Final Project

Design Document

By Mark Delp for CSC221 Final Project

[Introduction 2](#_Toc192066654)

[Project Functionality 2](#_Toc192066655)

[Design Process 2](#_Toc192066656)

[Project Development 5](#_Toc192066657)

[Pseudocode 5](#_Toc192066658)

[Flowchart 13](#_Toc192066659)

[UML Diagram 15](#_Toc192066660)

[Requirements 16](#_Toc192066661)

## Introduction

### Project Functionality

I want to create a simple game of snake that captures the core functions and essence of the original game. Some key features that I want to include are the simple movement (left, right, up, down), collecting snake food, randomly generate food coordinates, having the snake grow as it eats, and setting up collision boundaries for the size of the window, the head and body of the snake, head of the snake with the food, and the body with the food. I’ve settled on creating a score function that populates on the game over screen instead of on the play area screen. Furthermore, I’m including a start menu screen with a play button and a game over screen with a restart button. I am forgoing any graphics/images/sounds to be implemented into the game as I believe the game should be kept as simple as possible as the premise of the game is also very simple.

### Design Process

I initially started by researching the pygame module documentation to get me started as I have never used pygame and am unfamiliar with its syntax and functionality. Luckily, there was an example block of code that included the basic setup for a window, a while loop that keeps the game window running, and a quit function to kill the program. From there, there was example code that included movement functions within the while loop to control a circle around a screen. I found this helpful for the very beginning of this project but soon discovered some issues with it that would not suit the application of this project. For one, the shape but most importantly, the shape could move diagonally. This wouldn’t work as I wanted the snake to only move in a very rigid manner (left, right, up, and down only).

From there I experimented with the pygame.draw feature to create a square instead but had some trouble getting square to move in the manner that I wanted. So, I consulted geeksforgeeks.com to help me out. Geeks for Geeks showed me how to assign vector-like components to the shape in order to move it in only the x and y direction but I still needed to get the shape to continuously move on its own as opposed to having to keep the keys held down to move the shape. From there, I found documentation on reddit that explained to me the idea of taking those vector-like components and adding a “velocity” component to it within the while loop to keep the shape continuously moving. This idea proved itself useful but would be removed and reimplemented in an entirely different way later. Furthermore, it familiarized me with the .KEYDOWN functionality as opposed to the .get\_pushed which helped simplify the movement code.

At this point in development I had a square that continuously moved on its own and moved in the x,y directions. From there, I thought that this would be a good point to start implementing collisions. I started out by first creating another square on the screen and creating code saying that if the moving square touched the positions of the new square then the game would end. I had immense difficulty trying to implement this as I was trying to put all this code into the while loop. This was my first big mistake that led me to no resolution. I decided to do some more research on Object-Oriented Programming (OOP) where I came across an online article that familiarized me with creating classes and instances (as this was what I was not doing before) that could later be utilized when called.

As helpful as this was, it felt way beyond my understanding in how to implement these fundamentals to produce code that semantically fit my application. From there, I went to Youtube and watched a multipart video series on Classes and Objects with Python by CS Dojo. This further helped my understanding on what exactly a Class was and the functions created within them (aka Methods). Plus, I learned the act of calling a class and it’s methods is known as creating an instance. Despite this better understanding, I wanted to see code in action with pygame, so, I found another Youtube video by Clear Code titled “the ultimate introduction to Pygame”. This was an almost 4 hour long video where the creator walked through the creation of a 2D side-scroller game akin to Mario. Unfortunately, this video was only slightly helpful as it helped understand that coding a game is an iterative process that requires constant tweaking and additions to get a desired result. However, it never showed the creation of classes and methods that I would need to use for a Snake game.

