```
import pygame, random

#define quit variable
gameQuit = False
global count, win
count = 0
win = False

global newDirection, pacSpeed, debug
newDirection = -1
pacSpeed = 1
debug = False

#game framerate
gameFramerate = 120
frames = 0
clockCounter = pygame.time.Clock()

#initialises pygame
pygame.init()

#defines GUI variable (res = 1024x1024)
GUI = pygame.display.set_mode((1024, 1024))

with open("levels/level1.txt", "r") as file:
    lineCount = sum(1 for line in file) - 1

levelList = []
with open("levels/level1.txt", "r") as file:
    for i in range(16):
            tempList = []
            with open("levels/level1.txt", "r") as file:
                for j in range(16):
                    tempList.append(str(file.readline())[i])
            levelList.append(tempList)

global firstCorner, secondCorner, thirdCorner, fourthCorner
topRow, flag = 0, True
bottomRow = 0
firstCorner, flag1 = [0, 0], True
secondCorner = [0, 0]
thirdCorner, flag2 = [0, 0], True
fourthCorner = [0, 0]

for i in range(len(levelList)): #finds the top and bottom rows of map
    for j in range(len(levelList[i])):
        if int(levelList[i][j]) == 0 and flag and 0 < i < 15:
            topRow, flag = j, False
        if int(levelList[i][j]) == 0 and 0 < i < 15:
            bottomRow = j

for i in range(len(levelList)): #finds the corners of the map
    for j in range(len(levelList[i])):
        if int(levelList[i][j]) == 0 and j == topRow and flag1 and 0 < i < 15: # top left
            firstCorner, flag1 = [i, j], False
        if int(levelList[i][j]) == 0 and j == topRow and 0 < i < 15: # top right
            secondCorner = [i, j]
        if int(levelList[i][j]) == 0 and j == bottomRow and flag2 and 0 < i < 15: # bottom left
            thirdCorner, flag2 = [i, j], False
```

```
        if int(levelList[i][j]) == 0 and 0 < i < 15: # bottom right
            fourthCorner = [i, j]

levelList[firstCorner[0]][firstCorner[1]] = "3" #sets all four corners to 3. Signifying power-up
levelList[secondCorner[0]][secondCorner[1]] = "3"
levelList[thirdCorner[0]][thirdCorner[1]] = "3"
levelList[fourthCorner[0]][fourthCorner[1]] = "3"

print("TL: " + str(firstCorner))
print("TR: " + str(secondCorner))
print("BL: " + str(thirdCorner))
print("BR: " + str(fourthCorner))
if debug:
    print(levelList)

#imports images for Pac-Man animation and maze background
pacMan1 = pygame.transform.scale(pygame.image.load("assets/images/pacman1.png"), (64, 64))
pacMan2 = pygame.transform.scale(pygame.image.load("assets/images/pacman2.png"), (64, 64))
pacMan3 = pygame.transform.scale(pygame.image.load("assets/images/pacman3.png"), (64, 64))
redGhost = pygame.transform.scale(pygame.image.load("assets/images/redGhost.png"), (64, 64))
blueGhost = pygame.transform.scale(pygame.image.load("assets/images/blueGhost.png"), (64, 64))
pinkGhost = pygame.transform.scale(pygame.image.load("assets/images/pinkGhost.png"), (64, 64))
orangeGhost = pygame.transform.scale(pygame.image.load("assets/images/orangeGhost.png"), (64, 64))
maze = pygame.image.load("assets/images/pacManMaze.png").convert()
wall = pygame.rect.Rect(20, 20, 20, 400)

#Pac-Man Object
class Pacman():
    def __init__(self, posX, posY):
        #default lives, starting position, and animation markers
        self.lives = 3
        self.posX, self.posY = posX, posY
        self.ogPosX, self.ogPosY = posX, posY
        self.image = pacMan1
        self.currentImage = 1
        self.direction = 0                  # 0 - Left, 1 - Right, 2 - Up, 3 - Down
        self.count = 0
        self.dirCount = 0
    def render(self): #renders Pac-Man
        if self.count % 10 == 0:
            match self.currentImage:
                case 1:
                    self.image = pacMan2
                    self.currentImage = 2
                case 2:
                    self.image = pacMan3
                    self.currentImage = 3
