Boston University

**PROJECT REPORT**

**FOR**

**ENG EK 128**

Professor Roscoe C Giles

27 April 2016

Presented by:

Ian C. Ballou

Lina Lin Wei

Initial Goal

Our initial goal was to make a few levels of Pac-Man by using Python. We also thought of implementing a custom motion-based wireless controller if we did manage to complete the code earlier. The Controller will consist of an Arduino equipped with a 9-degree-of-freedom (9-Dof) device and an XBee radio which will transmit the data. (9-Dof: <https://www.adafruit.com/products/1714)>.

Work Accomplished

Three weeks proved to be too short for us to include the controller to the Pac-Man game. We also came to realized that the Pac-Man game would require more time because of the ghosts. They would demand a lot of coding because each of them would follow a different path and sometimes try to pursue Pac-Man. Moreover, they evolve into different forms with different properties when Pac-Man eats the magic pellet.

We have, therefore, scaled down our Pac-Man game. We removed the ghosts and pellets. We kept the fruits that would randomly pop onto the screen and custom- made the maze in which Pac-Man would be moving. The aim of the player for our final Pac-Man game is to move Pac man through a maze and collect a maximum amount of fruits before the energy runs out. The fruits would randomly pop up inside the maze, and when Pac man ate one fruit, its energy will go up a little bit.

We also managed to import the Pygame library into Python3. On Linux it is possible to grab the Pygame source from Bitbucket.org and then make and install the Pygame library. [This tutorial](http://askubuntu.com/questions/401342/how-to-download-pygame-in-python3-3) is helpful. For Mac, Pygame can be installed using the package manager Homebrew. Here is [the tutorial](http://florian-berger.de/en/articles/installing-pygame-for-python-3-on-os-x/) we followed. Note the Ruby command is old, it is better to use:

/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"

Lesson Learned

Graphics and classes were the two most important topics in our project. One challenge was figuring out how to actually display the figures and the Pac-Man animation. After researching online, we learned the most appropriate functions of the Pygame library to apply in order to meet our graphical expectations. One specific new skill was how to use a sprite sheet. A sprite sheet is one image with many different images, or sprites, spaced apart. In order to access the sprites, the pixel locations of these sprites are used to “clip” them out individually. To do an animation, for example, a list of 3 sprite clippings can be iterated through and drawn on the screen.

Another challenge was with collision detection. Our Pac-Man stops when he hits a wall, which was no easy task. Each individual Wall object has certain properties from the base Rect() class which allows for detection of overlapping rectangles. Every object (Pac-Man, Fruit, Wall) all have rectangular properties for collision detection.

Program Code with comments

* *pacman.py*

from SpriteSheet import SpriteSheet

import sys

import os

import pygame

import random

from level import level

BACKGROUND = pygame.image.load('blackbox.jpg')

BACKGROUND = pygame.transform.scale(BACKGROUND, (800, 600))

BACKGROUND\_RECT = BACKGROUND.get\_rect()

# Class for edible fruit

# Sprite has useful group and collision qualities

class Fruit(pygame.sprite.Sprite):

def \_\_init\_\_(self, x0, y0):

pygame.sprite.Sprite.\_\_init\_\_(self)

# Sheet with all character sprites

self.ss = SpriteSheet('pacmansprites2.png')

self.image = self.ss.get\_image(175, 165, 15, 15)

self.rect = self.image.get\_rect(x=x0, y=y0)

self.eaten = False

def checkCollision(self, sprite):

if self.rect.colliderect(sprite.rect):

self.eaten = True

print("Collision")

def checkWallCollision(self, wallRects):

for wall in wallRects:

while self.rect.colliderect(wall.rect):

self.rect.x += 15

if self.rect.x > 799:

self.rect.x = 80

for wall in wallRects:

if self.rect.colliderect(wall.rect):

return False

else:

return True

def isEaten(self):

return self.eaten

# Remember: image is part of sprite, used for drawing

# Class for Pacman

class Pacman(pygame.sprite.Sprite):

def \_\_init\_\_(self, x0, y0):

pygame.sprite.Sprite.\_\_init\_\_(self)

