

Up to Ghosts Setup

Nothing should be different about the game at this point. If it is, you did something wrong

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.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:
        d = self.position - pellet.position
        dSquared = d.magnitudeSquared()
        rSquared = (pellet.radius+self.collideRadius)**2
        if dSquared <= rSquared:
            return pellet
    return None

```

```

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

    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 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 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.overshotTarget():
        self.node = self.target
        directions = self.validDirections()
        direction = self.randomDirection(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)]

```