not self.freeze else False
        elif x >= 510 and x <= 650:
            subprocess.Popen(["python", "main.py"])
            glutLeaveMainLoop()

def animate(self):
    if not self.freeze and not self.gameOver:
        self.ball.x += self.ball_x_vel * self.dt
        self.ball.y += self.ball_y_vel * self.dt
    glutPostRedisplay()


def keyboardListener(self, key, x, y):
    if key == b" ":
        self.freeze = True if not self.freeze else False

    if not self.freeze and not self.gameOver:
        if key == b"w":
            if not self.pad1.y >= self.upper_bound-45:
                self.pad1.y += self.pad_y_vel * self.dt
        if key == b"s":
            if not self.pad1.y <= 0+45:
                self.pad1.y -= self.pad_y_vel * self.dt

    glutPostRedisplay()
```

```
    def mouseListener(self, button, state, x, y):
        if button == GLUT_LEFT_BUTTON and state == GLUT_DOWN:
            self.checkButton(x, self.height-y)
        glutPostRedisplay()


    def specialKeyListener(self, key, x, y):
        if not self.freeze and not self.gameOver:
            if key == GLUT_KEY_UP:
                if not self.pad2.y >= self.upper_bound-45:
                    self.pad2.y += self.pad_y_vel * self.dt
            if key == GLUT_KEY_DOWN:
                if not self.pad2.y <= 0+45:
                    self.pad2.y -= self.pad_y_vel * self.dt
        glutPostRedisplay()


    def run(self):
        glutInit()
        glutInitDisplayMode(GLUT_RGBA)
        glutInitWindowSize(self.width, self.height)
        glutInitWindowPosition(0, 0)
        glutCreateWindow(b"OpenGL Pong")
        glutDisplayFunc(self.showScreen)
        glutIdleFunc(self.animate)
        glutKeyboardFunc(self.keyboardListener)
        glutMouseFunc(self.mouseListener)
        glutSpecialFunc(self.specialKeyListener)
        glutMainLoop()



if __name__ == "__main__":
    Game().run()
```

File path: ./OpenGL_Pong/Modules/ImportGL.py

```
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *
```

File path: ./OpenGL_Pong/Modules/__init__.py

File path: ./OpenGL_Pong/Modules/RectangleModule.py

```
#!/usr/bin/env python3


from .ImportGL import *
from .LineModule import Line

class Rectangle:
    def __init__(self, x0, y0, w, h, fill=False, color=(1, 1, 1)):
        self.x0 = x0
        self.y0 = y0
        self.w = w
        self.h = h
        self.color=color

        if fill:
            self.drawFillRect()
        else:
            self.drawRect()


    def drawRect(self):
        x1 = self.x0 + self.w
        y1 = self.y0

        x2 = x1
        y2 = y1 - self.h

        x3 = self.x0
        y3 = y2

        Line(self.x0, self.y0, x1, y1, size=2, color=self.color)
        Line(x1, y1, x2, y2, size=2, color=self.color)
        Line(x2, y2, x3, y3, size=2, color=self.color)
        Line(x3, y3, self.x0, self.y0, size=2, color=self.color)


    def drawFillRect(self):
```

```
        x1 = self.x0 + self.w
        y1 = self.y0

        for i in range(self.h, 0, -2):
            Line(self.x0, self.y0-i, x1, y1-i, size=2,
color=self.color)
```

File path: ./OpenGL_Pong/Modules/LineModule.py

```
#!/usr/bin/env python3


from .ImportGL import *


class Line:
    def __init__(self, x0, y0, x1, y1, size = 1, color = (1, 1, 1)):
        self.size = size
        self.color = color

        self.zoneZero = {
                0: lambda x, y: (x, y),
                1: lambda x, y: (y, x),
                2: lambda x, y: (y, -x),
                3: lambda x, y: (-x, y),
                4: lambda x, y: (-x, -y),
                5: lambda x, y: (-y, -x),
                6: lambda x, y: (-y, x),
                7: lambda x, y: (x, -y),
                }

        self.zoneOriginal = {
                0: lambda x, y: (x, y),
                1: lambda x, y: (y, x),
                2: lambda x, y: (-y, x),
                3: lambda x, y: (-x, y),
                4: lambda x, y: (-x, -y),
                5: lambda x, y: (-y, -x),
```