self.ss = SpriteSheet('pacmansprites2.png')

self.leftAnim = [self.ss.get\_image(25, 5, 15, 15),\

self.ss.get\_image(5, 5, 15, 15),\

self.ss.get\_image(45, 5, 15, 15)]

self.rightAnim = [self.ss.get\_image(25, 25, 15, 15),\

self.ss.get\_image(5, 25, 15, 15),\

self.ss.get\_image(45, 5, 15, 15)]

self.upAnim = [self.ss.get\_image(25, 45, 15, 15),\

self.ss.get\_image(5, 45, 15, 15),\

self.ss.get\_image(45, 5, 15, 15)]

self.downAnim = [self.ss.get\_image(25, 65, 15, 15),\

self.ss.get\_image(5, 65, 15, 15),\

self.ss.get\_image(45, 5, 15, 15)]

self.leftV = -3

self.rightV = 3

self.upV = -3

self.downV = 3

self.image = self.leftAnim[1]

self.imageIndex = 1

# self.rect = self.image.get\_rect(x=x0, y=y0)

self.rect = pygame.Rect(x0, y0, 30, 30)

# self.radius = 5

self.vX = self.rightV

self.vY = 0

self.timer = 0.0

# Moves Pacman rect object

def move(self):

self.image = pygame.transform.scale(self.animate(), (30, 30))

self.rect.x += self.vX

if self.rect.x > 800:

self.rect.x = 1

if self.rect.x < 0:

self.rect.x = 799

self.rect.y += self.vY

if self.rect.y > 600:

self.rect.y = 1

if self.rect.y < 0:

self.rect.y = 599

# Checks every wall if Pacman hit one of them

def checkWallCollision(self, wallRects):

for wall in wallRects:

if self.rect.colliderect(wall.rect):

if self.vX == self.rightV:

while self.rect.colliderect(wall.rect):

self.rect.x -= 1

self.rect.x -= 1

# self.vX = self.leftV

self.vY = 0

if self.vX == self.leftV:

while self.rect.colliderect(wall.rect):

self.rect.x += 1

self.rect.x += 1

# self.vX = self.rightV

self.vY = 0

if self.vY == self.downV:

while self.rect.colliderect(wall.rect):

self.rect.y -= 1

self.rect.y -= 1

# self.vY = self.upV

self.vX = 0

if self.vY == self.upV:

while self.rect.colliderect(wall.rect):

self.rect.y += 1

self.rect.y += 1

# self.vY = self.downV

self.vX = 0

print("Wall Collision")

# Determine which image Pacman is displayed as

# Loops through each image depending on last displayed

# Returns image to be shown

def animate(self):

if self.vX == self.rightV:

if self.currentTime - self.timer > 100:

if self.imageIndex < len(self.rightAnim) - 1:

self.imageIndex += 1

else:

self.imageIndex = 0

self.timer = self.currentTime

return self.rightAnim[self.imageIndex]

elif self.vX == self.leftV:

if self.currentTime - self.timer > 100:

if self.imageIndex < len(self.leftAnim) - 1:

self.imageIndex += 1

else:

self.imageIndex = 0

self.timer = self.currentTime

return self.leftAnim[self.imageIndex]

elif self.vY == self.upV:

if self.currentTime - self.timer > 100:

if self.imageIndex < len(self.upAnim) - 1:

self.imageIndex += 1

else:

self.imageIndex = 0

self.timer = self.currentTime

return self.upAnim[self.imageIndex]

elif self.vY == self.downV:

if self.currentTime - self.timer > 100:

if self.imageIndex < len(self.downAnim) - 1:

self.imageIndex += 1

else:

self.imageIndex = 0

self.timer = self.currentTime

return self.downAnim[self.imageIndex]