At this point, I was getting frustrated having only my single moving square after spending all this time researching. I decided to watch another tutorial titled “Create Snake in Python in 20 Minutes” by Baraltech. This video showed me a way of creating snake game using classes and methods but more importantly taught me the idea of “game logic” and that each object has its own logic being performed at different points in the game. After playing with this creator’s code I found that his game logic was extremely difficult to work when it came to making additions or alterations. Particularly, when it came to the snake body growth mechanism that would increase ever so slightly and was based off an algorithm that was overly complicated. So I decided to look at another video by Clear Code titled “learning pygame by creating snake”. This video was the most helpful and it’s that game logic that I heavily relied on to get the game to a certain point. I didn’t utilize all things in that video (other than the game logic and class structure code that is the basis for snake movement and food behaviour) as this tutorial uses some elements that I didn’t want. For one, graphics/images, a counter that is on screen, and it doesn’t use multiple screens to navigate through different menus (start, gameover screen). Once I was able to get the snake moving and growing along with the food behaviour, I decided to look up another tutorial to learn how to create multiple screens for a start menu and game over menu.

I was able to find a short tutorial by Baraltech titled, “how to make a menu screen in pygame”. It explained all I needed to do to create multiple screens for my game over and start menu function was to create a while loop for each screen. Luckily, this tutorial made mention of another tutorial that explained how to make clickable buttons. This tutorial was made by the same creator and was titled “easy way to make buttons for python/pygame projects”. I was able to create my button class using this tutorial and combined that information with the menu screen tutorial to develop multiple while loops that contain buttons to start the game, restart the game, and populate a score on screen. Additionally, there was a moment where I could not “bind” my mouseclicks to the coordinates the button rectangles, however, an online reference from gamedevacademy.com titled, “pygame mouse click tutorial” helped me understand the collidepoint attribute to bind mouseclicks to an objects coordinates.

## Project Development

### Pseudocode

import pygame

import sys

import random

pygame.init()

button\_font = pygame.font.SysFont("cambria", 40)

cell\_size = 30

cell\_number = 20

gamewindow = pygame.display.set\_mode((cell\_size\*cell\_number,cell\_size\*cell\_number ))

pygame.display.set\_caption("Snake")

gameloops = True

class Button:

    def \_\_init\_\_(self, image, xpos, ypos, text\_input):

        self.image = image in parameters

        self.xpos = xpos in parameters

        self.ypos = ypos in parameters

        self.rect = self.image.create rectange shape and(center it at these coordinates(self.xpos, self.ypos))

        self.text\_input = text\_input in parameters

        self.text = use the button\_font variables data to write/render these inputs to the screen(self.text\_input, True, "white")

        self.text\_rect = self.text.create rectangle and(center it at these coordinates (self.xpos, (self.ypos-12)))

    def update(self):

        draw images onto this surface gamewindow.blit using these inputs (self.image, self.rect)

        draw images onto this surface gamewindow.blit using these inputs (self.text, self.text\_rect)

class Snake:

    def \_\_init\_\_(self):

self.body = List[x and y vector positions of first object, x and y vector positions of second object, x and y vector positions of third object]

      self.velocity = adding a vector unit of 1 to the x position of the first object

      self.add\_segment = False (method that creates additional body segment is False)

    def snake\_body(self):

        for segment in self.body:(repeat this entire procedure for every element)

            xpos = take the x component of an element in the body list and multiply it by the cell\_size (30 pixels)

            ypos = take the y component of an element in the body list and multiply it by the cell\_size(30 pixels)

            segment\_shape = create a rectangle shape(the upper left corner of the shape is at xpos, the top portion of the shape is at ypos, the width of the shape is the cell\_size(30), and the height of the shape is the cell\_size(30)) this will create a square

            pygame.draw. a rectangle (onto the gamewindow, that is white, and uses the segment\_shape template)

    def snake\_movement(self):

        if self.add\_segment == True:

            body\_movement = take the entirety of the list of self.body

            body\_movement.insert into the list(at the first position(0), the x,y components of the already existing first element in the list and add the x,y components of self.velocity to create an entirely new body segment)

            self.body = becomes the entirety of list body\_movement

            self.add\_segment = is False in order to stop creating new segments rapidly

        else:

