

Scatter Ghost

At this point, when Pacman eats a ghost, the ghost will scatter to the home box and default to chase mode

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))
        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.getStartTempNode())
        self.pellets = PelletGroup("maze01.txt.")
        self.ghost = Ghost(self.nodes.getStartTempNode(), self.pacman)
        self.ghost.setSpawnNode(self.nodes.getNodeFromTiles(2+11.5, 3+14))

    def update(self):
        dt = self.clock.tick(30) / 1000.0
        self.pacman.update(dt)
        self.ghost.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):
    if self.pacman.collideGhost(self.ghost):
        if self.ghost.mode.current is FREIGHT:
            self.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.ghost.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.ghost.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.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
```

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

    def startFreight(self):
        self.mode.setFreightMode()
        if self.mode.current == FREIGHT:
            self.setSpeed(50)
            self.directionMethod = self.randomDirection

    def normalMode(self):
        self.setSpeed(100)
        self.directionMethod = self.goalDirection

    def spawn(self):
        self.goal = self.spawnNode.position

    def setSpawnNode(self, node):
        self.spawnNode = node
```

```

def startSpawn(self):
    self.mode.setSpawnMode()
    if self.mode.current == SPAWN:
        self.setSpeed(150)
        self.directionMethod = self.goalDirection
        self.spawn()

```

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)
        if self.current is FREIGHT:
            self.timer += dt
            if self.timer >= self.time:
                self.time = None
                self.entity.normalMode()
                self.current = self.mainmode.mode
            elif self.current in [SCATTER, CHASE]:
                self.current = self.mainmode.mode

```

```
if self.current is SPAWN:
    if self.entity.node == self.entity.spawnNode:
        self.entity.normalMode()
        self.current = self.mainmode.mode

def setSpawnMode(self):
    if self.current is FREIGHT:
        self.current = SPAWN

def setFreightMode(self):
    if self.current in [SCATTER, CHASE]:
        self.timer = 0
        self.time = 7
        self.current = FREIGHT
    elif self.current is FREIGHT:
        self.timer = 0
```