# Updates Pacman

def update(self, currentTime, keys, wallRects):

self.currentTime = currentTime

self.handleInput(keys)

self.move()

self.checkWallCollision(wallRects)

def handleInput(self, keys):

if keys[pygame.K\_UP]:

self.vY = self.upV

self.vX = 0

if keys[pygame.K\_RIGHT]:

self.vY = 0

self.vX = self.rightV

if keys[pygame.K\_LEFT]:

self.vY = 0

self.vX = self.leftV

if keys[pygame.K\_DOWN]:

self.vY = self.downV

self.vX = 0

# Wall object

# Defines rectangle space for wall to be displayed

class Wall(object):

def \_\_init\_\_(self, pos):

self.rect = pygame.Rect(pos[0], pos[1], 20, 20)

# Main Game class

# Controls game state, input and display

class Game(object):

def \_\_init\_\_(self):

self.screen = self.initPygame()

self.screenRect = self.screen.get\_rect()

self.pacmanGroup = self.initPacman()

self.done = False

self.clock = pygame.time.Clock()

self.fps = 60

self.currentTime = 0.0

self.score = 0

self.font = pygame.font.Font(None, 36)

self.maxTime = 20000 # In ms

self.timeLeft = self.maxTime - self.currentTime

self.level = level

self.walls = []

self.initLevel()

self.fruitGroup = self.initFruit()

# self.pacman = Pacman(100, 500)

# Parse level file into actual level with Walls

def initLevel(self):

x = y = 0

for row in self.level:

for col in row:

if col == "W":

self.walls.append(Wall((x, y)))

x += 20

y += 20

x = 0

def initPacman(self):

spriteGroup = pygame.sprite.Group()

pacman = Pacman(50, 80)

spriteGroup.add(pacman)

# pacman2 = Pacman(40, 40)

# spriteGroup.add(pacman2)

return spriteGroup

# Pygame setup

def initPygame(self):

os.environ['SDL\_VIDEO\_CENTERED'] = '1'

pygame.init()

pygame.display.set\_caption('Eat-Man Ultra')

screen = pygame.display.set\_mode((800, 600))

return screen

def initFruit(self):

spriteGroup = pygame.sprite.Group()

fruit = Fruit(100, 150)

spriteGroup.add(fruit)

cont = False

while not cont:

cont = fruit.checkWallCollision(self.walls)

# fruit.checkWallCollision(self.walls)

return spriteGroup

# Checks if fruit was eaten

# Handles respawn of fruit

def updateFruit(self):

for fruit in self.fruitGroup:

fruit.checkCollision(self.pacmanGroup.sprites()[0])

if fruit.isEaten():

self.fruitGroup.remove(fruit)

self.score += 1

self.maxTime += 2000

if len(self.fruitGroup.sprites()) < 1:

random.seed(self.currentTime)

x = random.randint(40, 760)

y = random.randint(100, 500)

fruit = Fruit(x, y)

self.fruitGroup.add(fruit)

cont = False

while not cont:

cont = fruit.checkWallCollision(self.walls)

def updateText(self):

scoreText = self.font.render(str(self.score), 1, (255, 255, 255))

scoreRect = scoreText.get\_rect()

scoreRect.centerx = 400

scoreRect.centery = 580

self.screen.blit(scoreText, scoreRect)

enText = self.font.render("Energy", 1, (255, 255, 255))

enRect = enText.get\_rect()

enRect.centerx = 600

enRect.centery = 580

self.screen.blit(enText, enRect)

timeText = self.font.render(str(self.timeLeft), 1, (255, 255, 255))

timeRect = timeText.get\_rect()

timeRect.centerx = 700

timeRect.centery = 580

self.screen.blit(timeText, timeRect)

# Root of game logic

def update(self):

while not self.done and self.timeLeft > 0:

self.currentTime = pygame.time.get\_ticks()

self.timeLeft = self.maxTime - self.currentTime

self.keys = self.getUserInput()

self.pacmanGroup.update(self.currentTime, self.keys, self.walls)

# pygame.draw.rect(self.screen, (0, 0, 255), self.pacmanGroup.sprites()[0].rect)

# pygame.display.flip()

self.updateFruit()

self.screen.blit(BACKGROUND, BACKGROUND\_RECT)

for wall in self.walls:

pygame.draw.rect(self.screen, (136, 0, 137), wall.rect)

self.updateText()

self.pacmanGroup.draw(self.screen)

self.fruitGroup.draw(self.screen)

pygame.display.update()

self.clock.tick(self.fps)

self.endScreen(self.keys)

def getUserInput(self):

for event in pygame.event.get():

if event.type == pygame.QUIT:

self.done = True

keys = pygame.key.get\_pressed()

return keys

def endScreen(self, keys):