            body\_movement = the entire list of self.body except for the last element

            body\_movement.insert into the list(at the first position (0), add the x,y component value of the already existing first element of the list to the x,y component values of self.velocity)

            self.body = now becomes the entirety of the list body\_movement

    def add\_body\_segment(self):

        self.add\_segment = True

def snake\_reset(self):

        create the snake.body a completely new snake body and give it these segments = [pygame.math.Vector2(5,10),pygame.math.Vector2(4,10) ,pygame.math.Vector2(3,10)]

        self.velocity = pygame.math.Vector2(1,0)

class Snake\_Food:

    def \_\_init\_\_(self):

        self.random\_food\_position()

    def food(self):

        food\_shape = create rectangle shape(make result an integer value(xposition is derived from self.x variable then multiply it by cell\_size), make result an integer value(yposition is derived from self.y variable then multiply by cell\_size))

        pygame.draw. a rectangle (onto the gamewindow, that is blue, and uses the food\_shape template)

    def random\_food\_position(self):

        self.x = random.randinteger value between values (0 and cell\_number-1))

        self.y = random.randinteger value between values (0 and cell\_number-1)

        self.pos = x,y values of food position become x,y vector values of variables above(self.x,self.y)

class MAIN:

    def \_\_init\_\_(self):

        create an instance of the snake (self.s) = Snake()

        create an instance of the snake food (self.sf) = Snake\_Food()

    def update(self):

        update these methods every 120 milliseconds (derived from custom event type, SCREEN\_UPDATE)

self.s.snake\_movement()

        self.snake\_eat\_mechanism()

        self.collision()

    def object\_shapes(self):

method as a shorthand to populate these objects on screen within game loop

        self.sf.food()

        self.s.snake\_body()

    def snake\_eat\_mechanism(self):

        if vector values x,y of snake food == vector values x,y of the head of the snake:

            create self.sf.random\_food\_position()

            create self.s.add\_body\_segment()

        for any segment in the body (excluding the head):

            if segment == the vector values x,y of the snake food:

                create self.sf.random\_food\_position()

    def collision(self):

        if the x value of the head of the snake is not between 0 and cell\_number

            call the game\_over\_screen() function

        if the y value of the head of the snake is not between 0 and cell \_number

                        call the game\_over\_screen() function

        for any segment in the snake body (excluding the head):

            if segment == the head of the snake:

            call the game\_over\_screen() function

#initializes class MAIN()

main = MAIN()

SCREEN\_UPDATE = pygame.USEREVENT (creates custom user event)

pygame.time.set\_timer(SCREEN\_UPDATE,120) (custom user event is predicated on a timer of 120 milliseconds)

def game\_over\_screen():

    gameoverloop= True

    gameoverwindow = pygame.display.set\_mode((cell\_size\*cell\_number,cell\_size\*cell\_number ))

    pygame.display.set\_caption("Game Over")

    button\_surface = load this image in the directory("button.png")

    button\_surface = take the loaded image and rescale it to these pixel coordinates(button\_surface, (300,200))

    game\_over\_button = create object with these parameters Button(button\_surface, 300,300, "Restart")

    while gameoverloop:

        gameoverwindow.fill("white") fill the screen with white

        GAME\_OVER\_TEXT = take the data from button\_font variable and create/render these inputs onto screen ("GAME OVER", True, "red")

        SCORE\_TOTAL\_TEXT = take the data from button\_font variable and create/render these inputs onto screen ("your score was:" + " " + \*\*this is the score function\*\*str(len(main.s.body)-3), True, "green")

        GAME\_OVER\_RECT = create rectangle for GAME\_OVER\_TEXT and (center it at=(300,100))

        SCORE\_TOTAL\_RECT = create rectangle for SCORE\_TOTAL\_TEXT and(center it at=(300, 150))

        Put these images onto this window gameoverwindow.blit(SCORE\_TOTAL\_TEXT, SCORE\_TOTAL\_RECT)

