

Animate Ghosts

The ghosts' eyes now move in the direction they're going in.

Sprites.py

```
import pygame
from constants import *
import numpy as np
from animation import Animator

BASETILEWIDTH = 16
BASETILEHEIGHT = 16

class Spritesheet(object):
    def __init__(self):
        self.sheet = pygame.image.load("spritesheet.png").convert()
        transcolor = self.sheet.get_at((0,0))
        self.sheet.set_colorkey(transcolor)
        width = int(self.sheet.get_width() / BASETILEWIDTH * TILEWIDTH)
        height = int(self.sheet.get_height() / BASETILEHEIGHT * TILEHEIGHT)
        self.sheet = pygame.transform.scale(self.sheet, (width, height))

    def getImage(self, x, y, width, height):
        x *= TILEWIDTH
        y *= TILEHEIGHT
        self.sheet.set_clip(pygame.Rect(x, y, width, height))
        return self.sheet.subsurface(self.sheet.get_clip())

class PacmanSprites(Spritesheet):
    def __init__(self, entity):
        Spritesheet.__init__(self)
        self.entity = entity
        self.entity.image = self.getStartImage()
        self.animations = {}
        self.defineAnimations()
        self.stopimage = (8, 0)

    def defineAnimations(self):
        self.animations[LEFT] = Animator(((8, 0), (0, 0), (0, 2), (0, 0)))
        self.animations[RIGHT] = Animator(((10,0), (2, 0), (2, 2), (2, 0)))
        self.animations[UP] = Animator(((10,2), (6, 0), (6, 2), (6, 0)))
        self.animations[DOWN] = Animator(((8,2), (4, 0), (4, 2), (4, 0)))

    def update(self, dt):
        if self.entity.direction == LEFT:
            self.entity.image = self.getImage(*self.animations[LEFT].update(dt))
            self.stopimage = (8, 0)
        elif self.entity.direction == RIGHT:
```

```

        self.entity.image = self.getImage(*self.animations[RIGHT].update(dt))
        self.stopimage = (10, 0)
    elif self.entity.direction == DOWN:
        self.entity.image = self.getImage(*self.animations[DOWN].update(dt))
        self.stopimage = (8, 2)
    elif self.entity.direction == UP:
        self.entity.image = self.getImage(*self.animations[UP].update(dt))
        self.stopimage = (10, 2)
    elif self.entity.direction == STOP:
        self.entity.image = self.getImage(*self.stopimage)

def reset(self):
    for key in list(self.animations.keys()):
        self.animations[key].reset()

def getStartImage(self):
    return self.getImage(8, 0)

def getImage(self, x, y):
    return Spritesheet.getImage(self, x, y, 2*TILEWIDTH, 2*TILEHEIGHT)

class GhostSprites(Spritesheet):
    def __init__(self, entity):
        Spritesheet.__init__(self)
        self.x = {BLINKY:0, PINKY:2, INKY:4, CLYDE:6}
        self.entity = entity
        self.entity.image = self.getStartImage()

    def update(self, dt):
        x = self.x[self.entity.name]
        if self.entity.mode.current in [SCATTER, CHASE]:
            if self.entity.direction == LEFT:
                self.entity.image = self.getImage(x, 8)
            elif self.entity.direction == RIGHT:
                self.entity.image = self.getImage(x, 10)
            elif self.entity.direction == DOWN:
                self.entity.image = self.getImage(x, 6)
            elif self.entity.direction == UP:
                self.entity.image = self.getImage(x, 4)
        elif self.entity.mode.current == FREIGHT:
            self.entity.image = self.getImage(10, 4)
        elif self.entity.mode.current == SPAWN:
            if self.entity.direction == LEFT:
                self.entity.image = self.getImage(8, 8)
            elif self.entity.direction == RIGHT:
                self.entity.image = self.getImage(8, 10)
            elif self.entity.direction == DOWN:
                self.entity.image = self.getImage(8, 6)
            elif self.entity.direction == UP:
                self.entity.image = self.getImage(8, 4)

    def getStartImage(self):
        return self.getImage(self.x[self.entity.name], 4)

