Final Project Design Document

By Chandler Gibson

Introduction

Project Functionality

Simple and straightforward, my Final Project, “Generic Snake Game but With Randomized Grid Size,” leverages the ⬆️, ⬇️, ⬅️, and ➡️ keys to guide a snake around a gridded game board. The game’s objective is to rack up as many points as possible by eating (running into) “frogs,” grid tiles marked green. Every time a frog is eaten, a score indicator at the top of the board increments by one point, starting from zero. Only one frog, or “food” tile is present on the grid, and that food item is automatically respawned in a random grid location whenever it is eaten. Leveling up from its feasting, the snake grows longer by one tile for every point gained. However, the game ends* when the snake crashes into either itself or the walls of its enclosure (the edges of the game board). As a side note, one functionality error—or, as I prefer to think of it, a feature—existing in my code is that food can spawn directly on top of the snake’s head, thereby incrementing score instantaneously.

*In principle, it is possible to run the game forever in one sitting since a restart option exists.

Design Process

I created my final project using the “Create Snake with Python in 20 MINUTES!” tutorial* as the basic template for my code. Surprisingly, I had no technical reason for using that specific tutorial; I simply preferred the blocky aesthetic presented in the video over the aesthetics of other programming guides. Despite my use of code templates, “Generic Snake Game but with Randomized Grid Size” differs significantly from other Snake games. For example, each time the game is turned on, the program renders a randomly-scaled game board, ranging from 17-by-10 tiles to 19-by-14 tiles. In contrast, other Snake games output a fixed board size. Furthermore, the “Game Over” and restart mechanisms utilize my own unique code and graphics.

Problems I encountered during the production of my Snake game included issues with PyGame (an extension of Python used to create video games) installation, as well as troubleshooting throughout the debugging process. Nonetheless, there was still a part of the Final Project that I enjoyed; namely testing, which proceeded smoothly.

*From the baraltech YouTube channel.

Project Development

Pseudocode

This is the design, represented as pseudocode, upon which I based my algorithm*:

```
Initialize snake position**
Direction = Right**
Score = 0**
While (game is running):
   Hide Game Over screen
   Allow Snake to move
   If (⬆️ key pressed):***
      Direction = up
   If (⬇️ key pressed):***
      Direction = down
   If (➡️ key pressed):***
      Direction = right
   If (⬅️ key pressed):***
      Direction = left
   Move Snake in Direction***
   If (the snake has eaten an apple):****
      Increment score by 1****
   If (snake has collided with the walls or itself):*****
      Initialize snake position*****
      Direction = Right*****
      Score = 0*****
      If (spacebar key pressed):*****
         Proceed*****
      Else:*****
         Show Game Over screen*****
         Disallow snake from moving*****
         Wait until key spacebar pressed*****
```

---

*Some features of the final algorithm are absent from the pseudocode since I hadn't thought of them until after I had already begun coding my project.

**Sets the parameters for the start of the game

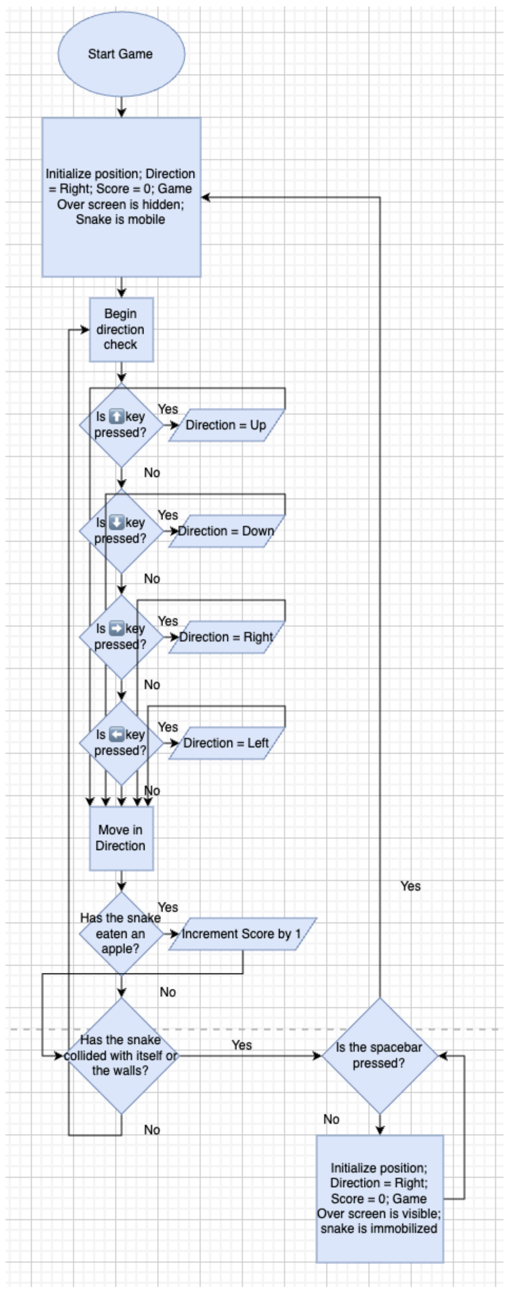
***Controls the mechanism for snake movement