                case 3:
                    self.image = pacMan2
                    self.currentImage = 4
                case 4:
                    self.image = pacMan1
                    self.currentImage = 1
        if self.count % 100 == 0 and debug:
            print(self.posX, self.posY)
        match self.direction: #sets Pac-Man orientation
            case 0:
                GUI.blit(pygame.transform.rotate(self.image, 180), (self.posX, self.posY))
            case 1:
```

```
            GUI.blit(self.image, (self.posX, self.posY))
        case 2:
            GUI.blit(pygame.transform.rotate(self.image, 90), (self.posX, self.posY))
        case 3:
            GUI.blit(pygame.transform.rotate(self.image, 270), (self.posX, self.posY))
    self.count += 1
    if self.posX > 1024 and self.direction == 1: #loops sprite back round
        self.posX = -40
    elif self.posX < -40 and self.direction == 0:
        self.posX = 1064
    elif self.posY > 1024 and self.direction == 3:
        self.posY = -40
    elif self.posY < -40 and self.direction == 2:
        self.posY = 1064
def testMove(self): #detects key presses
    dirTime = 32
    global newDirection, pacSpeed, debug
    if self.dirCount > 0:
        self.dirCount -= 1
    keys = pygame.key.get_pressed()
    if keys[pygame.K_a] or keys[pygame.K_LEFT]:
        newDirection, self.dirCount = 0, dirTime
    if keys[pygame.K_d] or keys[pygame.K_RIGHT]:
        newDirection, self.dirCount = 1, dirTime
    if keys[pygame.K_w] or keys[pygame.K_UP]:
        newDirection, self.dirCount = 2, dirTime
    if keys[pygame.K_s] or keys[pygame.K_DOWN]:
        newDirection, self.dirCount = 3, dirTime
    square = 1024 // lineCount
    posX = int((self.posX) // square)
    posY = int(self.posY // square)
    match newDirection:
        case 0:
            #left
            if 0 < self.posX < 960 and 0 < self.posY < 960:
                if self.posY / square == posY and self.dirCount > 0 and
                int(levelList[int((self.posX - 1) // square)][posY])  != 1:
                    self.direction = 0
        case 1:
            #right
            if 0 < self.posX < 960 and 0 < self.posY < 960:
                if self.posY / square == posY and self.dirCount > 0 and
                int(levelList[int((self.posX) // square) + 1][posY] ) != 1:
                    self.direction = 1
        case 2:
            #up
            if 0 < self.posX < 960 and 0 < self.posY < 960:
                if self.posX / square == posX and self.dirCount > 0 and int(levelList[posX]
                [int((self.posY - 1) // square)] ) != 1:
                    self.direction = 2
        case 3:
            #down
            if 0 < self.posX < 960 and 0 < self.posY < 960:
                if self.posX / square == posX and self.dirCount > 0 and int(levelList[posX]
                [int((self.posY) // square) + 1]) != 1:
                    self.direction = 3
    match self.direction: #moves Pac-Man in chosen direction
        case 0:
            #left
```

```
            if 0 < self.posX < 960 and 0 < self.posY < 960:
                if int(levelList[int((self.posX - pacSpeed) // square)][posY])  != 1:
                    self.posX = round((self.posX - pacSpeed), 1)
                if (int(levelList[int((self.posX - 1) // square)][posY]) == 0 or
                int(levelList[int((self.posX - 1) // square)][posY]) == 3) and self.posX ==
                (((self.posX - 1) // square)*square + 7):
                    levelList[int((self.posX - 1) // square)][posY] = "2"
            else:
                self.posX = round((self.posX - pacSpeed), 1)
        case 1:
            #right
            if 0 < self.posX < 960 and 0 < self.posY < 960:
                if int(levelList[int((self.posX) // square) + pacSpeed][posY] ) != 1:
                    self.posX = round((self.posX + pacSpeed), 1)