```
            6: lambda x, y: (y, -x),
            7: lambda x, y: (x, -y),
            }

    self.zone = self.findZone(x0, y0, x1, y1)
    # print(f"{self.zone} -> zone")

    x0, y0 = self.zoneZero[self.zone](x0, y0)
    x1, y1 = self.zoneZero[self.zone](x1, y1)
    # print(f"({x0}, {y0}) ({x1}, {y1}) -> converted")

    self.midPointAlgo(x0, y0, x1, y1)


def findZone(self, x0, y0, x1, y1):
    dy = y1 - y0
    dx = x1 - x0

    if (abs(dx) > abs(dy)):
        if (dx >= 0 and dy >= 0):
            return 0
        elif (dx <= 0 and dy >= 0):
            return 3
        elif (dx <= 0 and dy <= 0):
            return 4
        else:
            return 7
    else:
        if (dx >= 0 and dy >= 0):
            return 1
        elif (dx <= 0 and dy >= 0):
            return 2
        elif (dx <= 0 and dy <= 0):
            return 5
        else:
            return 6


def drawPoint(self, x, y):
    glPointSize(self.size)
    glBegin(GL_POINTS)
    glColor3f(self.color[0], self.color[1], self.color[2])
    glVertex2f(x, y)
    glEnd()
```

```
    def midPointAlgo(self, x0, y0, x1, y1):
        dy = y1 - y0
        dx = x1 - x0
        d_init = 2*dy - dx
        incE = 2*dy
        incNE = 2*(dy - dx)

        while (x0 <= x1):
            a, b = self.zoneOriginal[self.zone](x0, y0)
            self.drawPoint(a, b)
            # print(f"{a}, {b} done")

            x0 += 1

            if d_init <= 0:
                d_init += incE
            else:
                y0 += 1
                d_init += incNE
```

File path: ./OpenGL_Pong/Modules/CircleModule.py

```
#!/usr/bin/env python3


from .ImportGL import *


class Circle:
    def __init__(self, x=0, y=0, radius=10, weight=2, color=(1, 1,
1)):
        self.x = x
        self.y = y
        self.radius = radius
        self.color = color
        self.weight = weight

        self.eightWaySymmetry = {
                0: lambda x, y: (y, x),
                1: lambda x, y: (x, y),
                2: lambda x, y: (-x, y),
                3: lambda x, y: (-y, x),
                4: lambda x, y: (-y, -x),
```

```
                5: lambda x, y: (-x, -y),
                6: lambda x, y: (x, -y),
                7: lambda x, y: (y, -x),
                }

        self.midpointCircleAlgo()



    def circlePoint(self, x, y):
        for i in range(0, 8):
            a, b = self.eightWaySymmetry[i](x, y)
            self.drawPoint(a+self.x, b+self.y)


    def drawPoint(self, x, y):
        glPointSize(self.weight)
        glBegin(GL_POINTS)
        glColor3f(self.color[0], self.color[1], self.color[2])
        glVertex2f(x, y)
        glEnd()

    def midpointCircleAlgo(self):
        x = 0
        y = self.radius
        d = 1 - self.radius
        self.circlePoint(x, y)

        while (x < y):
            if (d < 0): # E
                d += 2*x + 3
                x += 1
            else:
                d += 2*x - 2*y + 5
                x += 1
                y -= 1
            self.circlePoint(x, y)
```

File path: ./OpenGL_Pong/TextUtils.py

