

# Creating Home Box

***At this point, the home box is now an open space that Pacman and the ghost are able to come and go into. We'll add the restrictions later***

## 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)

    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()

```

## Nodes.py

```

import pygame
from vector import Vector2
from constants import *
import numpy as np

class Node(object):
    def __init__(self, x, y):
        self.position = Vector2(x, y)
        self.neighbors = {UP:None, DOWN:None, LEFT:None, RIGHT:None, PORTAL: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, level):
        self.level = level
        self.nodesLUT = {}
        self.nodeSymbols = ['+', 'P', 'n']
        self.pathSymbols = ['.', '-', '|', 'p']
        data = self.readMazeFile(level)

```

```

self.createNodeTable(data)
self.connectHorizontally(data)
self.connectVertically(data)
self.homekey = None

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

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

def createNodeTable(self, data, xoffset=0, yoffset=0):
    for row in list(range(data.shape[0])):
        for col in list(range(data.shape[1])):
            if data[row][col] in self.nodeSymbols:
                x, y = self.constructKey(col+xoffset, row+yoffset)
                self.nodesLUT[(x, y)] = Node(x, y)

def constructKey(self, x, y):
    return x * TILEWIDTH, y * TILEHEIGHT

def connectHorizontally(self, data, xoffset=0, yoffset=0):
    for row in list(range(data.shape[0])):
        key = None
        for col in list(range(data.shape[1])):
            if data[row][col] in self.nodeSymbols:
                if key is None:
                    key = self.constructKey(col+xoffset, row+yoffset)
                else:
                    otherkey = self.constructKey(col+xoffset, row+yoffset)
                    self.nodesLUT[key].neighbors[RIGHT] = self.nodesLUT[otherkey]
                    self.nodesLUT[otherkey].neighbors[LEFT] = self.nodesLUT[key]
                    key = otherkey
            elif data[row][col] not in self.pathSymbols:
                key = None

def connectVertically(self, data, xoffset=0, yoffset=0):
    dataT = data.transpose()
    for col in list(range(dataT.shape[0])):
        key = None
        for row in list(range(dataT.shape[1])):
            if dataT[col][row] in self.nodeSymbols:
                if key is None:
                    key = self.constructKey(col+xoffset, row+yoffset)
                else:
                    otherkey = self.constructKey(col+xoffset, row+yoffset)
                    self.nodesLUT[key].neighbors[DOWN] = self.nodesLUT[otherkey]
                    self.nodesLUT[otherkey].neighbors[UP] = self.nodesLUT[key]
                    key = otherkey
            elif dataT[col][row] not in self.pathSymbols:
                key = None

```

```

def getNodeFromPixels(self, xpixel, ypixel):
    if (xpixel, ypixel) in self.nodesLUT.keys():
        return self.nodesLUT[(xpixel, ypixel)]
    return None

def getNodeFromTiles(self, col, row):
    x, y = self.constructKey(col, row)
    if (x, y) in self.nodesLUT.keys():
        return self.nodesLUT[(x, y)]
    return None

def getStartTempNode(self):
    nodes = list(self.nodesLUT.values())
    return nodes[0]

def setPortalPair(self, pair1, pair2):
    key1 = self.constructKey(*pair1)
    key2 = self.constructKey(*pair2)
    if key1 in self.nodesLUT.keys() and key2 in self.nodesLUT.keys():
        self.nodesLUT[key1].neighbors[PORTAL] = self.nodesLUT[key2]
        self.nodesLUT[key2].neighbors[PORTAL] = self.nodesLUT[key1]

def createHomeNodes(self, xoffset, yoffset):
    homedata = np.array([['X','X','+','X','X'],
                        ['X','X','.', 'X','X'],
                        ['+','X','.', 'X','+'],
                        ['+','.', '+','.', '+'],
                        ['+','X','X','X','+']])

    self.createNodeTable(homedata, xoffset, yoffset)
    self.connectHorizontally(homedata, xoffset, yoffset)
    self.connectVertically(homedata, xoffset, yoffset)
    self.homekey = self.constructKey(xoffset+2, yoffset)
    return self.homekey

def connectHomeNodes(self, homekey, otherkey, direction):
    key = self.constructKey(*otherkey)
    self.nodesLUT[homekey].neighbors[direction] = self.nodesLUT[key]
    self.nodesLUT[key].neighbors[direction*-1] = self.nodesLUT[homekey]

```