                try:
                    if (int(levelList[int((self.posX) // square) + 1][posY]) == 0 or
                    int(levelList[int((self.posX) // square) + 1][posY]) == 3) and self.posX ==
                    (50 + (self.posX // square)*square):
                        levelList[int((self.posX) // square) + 1][posY] = "2"
                except:
                    pass
            else:
                self.posX = round((self.posX + pacSpeed), 1)
        case 2:
            #up
            if 0 < self.posX < 960 and 0 < self.posY < 960:
                if int(levelList[posX][int((self.posY - pacSpeed) // square)] ) != 1:
                    self.posY = round((self.posY - pacSpeed), 1)
                if (int(levelList[posX][int((self.posY - 1) // square)]) == 0 or
                int(levelList[posX][int((self.posY - 1) // square)]) == 3) and self.posY ==
                (((self.posY - 1) // square)*square + 7):
                    levelList[posX][int((self.posY - 1) // square)] = "2"
            else:
                self.posY = round((self.posY - pacSpeed), 1)
        case 3:
            #down
            if 0 < self.posX < 960 and 0 < self.posY < 960:
                if int(levelList[posX][int((self.posY) // square) + pacSpeed]) != 1:
                    self.posY = round((self.posY + pacSpeed), 1)
                try:
                    if (int(levelList[posX][int((self.posY) // square) + 1]) == 0 or
                    int(levelList[posX][int((self.posY) // square) + 1]) == 3) and self.posY ==
                    (50 + (self.posY // square)*square):
                        levelList[posX][int((self.posY) // square) + 1] = "2"
                except:
                    pass
            else:
                self.posY = round((self.posY + pacSpeed), 1)
    if self.count % 100 == 0 and debug:
        print("X:",self.posX, "Y:", self.posY,"(X,Y):",posX,posY)
def setOgPos(self):
    self.posX, self.posY = self.ogPosX, self.ogPosY

class Ghost(): #ghost class
    def __init__(self, image, posX, posY):
        self.image = image
        self.posX = posX
        self.posY = posY
        self.ogPosX, self.ogPosY = posX, posY
```

```
        self.currentImage = 0
        self.count = 0
        self.dirChange = True
        self.direction = 0
        self.oldX = 0
        self.oldY = 0
        self.contDis, self.leftDis, self.rightDis, self.upDis, self.downDis = 0, 0, 0, 0, 0
    def move(self):
        square = 1024 // lineCount #allign to grid
        posX = int((self.posX) // square)
        posY = int(self.posY // square)

        if self.direction == 0: #changes direction to another random available direction upon hitting
        an intersection
            self.contDis = ((self.posX - 1 - pacMan.posX)**2 + (self.posY - pacMan.posX)**2)**0.5
        elif self.direction == 1:
            self.contDis = ((self.posX + 1 - pacMan.posX)**2 + (self.posY - pacMan.posX)**2)**0.5
        elif self.direction == 2:
            self.contDis = ((self.posX - pacMan.posX)**2 + (self.posY - 1 - pacMan.posX)**2)**0.5
        elif self.direction == 3:
            self.contDis = ((self.posX - pacMan.posX)**2 + (self.posY + 1 - pacMan.posX)**2)**0.5
        if self.count % 100 == 0:
            print(self.contDis)
        canGoLeft, canGoRight, canGoUp, canGoDown = False, False, False, False
        if self.posY / square == posY and int(levelList[int((self.posX - 1) // square)][posY])  != 1
        and self.direction != 0 and self.direction != 1:
            canGoLeft = True
            self.leftDis = ((self.posX - 64 - pacMan.posX)**2 + (self.posY - pacMan.posX)**2)**0.5
        if self.posY / square == posY and int(levelList[int((self.posX) // square) + 1][posY]) != 1
        and self.direction != 0 and self.direction != 1:
            canGoRight = True
            self.rightDis = ((self.posX + 64 - pacMan.posX)**2 + (self.posY - pacMan.posX)**2)**0.5
