

Setting Up Pellets

At this point, the pellets are in the maze and Pacman eats them. Nothing happens when a power pellet is eaten besides it disappearing.

Run.py

```
import pygame
from pygame.locals import *
from constants import *
from pacman import Pacman
from nodes import NodeGroup
from pellets import PelletGroup

class GameController(object):
    def __init__(self):
        pygame.init()
        self.screen = pygame.display.set_mode(SCREENSIZE, 0, 32)
        self.background = None
        self.clock = pygame.time.Clock()

    def setBackground(self):
        self.background = pygame.surface.Surface(SCREENSIZE).convert()
        self.background.fill(BLACK)

    def startGame(self):
        self.setBackground()
        self.nodes = NodeGroup("maze01.txt")
        self.nodes.setPortalPair((0,17), (27,17))
        self.pacman = Pacman(self.nodes.getStartTempNode())
        self.pellets = PelletGroup("maze01.txt.")

    def update(self):
        dt = self.clock.tick(30) / 1000.0
        self.pacman.update(dt)
        self.pellets.update(dt)
        self.checkEvents()
        self.checkPelletEvents()
        self.render()

    def checkEvents(self):
        for event in pygame.event.get():
            if event.type == QUIT:
                exit()

    def checkPelletEvents(self):
        pellet = self.pacman.eatPellets(self.pellets.pelletList)
        if pellet:
            self.pellets.numEaten += 1
            self.pellets.pelletList.remove(pellet)
```

```

def render(self):
    self.screen.blit(self.background, (0,0))
    self.nodes.render(self.screen)
    self.pellets.render(self.screen)
    self.pacman.render(self.screen)
    pygame.display.update()

if __name__ == "__main__":
    game = GameController()
    game.startGame()
    while True:
        game.update()

```

Nodes.py

```

import pygame
from vector import Vector2
from constants import *
import numpy as np

class Node(object):
    def __init__(self, x, y):
        self.position = Vector2(x, y)
        self.neighbors = {UP:None, DOWN:None, LEFT:None, RIGHT:None, PORTAL:None}

    def render(self, screen):
        for n in self.neighbors.keys():
            if self.neighbors[n] is not None:
                line_start = self.position.asTuple()
                line_end = self.neighbors[n].position.asTuple()
                pygame.draw.line(screen, WHITE, line_start, line_end, 4)
                pygame.draw.circle(screen, RED, self.position.asInt(), 12)

class NodeGroup(object):
    def __init__(self, level):
        self.level = level
        self.nodesLUT = {}
        self.nodeSymbols = ['+', 'P', 'n']
        self.pathSymbols = ['.', '-', '|', 'p']
        data = self.readMazeFile(level)
        self.createNodeTable(data)
        self.connectHorizontally(data)
        self.connectVertically(data)

    def render(self, screen):
        for node in self.nodesLUT.values():
            node.render(screen)

```

```

def readMazeFile(self, textfile):
    return np.loadtxt(textfile, dtype='<U1')

def createNodeTable(self, data, xoffset=0, yoffset=0):
    for row in list(range(data.shape[0])):
        for col in list(range(data.shape[1])):
            if data[row][col] in self.nodeSymbols:
                x, y = self.constructKey(col+xoffset, row+yoffset)
                self.nodesLUT[(x, y)] = Node(x, y)

def constructKey(self, x, y):
    return x * TILEWIDTH, y * TILEHEIGHT

def connectHorizontally(self, data, xoffset=0, yoffset=0):
    for row in list(range(data.shape[0])):
        key = None
        for col in list(range(data.shape[1])):
            if data[row][col] in self.nodeSymbols:
                if key is None:
                    key = self.constructKey(col+xoffset, row+yoffset)
                else:
                    otherkey = self.constructKey(col+xoffset, row+yoffset)
                    self.nodesLUT[key].neighbors[RIGHT] = self.nodesLUT[otherkey]
                    self.nodesLUT[otherkey].neighbors[LEFT] = self.nodesLUT[key]
                    key = otherkey
            elif data[row][col] not in self.pathSymbols:
                key = None

def connectVertically(self, data, xoffset=0, yoffset=0):
    dataT = data.transpose()
    for col in list(range(dataT.shape[0])):
        key = None
        for row in list(range(dataT.shape[1])):
            if dataT[col][row] in self.nodeSymbols:
                if key is None:
                    key = self.constructKey(col+xoffset, row+yoffset)
                else:
                    otherkey = self.constructKey(col+xoffset, row+yoffset)
                    self.nodesLUT[key].neighbors[DOWN] = self.nodesLUT[otherkey]
                    self.nodesLUT[otherkey].neighbors[UP] = self.nodesLUT[key]
                    key = otherkey
            elif dataT[col][row] not in self.pathSymbols:
                key = None

def getNodeFromPixels(self, xpixel, ypixel):
    if (xpixel, ypixel) in self.nodesLUT.keys():
        return self.nodesLUT[(xpixel, ypixel)]
    return None

def getNodeFromTiles(self, col, row):
    x, y = self.constructKey(col, row)
    if (x, y) in self.nodesLUT.keys():

```

```

        return self.nodesLUT[(x, y)]
    return None

def getStartTempNode(self):
    nodes = list(self.nodesLUT.values())
    return nodes[0]

def setPortalPair(self, pair1, pair2):
    key1 = self.constructKey(*pair1)
    key2 = self.constructKey(*pair2)
    if key1 in self.nodesLUT.keys() and key2 in self.nodesLUT.keys():
        self.nodesLUT[key1].neighbors[PORTAL] = self.nodesLUT[key2]
        self.nodesLUT[key2].neighbors[PORTAL] = self.nodesLUT[key1]

