**Game File**

import pygame

import sys

from pygame.locals import \*

from Grid import \*

from Player import \*

from Enemy import \*

import random

pygame.init()

pygame.display.set\_caption("Pacman")

screen\_width = 1260

screen\_height = 744

#-----------------------------------------------Menu\_State-----------------------------------------------#

class Menu(object):

def \_\_init\_\_(self):

self.window = pygame.display.set\_mode((screen\_width, screen\_height))

self.clock = pygame.time.Clock()

self.terminate = False

self.run = True

self.load()

self.button\_list = [(155, 152), (155, 272), (155, 392), (155, 512), (155, 632)]

self.draw\_buttons()

self.events()

def \_\_del\_\_(self):

print("You have exited main menu")

def events(self):

while not self.terminate and self.run:

self.menu\_event()

self.menu\_draw()

self.menu\_update()

self.clock.tick(60)

if self.terminate:

pygame.quit()

def menu\_event(self):

pygame.time.delay(30)

for event in pygame.event.get():

mouse\_pos = pygame.mouse.get\_pos()

if event.type == pygame.MOUSEMOTION:

if self.button\_collisions(mouse\_pos):

self.hover()

self.button\_text()

else:

self.draw\_buttons()

if event.type == pygame.MOUSEBUTTONDOWN:

if self.button\_collisions(mouse\_pos) and self.button\_count == 0:

self.run = False

board = Board(self.window)

elif self.button\_collisions(mouse\_pos) and self.button\_count == 4:

self.terminate = True

if event.type == pygame.QUIT or (self.terminate == True):

self.terminate = True

if event.type == pygame.KEYDOWN:

if event.key == K\_ESCAPE:

self.terminate = True

def button\_collisions(self, mouse\_pos):

self.button\_count = -1

for pos in self.button\_list:

self.button\_count += 1

if (mouse\_pos[0] > pos[0]) and (mouse\_pos[0] < (pos[0] + 200)):

if (mouse\_pos[1] > pos[1]) and (mouse\_pos[1] < (pos[1] + 100)):

return True

def menu\_draw(self):

None

def menu\_update(self):

pygame.display.update()

def button\_text(self):

display\_list = ["Single-player", "Local-Multiplayer", "Leaderboard", "Settings", "Quit"]

height = 8

for x in display\_list:

button\_font = pygame.font.Font(None, 33)

button\_surf = button\_font.render(x, 1, (255, 255, 255))

button\_pos = [3.7 \* 45, height \* 24]

height += 5

self.window.blit(button\_surf, button\_pos)

def draw\_buttons(self):

y = 154

for x in range(0, 5):

pygame.draw.rect(self.window, (0, 0, 0), (157, y, 204, 104), 0)

pygame.draw.rect(self.window, (255, 205, 0), (155, y-2, 200, 100), 0)

y += 120

self.button\_text()

def hover(self):

for index, pos in enumerate(self.button\_list):

if self.button\_count == index:

pygame.draw.rect(self.window, (178, 143, 0), (pos[0], pos[1], 200, 100), 0)

pygame.display.update()

def load(self):

self.background = pygame.image.load("menuscreen.jpg")

self.background = pygame.transform.smoothscale(self.background, (screen\_width, screen\_height))

self.window.blit(self.background, (0, 0))

pygame.display.update()

#-----------------------------------------------Playing\_State-----------------------------------------------#

class Board(object):

def \_\_init\_\_(self, window):

self.Maze = grid

self.window = window

self.clock = pygame.time.Clock()

self.terminate = False

self.x\_coord = 0

self.y\_coord = 0

self.walls = []

self.free\_cells = []

self.free\_pos = []

self.enemy\_spawn = []

self.dots = []

self.cell\_width = 45

self.offset\_width = self.cell\_width // 2

self.cell\_height = 24

self.offset\_height = self.cell\_height // 2

########Initialization###########

self.load()

self.cells()

self.player = Player(self)

self.inky = Enemy(self, (178, 225, 255), 'L')

self.blinky = Enemy(self, (178, 225, 120), 'R')

self.pinky = Enemy(self, (93, 5, 120), 'L')

self.clyde = Enemy(self, (154, 253, 78), 'R')

self.enemy = [self.inky, self.blinky, self.pinky, self.clyde]

self.events()

def events(self):

while not self.terminate:

self.play\_event()

self.play\_draw()

self.play\_update()

self.clock.tick(60)

self.blinky.create\_matrix()

pygame.quit()

sys.exit()

def play\_event(self):

if self.player.player\_lives == 0:

self.terminate = True

for event in pygame.event.get():

if event.type == pygame.QUIT or (self.terminate == True):

self.terminate = True

if event.type == pygame.KEYDOWN:

if event.key == K\_ESCAPE:

self.terminate = True

self.player\_collision()

def play\_update(self):

self.player.update()

self.inky.update()

self.blinky.update()

self.pinky.update()

self.clyde.update()

pygame.display.update()

def play\_draw(self):

self.window.fill((0, 0, 0))

self.window.blit(self.background, (0, 0))

