

# Eating Ghosts (Error)

***An error returns when Pacman collides with a ghost that ISN'T in freight mode. I don't feel like dealing with it, so this is the main game-breaking bug I'm gonna leave in.***

## 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
from fruit import Fruit
from pauser import Pause

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()
        self.fruit = None
        self.pause = Pause(True)
        self.level = 0
        self.lives = 5

    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.pellets.update(dt)
    if not self.pause.paused:
        self.pacman.update(dt)
        self.ghosts.update(dt)
        if self.fruit is not None:
            self.fruit.update(dt)
        self.checkGhostEvents()
        self.checkPelletEvents()
        self.checkFruitEvents
    afterPauseMethod = self.pause.update(dt)
    if afterPauseMethod is not None:
        afterPauseMethod()
    self.checkEvents()
    self.render()

def checkEvents(self):
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()
        elif event.type == KEYDOWN:
            if event.key == K_SPACE:
                if self.pacman.alive:
                    self.pause.setPause(playerPaused=True)
                    if not self.pause.paused:
                        self.showEntities()
                else:
                    self.hideEntities()

def checkGhostEvents(self):
    for ghost in self.ghosts:
        if self.pacman.collideGhost(ghost):
            if ghost.mode.current is FREIGHT:
                self.pacman.visible = False
                ghost.visible = False
                self.pause.setPause(pauseTime=1, func=self.showEntities)
                ghost.startSpawn()
            elif ghost.mode.current is not SPAWN:
                if self.pacman.alive:
                    self.lives -= 1
                    self.pacman.die()
                    self.ghosts.hide()
                if self.lives <= 0:
                    self.pause.setPause(pauseTime=3, func=self.restartGame)
                else:
                    self.pause.setPause(pauseTime=3, func=self.resetLevel)

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()
        if self.pellets.isEmpty():
            self.hideEntities()
            self.pause.setPause(pauseTime=3, func=self.nextLevel)

def checkFruitEvents(self):
    if self.pellets.numEaten == 50 or self.pellets.numEaten == 140:
        if self.fruit is None:
            self.fruit = Fruit(self.nodes.getNodeFromTiles(9, 20))
        if self.fruit is not None:
            if self.pacman.collideCheck(self.fruit):
                self.fruit = None
            elif self.fruit.destroy:
                self.fruit = None

def showEntities(self):
    self.pacman.visible = True
    self.ghosts.show()

def hideEntities(self):
    self.pacman.visible = False
    self.ghosts.hide()

def nextLevel(self):
    self.showEntities()
    self.level += 1
    self.pause.paused = True
    self.startGame()

def restartGame(self):
    self.lives = 5
    self.level = 0
    self.pause.paused = True
    self.fruit = None
    self.startGame()

def resetLevel(self):
    self.pause.paused = True
    self.pacman.reset()
    self.ghosts.reset()
    self.fruit = None

def render(self):
    self.screen.blit(self.background, (0,0))
    self.nodes.render(self.screen)
    self.pellets.render(self.screen)
    if self.fruit is not None:
        self.fruit.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.setBetweenNodes(LEFT)
        self.node = node
        self.setPosition()
        self.target = node
        self.collideRadius = 5
        self.alive = True

    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

def reset(self):
    Entity.reset(self)
    self.direction = LEFT
    self.setBetweenNodes(LEFT)
    self.alive = True

def die(self):
    self.alive = False
    self.direction = 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, 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.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

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

    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

    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

    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

```

## 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.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 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.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]

```

```
def setBetweenNodes(self, direction):
    if self.node.neighbors[direction] is not None:
        self.target = self.node.neighbors[direction]
        self.position = (self.node.position + self.target.position) / 2.0

def reset(self):
    self.setStartNode(self.startNode)
    self.direction = STOP
    self.speed = 100
    self.visible = True
```