

Making the Chase

At this point the ghost should start moving in the top-left corner as before, then eventually start chasing Pacman. Still nothing happens when they collide

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 Ghost

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.")
        self.ghost = Ghost(self.nodes.getStartTempNode(), self.pacman)

    def update(self):
        dt = self.clock.tick(30) / 1000.0
        self.pacman.update(dt)
        self.ghost.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)
        self.ghost.render(self.screen)
        pygame.display.update()

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

```

Modes.py

```

from constants import *

class MainMode(object):
    def __init__(self):
        self.timer = 0
        self.scatter()

    def update(self, dt):
        self.timer += dt
        if self.timer >= self.time:
            if self.mode is SCATTER:
                self.chase()
            elif self.mode is CHASE:
                self.scatter()

    def scatter(self):
        self.mode = SCATTER
        self.time = 7
        self.timer = 0

    def chase(self):
        self.mode = CHASE
        self.time = 20
        self.timer = 0

class ModeController(object):
    def __init__(self, entity):
        self.timer = 0
        self.time = None

```

```

self.mainmode = MainMode()
self.current = self.mainmode.mode
self.entity = entity

def update(self, dt):
    self.mainmode.update(dt)
    self.current = self.mainmode.mode

```

Ghosts.py

```

import pygame
from pygame.locals import *
from vector import Vector2
from constants import *
from entity import Entity
from modes import ModeController

class Ghost(Entity):
    def __init__(self, node, pacman=None):
        Entity.__init__(self, node)
        self.name = GHOST
        self.points = 200
        self.goal = Vector2()
        self.directionMethod = self.goalDirection
        self.pacman = pacman
        self.mode = ModeController(self)

    def update(self, dt):
        self.mode.update(dt)
        if self.mode.current is SCATTER:
            self.scatter()
        elif self.mode.current is CHASE:
            self.chase()
        Entity.update(self, dt)

    def scatter(self):
        self.goal = Vector2()

    def chase(self):
        self.goal = self.pacman.position

```

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
GHOST = 3
```

```
WHITE = (255, 255, 255)
RED = (255, 0, 0)
```

```
PORTAL = 3
```

```
SCATTER = 0
CHASE = 1
FREIGHT = 2
SPAWN = 3
```

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

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]

```