"""self.draw\_grid()"""

self.draw\_pops()

self.player.draw()

self.enemy\_moves()

def enemy\_moves(self):

pygame.time.delay(150)

self.inky.changeLocation(random.choice(['L', 'U', 'D', 'R']))

self.enemy\_collision(self.inky.direction, self.inky)

self.blinky.changeLocation(random.choice(['L', 'U', 'D', 'R']))

self.enemy\_collision(self.blinky.direction, self.blinky)

self.pinky.changeLocation(random.choice(['L', 'U', 'D', 'R']))

self.enemy\_collision(self.pinky.direction, self.pinky)

self.clyde.changeLocation(random.choice(['L', 'U', 'D', 'R']))

self.enemy\_collision(self.clyde.direction, self.clyde)

def draw\_grid(self):

for line in range(screen\_width // 45):

pygame.draw.line(self.window, (107, 107, 107), (line \* self.cell\_width, 0),

(line \* self.cell\_width, screen\_height))

for line in range(screen\_height // 24):

pygame.draw.line(self.window, (107, 107, 107), (0, line \* self.cell\_height),

(screen\_width, line \* self.cell\_height))

def cells(self):

for y, row in enumerate(self.Maze):

for x, cell in enumerate(row):

if cell == 0:

self.free\_pos.append((x, y))

self.free\_cells.append((self.x\_coord + self.offset\_width, self.y\_coord + self.offset\_height))

self.dots.append((self.x\_coord + self.offset\_width, self.y\_coord + self.offset\_height))

elif cell == 1:

self.walls.append((self.x\_coord + self.offset\_width, self.y\_coord + self.offset\_height))

else:

self.enemy\_spawn.append((self.x\_coord + self.offset\_width, self.y\_coord + self.offset\_height))

self.x\_coord += 45

if self.x\_coord == 1260:

self.x\_coord = 0

self.y\_coord += 24

def draw\_pops(self):

for value in self.dots:

pygame.draw.circle(self.window, (255, 215, 0), (value[0], value[1]), 5)

if len(self.dots) == 0:

for value in self.free\_cells:

self.dots.append((value[0], value[1]))

pygame.draw.circle(self.window, (255, 215, 0), (value[0], value[1]), 5)

def player\_collision(self):

keys = pygame.key.get\_pressed()

if keys[pygame.K\_LEFT]:

for tup in self.free\_cells:

if (tup[0] == self.player.x - self.cell\_width) and (tup[1] == self.player.y):

self.player.movement(-self.cell\_width, 0)

return None

if keys[pygame.K\_RIGHT]:

for tup in self.free\_cells:

if (tup[0] == self.player.x + self.cell\_width) and (tup[1] == self.player.y):

self.player.movement(self.cell\_width, 0)

return None

if keys[pygame.K\_UP]:

for tup in self.free\_cells:

if (tup[0] == self.player.x) and (tup[1] == self.player.y - self.cell\_height):

self.player.movement(0, -self.cell\_height)

return None

if keys[pygame.K\_DOWN]:

for tup in self.free\_cells:

if (tup[0] == self.player.x) and (tup[1] == self.player.y + self.cell\_height):

self.player.movement(0, self.cell\_height)

return None

def enemy\_collision(self, direction, enemy):

if direction == "L":

for tup in self.free\_cells:

if (tup[0] == enemy.x - self.cell\_width) and (tup[1] == enemy.y):

enemy.moves()

break

if direction == "R":

for tup in self.free\_cells:

if (tup[0] == enemy.x + self.cell\_width) and (tup[1] == enemy.y):

enemy.moves()

break

if direction == "U":

for tup in self.free\_cells:

if (tup[0] == enemy.x) and (tup[1] == enemy.y - self.cell\_height):

enemy.moves()

break

if direction == "D":

for tup in self.free\_cells:

if (tup[0] == enemy.x) and (tup[1] == enemy.y + self.cell\_height):

enemy.moves()

break

def load(self):

self.background = pygame.image.load("Maze.png")

self.background = pygame.transform.smoothscale(self.background, (screen\_width, screen\_height))

self.window.blit(self.background, (0, 0))

pygame.display.update()

