Snake Game Manual

Mitchell Marino – mitchelltmarino@gmail.com

Overview

Instructions

This program is a replication of the classic game 'Snake'. This version of the game takes place within a 29-block width by 29 block height environment. The user controls a snake which continuously moves in the direction its' head is facing. If the snake runs into one of the outer boundary walls or its' own body, it will result in fatal injury to the snake and the player will lose the game as a result. An apple will always be spawned on the map, at a randomly generated location. The snake can eat apples by running over them. When the snake eats the apple that is on the map, the snake will grow 1 unit longer and another apple will appear in another randomly generated location. The goal of the game is to control the snake in such a way that it eats as many apples as possible (i.e. the snake grows as long as possible) without dying. This gets harder the larger the snake gets, because there will be less available space for the snake to move in its' environment, so the game gets very strategic as the snake approaches longer lengths. There are brief instructions and more on the application sheet.

Controls

- **Spacebar**
  - Pressing the spacebar will start (or restart) the game.
- **Arrow Keys** ○ Pressing an arrow key will make the snake's head face the respective location of the arrow key.
  - Up arrow → Snake's head will face upward.
  - Down arrow → Snake's head will face downward. ○ Left arrow → Snake's head will face to the left. ○ Right arrow → Snake's head will face to the right.
  - Note:
    - ✦ The snake cannot instantly turn backwards; it must go around in a semi circle. (The snake cannot backtrack over its body)

Design

The application is fitted with a simple and elegant design. It consists mainly of light shades of blue as an outline colour, black text for clarity, white background for transparency, and a touch of peach to compliment header spaces. This leads to an effective layout for the program that not only looks nice but puts emphasis on the ease of use. The body of the snake is black, while the head is blue. Apples are red.

Functionality

High Scores

The snake application uses a database to keep track of high scores. The application is capable of both reading and writing to the database. Upon first attempting to access the high scores (for either reading or for input) the user will be prompted for the database file of which the high scores are stored. The user will have the option to select the file via a file dialog that opens. Input is checked for validity.

Timer

I originally had planned to create a delay in processing by just using a sleep method, but I realized that it was not possible to do so because user input was not being picked up whilst the programs processes were halted by sleep. In order to create the timer, I had to access a couple of Microsoft's libraries that are available for use. This includes the "user32" and "kernel32" libraries. Upon the user starting the game, the timer starts. The timer ticks every 200ms and runs a single cycle for the game simulation. The way this works is a tick triggers the TimerEvent subroutine which then calls the Game_Run subroutine. This is the main process that runs the game. When a collision is detected, the timer is stopped.

Score

Score is equivalent to the length of the snake, and is tracked using a global variable throughout the entire duration of the game. Since the starting length of the snake is 3, the minimum score possible for a user to achieve is 3.

Snake

The way the program keeps track of the snake is through an array of ranges. The array is re-dimensioned every time the snake grows. Every time the snake moves in a direction, each index takes the value of the index prior to it. Then the range at index 0 takes on the value of the range of its current location, offset in whichever direction the head is facing. It is worthy of noting that the entire snake is not painted at once, but rather only the head is painted and the rear is erased during each movement.

Collision

The main method that the program determines whether a collision is occurred or not, is by examining the colour of the block the head of the snake is about to traverse. If the colour is red, the snake has eaten an apple. If it is white, the snake is traversing empty space. If it is not red or white, a collision has occurred.

FAQ:

Q: Why did you include an initialize button?

A: It generally isn't needed because the program assigns keypress values for the space bar and arrow keys upon opening the workbook. But assuming the possibility that someone was to open the workbook and enabling macros at a different time, this would allow that user to play the game without having to reopen the workbook.

Q: How do I view the High Scores?

A: Click the 'view high scores' button, then select the database file which you plan to read the data from. (the file HighScores.mdb is included by default)

Q: How do I add my High Score?

A: Click the 'submit high scores' button, then select the database file which you plan to read data from. (the file HighScores.mdb is included by default) Then input your first name and last name into the correct fields. These fields are checked for valid input, so it will catch any common improper name formatting.

Q: Why not lock certain cells on the spreadsheet?

A: Allows for easier for testing.

Q: Why did you choose to do snake?

A: I thought that snake would be both fun and challenging to do. I would rather do something mildly algorithmic, fun to look at and interactive than create an office tool!

Q: What was the most challenging part?

A: The most challenging part by far was the timer. The timer alone took me multiple hours. Mainly because I had to learn how it worked before using it. However, the timer definitely made the process easier once I got it working.