        Put these images onto this window gameoverwindow.blit(GAME\_OVER\_TEXT, GAME\_OVER\_RECT)

        game\_over\_button (initialize button)

        game\_over\_button.update() (call button class update method)

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                sys.exit()

            if event.type == pygame.MOUSEBUTTONDOWN:

                create tuple of coordinates x,y = pygame.mouse.get\_pos()

                if the game\_over\_buttons rectangle collides at these points (x,y):

                    then print("Game Restarted!")

                    create a new snake main.s.snake\_reset()

                    call this function (goes back to gameloop while loop)play()

        pygame.display.update()

def play():

    pygame.display.set\_caption("Snake")

while gameloops:

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            pygame.quit()

            sys.exit()

#Custom event that calls instances within update method to update themselves

        if event.type == custom event SCREEN\_UPDATE:

            call this method main.update()

        #Movement control

        if event.type == pygame.KEYDOWN:

            if event.key == pygame.K\_DOWN:

                if y vector value does not equal -1:

                    change vector velocity value in the y direction to 1 (0,1)

            elif event.key == pygame.K\_UP:

               if y vector value does not equal 1:

                    change vector velocity value in the y direction to -1 (0,-1)

            elif event.key == pygame.K\_LEFT:

                if x vector value does not equal 1:

                    change vector velocity value in the x direction to -1 (-1,0)

            elif event.key == pygame.K\_RIGHT:

                if x vector value does not equal -1:

                    change vector velocity value in the x direction to 1 (1,0)

    fill gamewindow with color black on every iteration of the gameloop

    populate the shapes from the method object\_shapes from the main class on every iteration of the gameloop

    update the display of the game every iteration of the gameloop (pygame.display.update)

def start\_menu():

    startmenuloop = True

    startmenuscreen = pygame.display.set\_mode((cell\_size\*cell\_number,cell\_size\*cell\_number ))

    pygame.display.set\_caption("Start")

    button\_surface = load this image in the directory("button.png")

    button\_surface = take the loaded image and rescale it to these pixel coordinates(button\_surface, (300,200))

    game\_over\_button = create object with these parameters Button(button\_surface, 300,300, "Restart")

    while startmenuloop:

        startmenuscreen.fill("white")

        START\_MENU\_TEXT = take the data from button\_font variable and create/render these inputs onto screen button("SNAKE", True, "red")

        START\_MENU\_RECT = create rectangle for START\_MENU\_TEXT and (center it at=(300,100))

        Put these inputs onto screen startmenuscreen.blit(START\_MENU\_TEXT, START\_MENU\_RECT)

        start\_button (intilize button object)

        start\_button.update() (calls the button class update method)

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                sys.exit()

            if event.type == pygame.MOUSEBUTTONDOWN:

                create a tuple of coordinates x,y = pygame.mouse.get\_pos()

                if the start\_buttons rectangle collides with mouseclick at these points(x,y):

                    then print("Game Started!")

                    and call this function (starts gameloop while loop)play()

        pygame.display.update()

start games automatically with a call to the start\_menu() function

### Flowchart

Snake Game Flowchart

(This flow chart is rather large, so I’ve split it into 4 screenshots starting from left going to right, with the 4th picture being the full chart)

**LEFT MOST: Begins with start\_menu loop**

A diagram of a diagram

AI-generated content may be incorrect.

**MIDDLE: Centered around the gameloop which is responsible for gameplay**

A diagram of a company

AI-generated content may be incorrect.

**RIGHT MOST: game\_over\_screen loop back to calling the play() function to restart the game**

A diagram of a diagram

AI-generated content may be incorrect.

**FULL VIEW OF FLOWCHART**

**A diagram of a company

AI-generated content may be incorrect.**

### UML Diagram

A diagram of a computer

AI-generated content may be incorrect.

### Requirements

1.Board Size / Play Area --- have  
2. Snake Movement --- have  
3. Snake Growth--- have  
4. Food Generation--- have  
5. Collision Detection--- have  
6. Game Over and Score--- have  
7. Restart Option --- have