**Player File**

import pygame

class Player(object):

def \_\_init\_\_(self, board):

self.board = board

self.x = 607

self.y = 420

self.pos = [(self.x, self.y)]

self.score = 0

self.player\_lives = 3

def movement(self, x, y):

self.x += x

self.y += y

self.pos = [self.x, self.y]

def draw(self):

pygame.draw.circle(self.board.window, (255, 255, 0), (self.x, self.y), 8)

def score\_system(self):

score\_font = pygame.font.Font(None, 50)

score\_surf = score\_font.render(str(self.score), 1, (255, 255, 255))

score\_pos = [3 \* self.board.cell\_width, 13 \* self.board.cell\_height]

self.board.window.blit(score\_surf, score\_pos)

def lives\_system(self):

lives\_font = pygame.font.Font(None, 50)

lives\_surf = lives\_font.render(str(self.player\_lives), 1, (255, 255, 255))

lives\_pos = [3 \* self.board.cell\_width, 17 \* self.board.cell\_height]

self.board.window.blit(lives\_surf, lives\_pos)

def update(self):

score\_font = pygame.font.Font(None, 50)

score\_surf = score\_font.render("Score", 1, (255, 255, 255))

score\_pos = [2 \* self.board.cell\_width, 11.5 \* self.board.cell\_height]

lives\_font = pygame.font.Font(None, 50)

lives\_surf = lives\_font.render("Lives", 1, (255, 255, 255))

lives\_pos = [2 \* self.board.cell\_width, 15.5 \* self.board.cell\_height]

self.board.window.blit(lives\_surf, lives\_pos)

self.board.window.blit(score\_surf, score\_pos)

for tup2 in self.board.dots:

if (tup2[0] == self.x) and (tup2[1] == self.y):

self.board.dots.remove(tup2)

self.score += 1

self.score\_system()

break

self.score\_system()

for tup2 in self.board.enemy:

if (tup2.x == self.x) and (tup2.y == self.y):

self.player\_lives -= 1

self.board.player.x = 607

self.board.player.y = 420

break

self.lives\_system()

**Enemy File**

import pygame

class Enemy:

def \_\_init\_\_(self, board, colour, direction):

self.board = board

self.x = 607

self.y = 276

self.direction = direction

self.colour = colour

self.intersections = []

self.matrix = []

self.matrix\_equivalent = {}

def changeLocation(self, direction):

self.direction = direction

def moves(self):

if self.direction == 'L':

self.x -= self.board.cell\_width

pygame.draw.circle(self.board.window, (self.colour), (self.x, self.y), 8)

if self.direction == 'R':

self.x += self.board.cell\_width

pygame.draw.circle(self.board.window, (self.colour), (self.x, self.y), 8)

if self.direction == 'U':

self.y -= self.board.cell\_height

pygame.draw.circle(self.board.window, (self.colour), (self.x, self.y), 8)

if self.direction == 'D':

self.y += self.board.cell\_height

pygame.draw.circle(self.board.window, (self.colour), (self.x, self.y), 8)

def update(self):

pygame.draw.circle(self.board.window, (self.colour), (self.x, self.y), 8)

pygame.display.update()

"---------------------------------------------------------------------"

def create\_matrix(self):

for value in self.board.free\_pos:

x1 = (value[0] + 1, value[1])

x2 = (value[0] - 1, value[1])

y1 = (value[0], value[1] + 1)

y2 = (value[0], value[1] - 1)

if x1 in self.board.free\_pos or x2 in self.board.free\_pos:

if y1 in self.board.free\_pos or y2 in self.board.free\_pos:

if value not in self.intersections:

self.intersections.append((value[0], value[1]))

index = 0

for value in self.intersections:

self.matrix\_equivalent[value] = index

index += 1

for x in range(0, len(self.intersections)):

self.matrix.append([])

for y in range(0, len(self.intersections)):

self.matrix[x].append(float("inf"))

for index, value in enumerate(self.intersections):

for index2, value2 in enumerate(self.intersections):

if value != value2:

if value[0] == value2[0]:

weight = abs(value2[1] - value[1])

self.matrix[index][index2] = weight

if value[1] == value2[1]:

weight = abs(value2[0] - value[0])

self.matrix[index][index2] = weight

print(self.matrix)

print(self.matrix\_equivalent)

def neighbours\_location(self):

None

def path\_finding(self):

None

def matrix\_values(self):

None

def board\_location(self):

None

**Power-Up File**

import pygame

import random

class Items(object):

def \_\_init\_\_(self, board):

self.name\_power\_ups = ["invisibility"]

self.board = board

self.power\_ups = {}

def spawn(self):

for name in self.name\_power\_ups:

location = random.choice(self.board.free\_cells)

self.power\_ups[name] = location

pygame.draw.circle(self.board.window, (255, 215, 0), (location[0], location[1]), 2)

**Main File**

from Game import \*

from Enemy import \*

from Grid import \*

from Player import \*

from Powerup import \*

from pygame import \*

if \_\_name\_\_ == '\_\_main\_\_':

menu = Menu(False)

board = Board(False)

def load(state):

if state == "Menu":

menu.load()

if state == "Play":

board.load()

def game\_loop(state, terminate):

while not terminate and state == "Menu":

menu.menu\_event()

menu.menu\_draw()

menu.menu\_update()

if menu.menu\_event() == "Play":

state = "Play"

load("Play")

if menu.terminate:

terminate = True

while not terminate and state == "Play":

board.play\_event()

board.play\_draw()

board.play\_update()

if board.terminate:

terminate = True

board.inky.create\_matrix()

pygame.quit()

load("Menu")

game\_loop("Menu", False)