# Handles entire end screen

# Really should be split into classes

self.screen.blit(BACKGROUND, BACKGROUND\_RECT)

finishInfoText = self.font.render("Final Score:", 1, (255, 255, 255))

finishInfoRect = finishInfoText.get\_rect()

finishInfoRect.centerx = 400

finishInfoRect.centery = 260

self.screen.blit(finishInfoText, finishInfoRect)

finishText = self.font.render(str(self.score), 1, (255, 255, 255))

finishRect = finishText.get\_rect()

finishRect.centerx = 400

finishRect.centery = 300

self.screen.blit(finishText, finishRect)

enterText = self.font.render("Press enter to quit", 1, (255, 255, 255))

enterRect = enterText.get\_rect()

enterRect.centerx = 400

enterRect.centery = 360

self.screen.blit(enterText, enterRect)

pygame.display.update()

disp = False

while not keys[pygame.K\_RETURN]:

oldTime = pygame.time.get\_ticks()

timer = pygame.time.get\_ticks() - oldTime

keys = self.getUserInput()

while timer < 500 and not keys[pygame.K\_RETURN]:

timer = pygame.time.get\_ticks() - oldTime

if disp:

self.screen.blit(BACKGROUND, BACKGROUND\_RECT)

self.screen.blit(finishInfoText, finishInfoRect)

self.screen.blit(finishText, finishRect)

self.screen.blit(enterText, enterRect)

else: s

self.screen.blit(BACKGROUND, BACKGROUND\_RECT)

self.screen.blit(finishInfoText, finishInfoRect)

self.screen.blit(finishText, finishRect)

disp = not disp

pygame.display.update()

game = Game()

game.update()

*level.py*

level = [

"WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW",

" W W WW ",

" W W W WWWWW ",

" WWWW WWWW WWW W WWWW",

" W W W ",

" W W WWWWWW WWW ",

" WWWW W W W WWW",

" W W W W W ",

" WW WWW WWWWWW W W W ",

" W WWW W ",

"WWW W WWW W ",

" WWWWWWW WWW W W ",

" W W W W ",

" W W wwwwWWW WWWWW ",

" WWW WWW W ",

" WWW W ",

"WWW W WWWWW ",

" WWWWWWW WWW WWW WWWW ",

" W WWW W ",

" W W W ",

" W WWW W WWWW W ",

" W WWWWWW W W ",

" W W WWWWWW WW ",

" WWWWWWW WWWWW W W ",

" W W ",

" W ",

"WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW",

]

*constants.py*

"""

Global constants for SpriteSheet.py

"""

# Colors

BLACK = ( 0, 0, 0)

WHITE = ( 255, 255, 255)

BLUE = ( 0, 0, 255)

# Screen dimensions

SCREEN\_WIDTH = 800

SCREEN\_HEIGHT = 600

*SpriteSheet.py*

"""

This module is used to pull individual sprites from sprite sheets.

"""

import pygame

import constants

class SpriteSheet(object):

""" Class used to grab images out of a sprite sheet. """

def \_\_init\_\_(self, file\_name):

""" Constructor. Pass in the file name of the sprite sheet. """

# Load the sprite sheet.

self.sprite\_sheet = pygame.image.load(file\_name).convert()

def get\_image(self, x, y, width, height):

""" Grab a single image out of a larger spritesheet

Pass in the x, y location of the sprite

and the width and height of the sprite. """

# Create a new blank image

image = pygame.Surface([width, height]).convert()

# Copy the sprite from the large sheet onto the smaller image

image.blit(self.sprite\_sheet, (0, 0), (x, y, width, height))

# Assuming black works as the transparent color

image.set\_colorkey(constants.BLACK)

# Return the image

return image

*blackbox.jpg*

**

*pacmansprites2.png*

