

Starting Positions

At this point, Pacman and the ghosts all start on their assigned nodes. Blinky and Pinky start moving right off the bat. Inky and Clyde circle around in the ghost home before finally coming out. Before adding that 'self.direction' line in the init of pacman.py, he's still. With it, he starts moving off the bat too.

Run.py

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

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))
        homekey = self.nodes.createHomeNodes(11.5, 14)
        self.nodes.connectHomeNodes(homekey, (12,14), LEFT)
        self.nodes.connectHomeNodes(homekey, (15,14), RIGHT)
        self.pacman = Pacman(self.nodes.getNodeFromTiles(15, 26))
        self.pellets = PelletGroup("maze01.txt.")
        self.ghosts = GhostGroup(self.nodes.getStartTempNode(), self.pacman)
        self.ghosts.blinky.setStartNode(self.nodes.getNodeFromTiles(2+11.5, 0+14))
        self.ghosts.pinky.setStartNode(self.nodes.getNodeFromTiles(2+11.5, 3+14))
        self.ghosts.inky.setStartNode(self.nodes.getNodeFromTiles(0+11.5, 3+14))
        self.ghosts.clyde.setStartNode(self.nodes.getNodeFromTiles(4+11.5, 3+14))
        self.ghosts.setSpawnNode(self.nodes.getNodeFromTiles(2+11.5, 3+14))

    def update(self):
        dt = self.clock.tick(30) / 1000.0
        self.pacman.update(dt)
        self.ghosts.update(dt)
        self.pellets.update(dt)
```

```
self.checkGhostEvents()
self.checkEvents()
self.checkPelletEvents()
self.render()
```

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

```
def checkGhostEvents(self):
    for ghost in self.ghosts:
        if self.pacman.collideGhost(ghost):
            if ghost.mode.current is FREIGHT:
                ghost.startSpawn()
```

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

```
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)
    self.ghosts.render(self.screen)
    pygame.display.update()
```

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

Pacman.py

```
import pygame
from pygame.locals import *
from vector import Vector2
from constants import *
from entity import Entity

class Pacman(Entity):
    def __init__(self, node):
        Entity.__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.direction = LEFT
self.node = node
self.setPosition()
self.target = node
self.collideRadius = 5

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:
        if self.collideCheck(pellet):
            return pellet
    return None

def collideGhost(self, ghost):
    return self.collideCheck(ghost)

def collideCheck(self, other):
    d = self.position - other.position
    dSquared = d.magnitudeSquared()
    rSquared = (self.collideRadius + other.collideRadius)**2
    if dSquared <= rSquared:
        return True
    return False

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

```

Entity.py

```

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

class Entity(object):
    def __init__(self, node):
        self.name = None
        self.directions = {UP:Vector2(0, -1),DOWN:Vector2(0, 1),
                           LEFT:Vector2(-1, 0), RIGHT:Vector2(1, 0), STOP:Vector2()}
        self.direction = STOP
        self.setSpeed(100)
        self.radius = 10
        self.collideRadius = 5
        self.color = WHITE
        self.node = node
        self.setPosition()
        self.target = node
        self.visible = True
        self.disablePortal = False
        self.goal = None
        self.directionMethod = self.randomDirection
        self.setStartNode(node)

    def setStartNode(self, node):
        self.node = node
        self.startNode = node
        self.target = node
        self.setPosition()

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

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

    def getNewTarget(self, direction):

```

```

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

def overshootTarget(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 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 setSpeed(self, speed):
    self.speed = speed * TILEWIDTH / 16

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

def update(self, dt):
    self.position += self.directions[self.direction]*self.speed*dt

    if self.overshootTarget():
        self.node = self.target
        directions = self.validDirections()
        direction = self.directionMethod(directions)
        if not self.disablePortal:
            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)

    self.setPosition()

def validDirections(self):

```

```

directions = []
for key in [UP, DOWN, LEFT, RIGHT]:
    if self.validDirection(key):
        if key != self.direction * -1:
            directions.append(key)
if len(directions) == 0:
    directions.append(self.direction * -1)
return directions

def randomDirection(self, directions):
    return directions[randint(0, len(directions)-1)]

def goalDirection(self, directions):
    distances = []
    for direction in directions:
        vec = self.node.position + self.directions[direction]*TILEWIDTH - self.goal
        distances.append(vec.magnitudeSquared())
    index = distances.index(min(distances))
    return directions[index]

```