```

Pacman.py

```

import pygame
from pygame.locals import *
from vector import Vector2
from constants import *

class Pacman(object):
    def __init__(self, node):
        self.name = PACMAN
        self.position = Vector2(200, 400)
        self.directions = {STOP:Vector2(), UP:Vector2(0,-1), DOWN:Vector2(0,1), LEFT:Vector2(-1,0),
RIGHT:Vector2(1,0)}
        self.direction = STOP
        self.speed = 100
        self.radius = 10
        self.color = YELLOW
        self.node = node
        self.setPosition()
        self.target = node
        self.collideRadius = 5

    def setPosition(self):
        self.position = self.node.position.copy()

    def update(self, dt):
        self.position += self.directions[self.direction]*self.speed*dt
        direction = self.getValidKey()
        if self.overshotTarget():
            self.node = self.target
            if self.node.neighbors[PORTAL] is not None:
                self.node = self.node.neighbors[PORTAL]
            self.target = self.getNewTarget(direction)
            if self.target is not self.node:
                self.direction = direction
            else:
                self.target = self.getNewTarget(self.direction)

        if self.target is self.node:

```

```

        self.direction = STOP
        self.setPosition()
    else:
        if self.oppositeDirection(direction):
            self.reverseDirection()

def eatPellets(self, pelletList):
    for pellet in pelletList:
        d = self.position - pellet.position
        dSquared = d.magnitudeSquared()
        rSquared = (pellet.radius+self.collideRadius)**2
        if dSquared <= rSquared:
            return pellet
    return None

def overshotTarget(self):
    if self.target is not None:
        vec1 = self.target.position - self.node.position
        vec2 = self.position - self.node.position
        node2Target = vec1.magnitudeSquared()
        node2Self = vec2.magnitudeSquared()
        return node2Self >= node2Target
    return False

def validDirection(self, direction):
    if direction is not STOP:
        if self.node.neighbors[direction] is not None:
            return True
    return False

def reverseDirection(self):
    self.direction *= -1
    temp = self.node
    self.node = self.target
    self.target = temp

def oppositeDirection(self, direction):
    if direction is not STOP:
        if direction == self.direction * -1:
            return True
    return False

def getNewTarget(self, direction):
    if self.validDirection(direction):
        return self.node.neighbors[direction]
    return self.node

def getValidKey(self):
    key_pressed = pygame.key.get_pressed()
    if key_pressed[K_UP]:
        return UP
    if key_pressed[K_DOWN]:
        return DOWN

```

```

    if key_pressed[K_LEFT]:
        return LEFT
    if key_pressed[K_RIGHT]:
        return RIGHT
    return STOP

def render(self, screen):
    p = self.position.asInt()
    pygame.draw.circle(screen, self.color, p, self.radius)

```

Pellets.py

```

import pygame
from vector import Vector2
from constants import *
import numpy as np

class Pellet(object):
    def __init__(self, row, column):
        self.name = PELLET
        self.position = Vector2(column*TILEWIDTH, row*TILEHEIGHT)
        self.color = WHITE
        self.radius = int(4 * TILEWIDTH / 16)
        self.collideRadius = int(4 * TILEWIDTH / 16)
        self.points = 10
        self.visible = True

    def render(self, screen):
        if self.visible:
            p = self.position.asInt()
            pygame.draw.circle(screen, self.color, p, self.radius)

class PowerPellet(Pellet):
    def __init__(self, row, column):
        Pellet.__init__(self, row, column)
        self.name = POWERPELLET
        self.radius = int(8 * TILEWIDTH / 16)
        self.points = 50
        self.flashTime = 0.2
        self.timer = 0

    def update(self, dt):
        self.timer += dt
        if self.timer >= self.flashTime:
            self.visible = not self.visible
            self.timer = 0

class PelletGroup(object):
    def __init__(self, pelletfile):
        self.pelletList = []
        self.powerpellets = []

```

```

self.createPelletList(pelletfile)
self.numEaten = 0

def update(self, dt):
    for powerpellet in self.powerpellets:
        powerpellet.update(dt)

def createPelletList(self, pelletfile):
    data = self.readPelletfile(pelletfile)
    for row in range(data.shape[0]):
        for col in range(data.shape[1]):
            if data[row][col] in ['. ', '+']:
                self.pelletList.append(Pellet(row, col))
            elif data[row][col] in ['P', 'p']:
                pp = PowerPellet(row, col)
                self.pelletList.append(pp)
                self.powerpellets.append(pp)

def readPelletfile(self, textfile):
    return np.loadtxt(textfile, dtype='<U1')

def isEmpty(self):
    if len(self.pelletList) == 0:
        return True
    return False

def render(self, screen):
    for pellet in self.pelletList:
        pellet.render(screen)

```

Constants.py

```

TILEWIDTH = 16
TILEHEIGHT = 16
NROWS = 36
NCOLS = 28
SCREENWIDTH = NCOLS*TILEWIDTH
SCREENHEIGHT = NROWS*TILEHEIGHT
SCREENSIZE = (SCREENWIDTH, SCREENHEIGHT)
BLACK = (0, 0, 0)

```

```

YELLOW = (255, 255, 0)

```

```

STOP = 0
UP = 1
DOWN = -1
LEFT = 2
RIGHT = -2

```

```

PACMAN = 0
PELLET = 1
POWERPELLET = 2

```

WHITE = (255, 255, 255)

RED = (255, 0, 0)

PORTAL = 3