```
from Utils import *

def drawText(x, y, text, color=(1, 1, 1)):
    glColor3f(color[0], color[1], color[2])
```

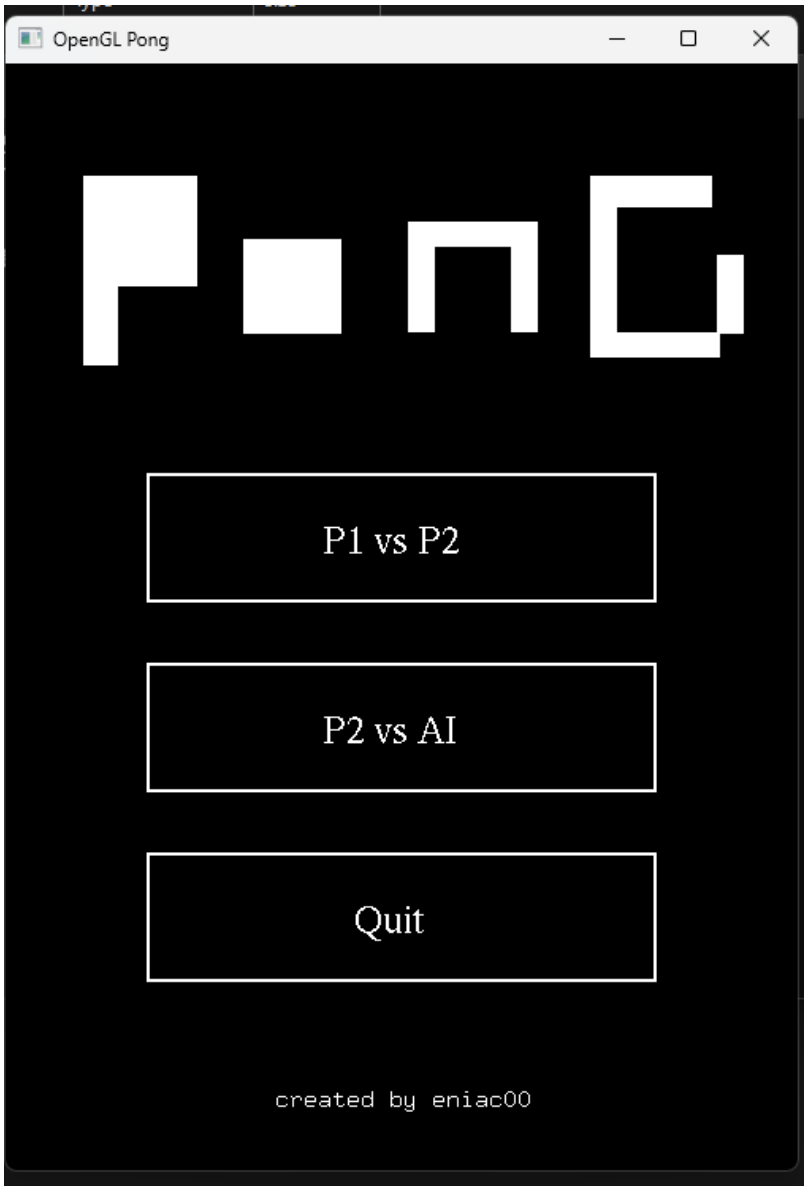
```
    glRasterPos2f(x, y)
    for character in text:
        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,
ord(character))


def drawTextSmall(x, y, text, color=(1, 1, 1)):
    glColor3f(color[0], color[1], color[2])
    glRasterPos2f(x, y)
    for character in text:
        glutBitmapCharacter(GLUT_BITMAP_9_BY_15, ord(character))


def drawPauseText(x=430, y=300):
    Line(x0=x-10, y0=y-20, x1=x+90, y1=y-20)
    Line(x0=x-10, y0=y+35, x1=x+90, y1=y+35)
    drawText(x, y, "Paused!")