```

```

def getImage(self, x, y):
    return Spritesheet.getImage(self, x, y, 2*TILEWIDTH, 2*TILEHEIGHT)

class FruitSprites(Spritesheet):
    def __init__(self, entity):
        Spritesheet.__init__(self)
        self.entity = entity
        self.entity.image = self.getStartImage()

    def getStartImage(self):
        return self.getImage(16, 8)

    def getImage(self, x, y):
        return Spritesheet.getImage(self, x, y, 2*TILEWIDTH, 2*TILEHEIGHT)

class LifeSprites(Spritesheet):
    def __init__(self, numlives):
        Spritesheet.__init__(self)
        self.resetLives(numlives)

    def removeImage(self):
        if len(self.images) > 0:
            self.images.pop(0)

    def resetLives(self, numlives):
        self.images = []
        for i in range(numlives):
            self.images.append(self.getImage(0,0))

    def getImage(self, x, y):
        return Spritesheet.getImage(self, x, y, 2*TILEWIDTH, 2*TILEHEIGHT)

class MazeSprites(Spritesheet):
    def __init__(self, mazefile, rotfile):
        Spritesheet.__init__(self)
        self.data = self.readMazeFile(mazefile)
        self.rotdata = self.readMazeFile(rotfile)

    def getImage(self, x, y):
        return Spritesheet.getImage(self, x, y, TILEWIDTH, TILEHEIGHT)

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

    def constructBackground(self, background, y):
        for row in list(range(self.data.shape[0])):
            for col in list(range(self.data.shape[1])):
                if self.data[row][col].isdigit():
                    x = int(self.data[row][col]) + 12
                    sprite = self.getImage(x, y)
                    rotval = int(self.rotdata[row][col])
                    sprite = self.rotate(sprite, rotval)

```

```

        background.blit(sprite, (col*TILEWIDTH, row*TILEHEIGHT))
    elif self.data[row][col] == '=':
        sprite = self.getImage(10, 8)
        background.blit(sprite, (col*TILEWIDTH, row*TILEHEIGHT))

    return background

def rotate(self, sprite, value):
    return pygame.transform.rotate(sprite, value*90)

```

Ghosts.py

```

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

class Ghost(Entity):
    def __init__(self, node, pacman=None, blinky=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)
        self.blinky = blinky
        self.homeNode = node

    def update(self, dt):
        self.sprites.update(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()

class Blinky(Ghost):
    def __init__(self, node, pacman=None, blinky=None):
        Ghost.__init__(self, node, pacman, blinky)
        self.name = BLINKY
        self.color = RED
        self.sprites = GhostSprites(self)

class Pinky(Ghost):
    def __init__(self, node, pacman=None, blinky=None):
        Ghost.__init__(self, node, pacman, blinky)
        self.name = PINKY
        self.color = PINK
        self.sprites = GhostSprites(self)

def scatter(self):
    self.goal = Vector2(TILEWIDTH*NCOLS, 0)

def chase(self):
    self.goal = self.pacman.position + self.pacman.directions[self.pacman.direction] * TILEWIDTH * 4

class Inky(Ghost):
    def __init__(self, node, pacman=None, blinky=None):
        Ghost.__init__(self, node, pacman, blinky)
        self.name = INKY
        self.color = TEAL
        self.sprites = GhostSprites(self)

def scatter(self):
    self.goal = Vector2(TILEWIDTH*NCOLS, TILEHEIGHT*NROWS)

def chase(self):
    vec1 = self.pacman.position + self.pacman.directions[self.pacman.direction] * TILEWIDTH * 2
    vec2 = (vec1 - self.blinky.position) * 2
    self.goal = self.blinky.position + vec2

```

```

class Clyde(Ghost):
    def __init__(self, node, pacman=None, blinky=None):
        Ghost.__init__(self, node, pacman, blinky)
        self.name = CLYDE
        self.color = ORANGE
        self.sprites = GhostSprites(self)

    def scatter(self):
        self.goal = Vector2(0, TILEHEIGHT*NROWS)

    def chase(self):
        d = self.pacman.position - self.position
        ds = d.magnitudeSquared()
        if ds <= (TILEWIDTH * 8)**2:
            self.scatter()
        else:
            self.goal = self.pacman.position + self.pacman.directions[self.pacman.direction] * TILEWIDTH * 4

class GhostGroup(object):
    def __init__(self, node, pacman):
        self.blinky = Blinky(node, pacman)
        self.pinky = Pinky(node, pacman)
        self.inky = Inky(node, pacman, self.blinky)
        self.clyde = Clyde(node, pacman)
        self.ghosts = [self.blinky, self.pinky, self.inky, self.clyde]

    def __iter__(self):
        return iter(self.ghosts)

    def update(self, dt):
        for ghost in self:
            ghost.update(dt)

    def startFreight(self):
        for ghost in self:
            ghost.startFreight()
        self.resetPoints()

    def setSpawnNode(self, node):
        for ghost in self:
            ghost.setSpawnNode(node)

    def updatePoints(self):
        for ghost in self:
            ghost.points *= 2

    def resetPoints(self):
        for ghost in self:
            ghost.points = 200

    def reset(self):
        for ghost in self:
            ghost.reset()

```

```
def hide(self):
    for ghost in self:
        ghost.visible = False

def show(self):
    for ghost in self:
        ghost.visible = True

def render(self, screen):
    for ghost in self:
        ghost.render(screen)

def reset(self):
    Entity.reset(self)
    self.points = 200
    self.directionMethod = self.goalDirection
```