****Scoring system

*****Game Over/Restart mechanism

Flowchart

This is the design, represented as a flowchart, upon which I based my algorithm*:


Start Game
Initialize position; Direction = Right; Score = 0; Game Over screen is hidden; Snake is mobile
Begin direction check
Is ⬆ key pressed?
Yes
Direction = Up
No
Is ⬇ key pressed?
Yes
Direction = Down
No
Is ➡ key pressed?
Yes
Direction = Right
No
Is ⬅ key pressed?
Yes
Direction = Left
No
Move in Direction
Has the snake eaten an apple?
Yes
Increment Score by 1
No
Has the snake collided with itself or the walls?
Yes
No
Is the spacebar pressed?
Yes
No
Initialize position; Direction = Right; Score = 0; Game Over screen is visible; snake is immobilized

*Some features of the final algorithm are absent from the flowchart since I hadn't thought of them until after I had already begun coding my project.

Requirements

The document “Final Project Instructions” lists the following requirements for a game of Snake, each of which my project satisfies:

1. “The game should be played on a window containing a 10-by-10 tile set.”

Although the game’s grid dimensions are randomized every time the game is run, the lower bounds for those dimensions correspond with a 17-by-10 tile set. Therefore, the number of tiles along any given edge of the board is always greater than or equal to 10.

2. (Part One): “The snake should move continuously in one of four directions (up, down, left, right).”

My project satisfies this requirement since the snake travels in only four possible directions, and at a constant speed of 16 tiles/second: right (0º angle), up (90º angle), left (180º angle), and down (270º angle).

2. (Part Two): “The player should control the direction using the arrow keys.”

My project satisfies this requirement since the snake’s direction of travel is dictated by keys pressed by the player: the ⬆️ key to shift the snake upwards (90º angle), the ⬇️ key to shift the snake downwards (270º angle), the ⬅️ key to shift the snake to the left (180º angle), and the ➡️ key to shift the snake to the right (0º angle).

3. “Every time the snake eats a food item, it should grow longer by one unit.”

Whenever the head of the player’s snake intercepts a randomly-selected tile on the grid (marked green to represent a frog), the snake’s onscreen representation is rendered one tile longer than before.

4. “Food should appear randomly on the game screen after being eaten by the snake.”

Once the snake's head overlaps with the "food" tile, that intercepted tile reverts to being a normal tile. Meanwhile, another randomly-selected tile on the board is marked as the next "food" tile.

5. "The game should end if the snake collides with" either "the walls of the screen" or "its own body."

Upon colliding with itself or the edges of the gameboard, the snake has its body length reduced to one tile (excluding the head). Additionally, it is reset to its starting position and immobilized.

6. (Part One): "Display a Game Over message when the snake dies."

When the snake collides with itself or the edges of the gameboard, the background grid is concealed by a wall of text reading "Game Over." Moreover, the tile currently designated as "food" disappears, and the score is reset to zero.

6. (Part Two): "Display the player's score, which is based on how many food items the snake has eaten."

A score indicator ticks at the top of the screen, starting as "0" and progressing to "1," "2," "3," and so on as the snake consumes the green tiles.

7. "There should be an option to replay the game after a game over."

Pressing the spacebar after the game's end will cause the snake to continue moving as normal. Additionally, the "Game Over" text wall is cleared away while a new randomly-chosen tile on the gameboard is marked as food.