        if self.posX / square == posX and int(levelList[posX][int((self.posY - 1) // square)]) != 1
        and self.direction != 2 and self.direction != 3:
            canGoUp = True
            self.upDis = ((self.posX - pacMan.posX)**2 + (self.posY - 64 - pacMan.posX)**2)**0.5
        if self.posX / square == posX and int(levelList[posX][int((self.posY) // square) + 1]) != 1
        and self.direction != 2 and self.direction != 3:
            canGoDown = True
            self.downDis = ((self.posX - pacMan.posX)**2 + (self.posY + 64 - pacMan.posX)**2)**0.5

        disList = [self.leftDis, self.rightDis, self.upDis, self.downDis, self.contDis]
        flag = True
        while flag:
            for i in range(len(disList)):
                flag = False
                try:
                    if disList[i] > disList[i + 1]:
                        disList[i], disList[i + 1] = disList[i + 1], disList[i]
                        flag = True
                except:
                    pass

        for i in range(len(disList)):
            if disList[i] == self.leftDis and canGoLeft:
                self.newdirection = 0
                break
            elif disList[i] == self.rightDis and canGoRight:
                self.newdirection = 1
```

```
            break
        elif disList[i] == self.upDis and canGoUp:
            self.newdirection = 2
            break
        elif disList[i] == self.downDis and canGoDown:
            self.newdirection = 3
            break
        elif disList[i] == self.contDis:
            break

    #if self.posX == self.oldX and self.posY == self.oldY:
    #    self.dirChange = True
    self.oldX = self.posX
    self.oldY = self.posY
    if self.dirChange:
        self.newdirection = random.randint(0, 3)
        self.dirChange = False
    match self.newdirection:
        case 0:
            #left
            if 0 < self.posX < 960 and 0 < self.posY < 960:
                if self.posY / square == posY and int(levelList[int((self.posX - 1) // square)]
                [posY])  != 1:
                    self.direction = 0 #direction change
        case 1:
            #right
            if 0 < self.posX < 960 and 0 < self.posY < 960:
                if self.posY / square == posY and int(levelList[int((self.posX) // square) + 1]
                [posY]) != 1:
                    self.direction = 1 #direction change
        case 2:
            #up
            if 0 < self.posX < 960 and 0 < self.posY < 960:
                if self.posX / square == posX and int(levelList[posX][int((self.posY - 1) //
                square)]) != 1:
                    self.direction = 2 #direction change
        case 3:
            #down
            if 0 < self.posX < 960 and 0 < self.posY < 960:
                if self.posX / square == posX and int(levelList[posX][int((self.posY) // square)
                + 1]) != 1:
                    self.direction = 3 #direction change
    match self.direction: #moves Ghost in chosen direction
        case 0:
            #left
            if 0 < self.posX < 960 and 0 < self.posY < 960:
                if int(levelList[int((self.posX - pacSpeed) // square)][posY])  != 1:
                    self.posX = round((self.posX - pacSpeed), 1)
            else:
                self.posX = round((self.posX - pacSpeed), 1)
        case 1:
            #right
            if 0 < self.posX < 960 and 0 < self.posY < 960:
                if int(levelList[int((self.posX) // square) + pacSpeed][posY] ) != 1:
                    self.posX = round((self.posX + pacSpeed), 1)
            else:
                self.posX = round((self.posX + pacSpeed), 1)
        case 2:
            #up
```

```
                if 0 < self.posX < 960 and 0 < self.posY < 960:
                    if int(levelList[posX][int((self.posY - pacSpeed) // square)] ) != 1:
                        self.posY = round((self.posY - pacSpeed), 1)
                else:
                    self.posY = round((self.posY - pacSpeed), 1)
            case 3:
                #down
                if 0 < self.posX < 960 and 0 < self.posY < 960:
                    if int(levelList[posX][int((self.posY) // square) + pacSpeed]) != 1:
                        self.posY = round((self.posY + pacSpeed), 1)
                else:
                    self.posY = round((self.posY + pacSpeed), 1)
        self.count += 1
    def render(self):
        GUI.blit(self.image, (self.posX, self.posY))
    def checkCollide(self):
        if -40 < pacMan.posX - self.posX < 40 and -40 < pacMan.posY - self.posY < 40: #checks if
position is colliding with pacman
            pacMan.lives -= 1
            pacMan.setOgPos()
            self.allGhostRecall()
            print("life lost")
    def setOgPos(self):
        self.posX, self.posY = self.ogPosX, self.ogPosY
    def allGhostRecall(self): #sets all ghosts to original spawn locations
        blinky.setOgPos()
        inky.setOgPos()
        pinky.setOgPos()
        clyde.setOgPos()

pacMan = Pacman(512, 768)
blinky = Ghost(redGhost, 448, 448)
pinky = Ghost(pinkGhost, 448, 448)
inky = Ghost(blueGhost, 448, 448)
clyde = Ghost(orangeGhost, 448, 448)

def renderBackG(): #draws background
    global debug
    keys = pygame.key.get_pressed()
    grid = 0
    if keys[pygame.K_HASH]:
        debug = not debug
    if keys[pygame.K_q]  and debug:
        GUI.fill((255,255,255))
        grid = 1
    col = 100
    square = int(1024 // lineCount)
    for i in range(len(levelList)):
        for j in range(len(levelList[i])):
            if int(levelList[i][j]) == 1:
                col += 1
                pygame.draw.rect(GUI, (0, 0, col), (i*square, j*square, square - grid , square -grid
                ))
            else:
                pygame.draw.rect(GUI, (0, 0, 0), (i*square, j*square, square -grid , square -grid ))

def renderPacMan(): #makes sprite appear and move
    pacMan.testMove()
    pacMan.render()
```

```
def renderGhosts(): #ghost render / logic loop
    blinky.move()
    blinky.render()
    blinky.checkCollide()
    pinky.move()
    pinky.render()
    pinky.checkCollide()
    inky.move()
    inky.render()
    inky.checkCollide()
    clyde.move()
    clyde.render()
    clyde.checkCollide()

def createDots(): #generates the pellets on the map where there are no wall tiles
    global firstCorner, secondCorner, thirdCorner, fourthCorner
    square = int(1024 // lineCount)
    for i in range(len(levelList)):
        for j in range(len(levelList[i])):
            if int(levelList[i][j]) == 0 and 0 < i < 15:
                pygame.draw.rect(GUI, (255, 255, 255), (i*square + 24, j*square + 24, 8 , 8))
            elif int(levelList[i][j]) == 3 and 0 < i < 15:
                pygame.draw.rect(GUI, (255, 255, 255), (i*square + 24, j*square + 24, 16, 16))

def countRemainingDots(): #counts remaining pellets to check if player has won
    count = 0
    square = int(1024 // lineCount)
    for i in range(len(levelList)):
        for j in range(len(levelList[i])):
            if (int(levelList[i][j]) == 0 or int(levelList[i][j]) == 3) and 0 < i < 15:
                count += 1
    if count <= 0:
        return False

while True: #main game loop
    clockCounter.tick(gameFramerate) #slows game to framerate
    pygameEvents = pygame.event.get() #get all pyagme events
    for event in pygameEvents:
        if event.type == pygame.QUIT:
            gameQuit = True
    renderBackG()
    if frames % 10 == 0:
        if countRemainingDots() == False:
            win = True
    createDots()
    renderPacMan()
    renderGhosts()
    pygame.display.flip()
    if gameQuit: #breaks out of loop
        break
    if win:
        gameQuit = True


pygame.quit() #closes game
```