def drawGOText(x=400, y=350, winner=None):
    if winner:
        Line(x0=x-10, y0=y+35, x1=x+135, y1=y+35)
        drawText(x, y, "GameOver!!!")
        drawText(x, y-50, f"Player {winner} won!")
        Line(x0=x-10, y0=y-70, x1=x+140, y1=y-70)
    else:
        print("error")
```

Demo:

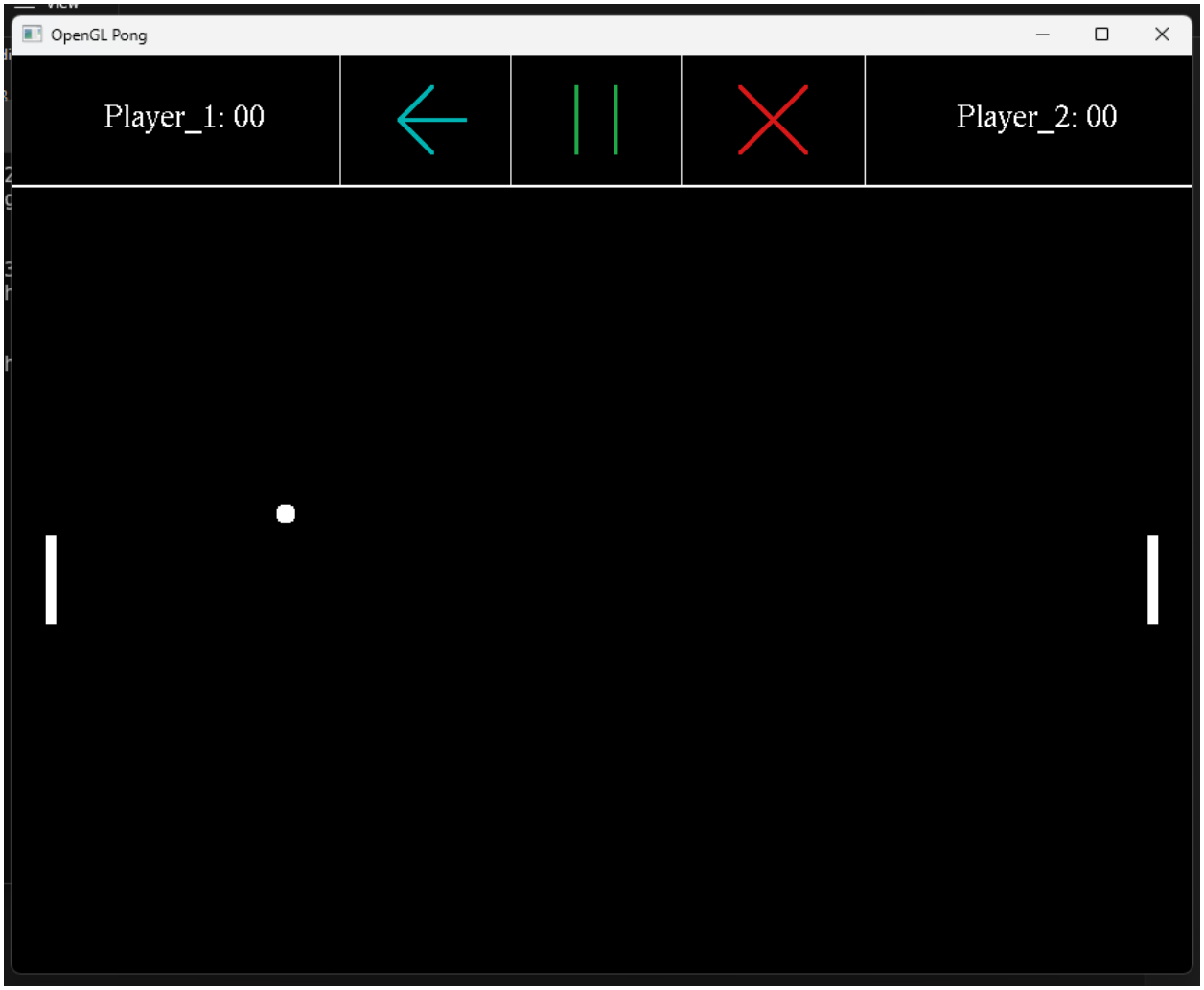
The demo gif can be found in this link -> https://0x0.st/H3eR.gif

ScreenShots:

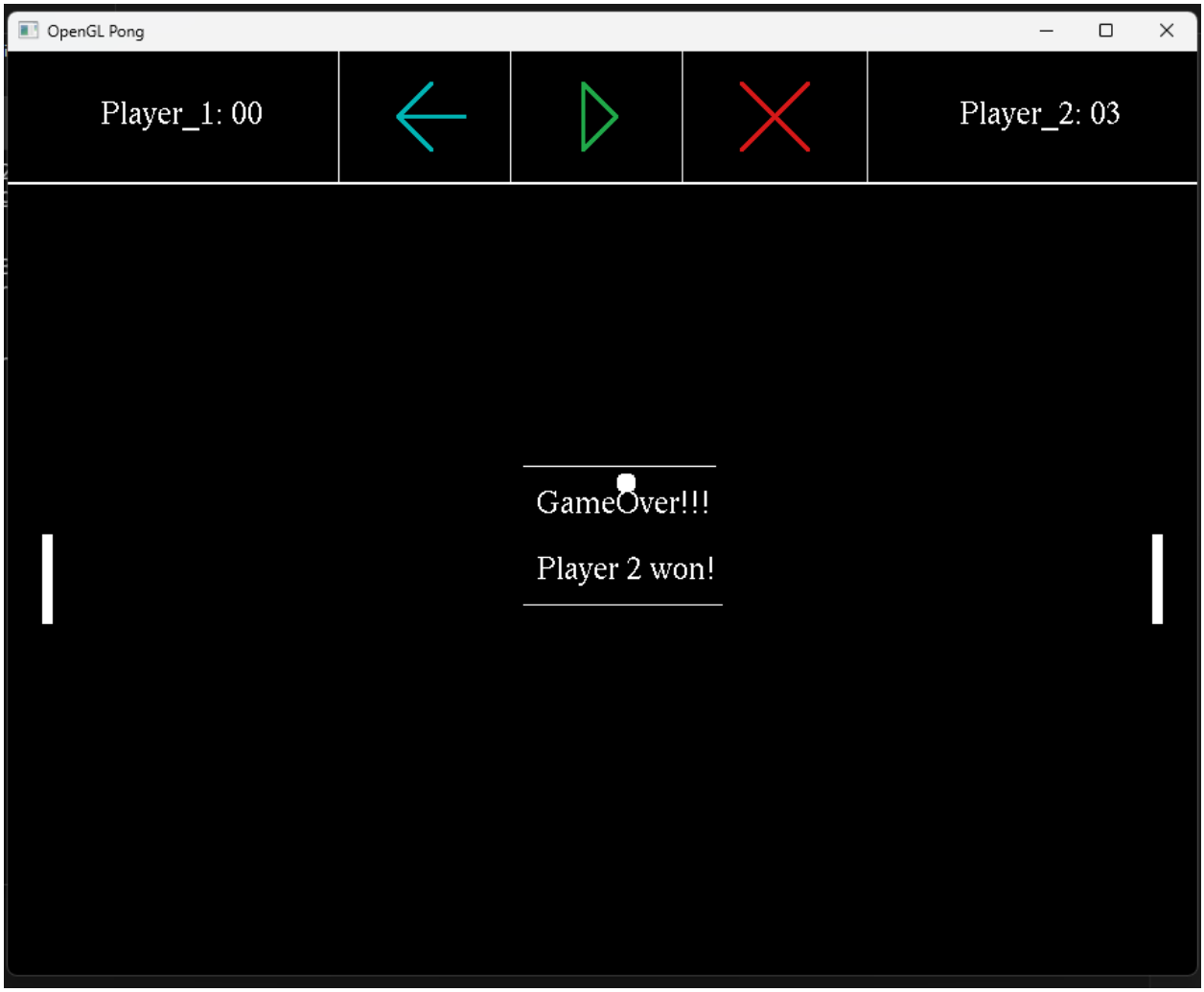
Main Menu



Game Play



Game Over



Pause