

Smoothing Pacman's Movement

At this point, Pacman now moves smoothly along the path between nodes. No more 'jumping'

Run.py

```
import pygame
from pygame.locals import *
from constants import *
from pacman import Pacman
from nodes import NodeGroup

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()
        self.nodes.setupTestNodes()
        self.pacman = Pacman(self.nodes.nodeList[0])

    def update(self):
        dt = self.clock.tick(30) / 1000.0
        self.pacman.update(dt)
        self.checkEvents()
        self.render()

    def checkEvents(self):
        for event in pygame.event.get():
            if event.type == QUIT:
                exit()

    def render(self):
        self.screen.blit(self.background, (0,0))
        self.nodes.render(self.screen)
        self.pacman.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 *

class Pacman(object):
    def __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

    def setPosition(self):
        self.position = self.node.position.copy()

    def update(self, dt):
        self.position += self.directions[self.direction]*self.speed*dt
        direction = self.getValidKey()
        if self.overshotTarget():
            self.node = self.target
            self.target = self.getNewTarget(direction)
            if self.target is not self.node:
                self.direction = direction
            else:
                self.direction = STOP
            self.setPosition()

    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 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 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 render(self, screen):
    p = self.position.asInt()
    pygame.draw.circle(screen, self.color, p, self.radius)

```

Nodes.py

```

import pygame
from vector import Vector2
from constants import *

class Node(object):
    def __init__(self, x, y):
        self.position = Vector2(x, y)
        self.neighbors = {UP:None, DOWN:None, LEFT:None, RIGHT:None}

    def render(self, screen):
        for n in self.neighbors.keys():
            if self.neighbors[n] is not None:
                line_start = self.position.asTuple()
                line_end = self.neighbors[n].position.asTuple()
                pygame.draw.line(screen, WHITE, line_start, line_end, 4)
                pygame.draw.circle(screen, RED, self.position.asInt(), 12)

class NodeGroup(object):
    def __init__(self,):
        self.nodeList = []

    def setupTestNodes(self):
        nodeA = Node(80,80)

```

```

nodeB = Node(160, 80)
nodeC = Node(80, 160)
nodeD = Node(160, 160)
nodeE = Node(208, 160)
nodeF = Node(80, 320)
nodeG = Node(208, 320)
nodeA.neighbors[RIGHT] = nodeB
nodeA.neighbors[DOWN] = nodeC
nodeB.neighbors[LEFT] = nodeA
nodeB.neighbors[DOWN] = nodeD
nodeC.neighbors[UP] = nodeA
nodeC.neighbors[RIGHT] = nodeD
nodeC.neighbors[DOWN] = nodeF
nodeD.neighbors[UP] = nodeB
nodeD.neighbors[LEFT] = nodeC
nodeD.neighbors[RIGHT] = nodeE
nodeE.neighbors[LEFT] = nodeD
nodeE.neighbors[DOWN] = nodeG
nodeF.neighbors[UP] = nodeC
nodeF.neighbors[RIGHT] = nodeG
nodeG.neighbors[UP] = nodeE
nodeG.neighbors[LEFT] = nodeF
self.nodeList = [nodeA, nodeB, nodeC, nodeD, nodeE, nodeF, nodeG]

def render(self, screen):
    for node in self.nodeList:
        node.render(screen)

```