Implementation of the board game *Hex* in Microsoft Excel using Visual Basic for Applications (VBA)

Adilet Baimyrza uulu

January 23, 2023

Introduction

This is the documentation to the implementation of the popular board game *Hex* in Microsoft Excel using Visual Basic for Applications (VBA). This documentation consists of the short description of the game, the way how the game board was created, its code components, where the logic behind every subroutine is explained, and the biggest challenges faced during the creation of the game, as well as sources used during creation.

Game Logic

Hex is a two-player abstract strategy board game in which players attempt to connect opposite sides of a game board. It is played on an 11×11 board, which is composed of hexagon-like cells called *hexes*. Each player is assigned a pair of opposite sides of the board, which they must try to connect by alternately marking hexes with either red or blue. The *Red player* must connect upper side with the bottom side, while the *Blue player* must connect the left side with the right side. A player wins when they successfully connect their sides together through a chain of adjacent same-color hexes. Draws are impossible in Hex due to the topology of the game board.

Game Board

The game board is made up of shapes representing hexagons, it has one *button Start* and a box that keeps track of players' turns. Each hex has equal sides and was grouped and aligned according to the game's conventions. Each hex has been numbered from 1 to 121 and given names like so: "Hex 1", "Hex 2", … "Hex 121". Hexes were made clickable, so it calls the *selectHex* subroutine and marks the hex either blue or red. Lastly, the *Start button* is a regular button, and it calls *startGame* subroutine and sets the color of all hexes to grey.

Code Components

1. playerTurnDisplay

turn – global variable that keeps track of players' turns.

This subroutine updates a rectangular shape on the spreadsheet to indicate current player's turn (red or blue) by changing the color of the shape. It uses the *turn* global variable to determine the current player's turn. If the *turn* variable is odd, then it is blue player's turn. Otherwise, it is red player's turn.

2. selectHex

red(11, 11) – array that keeps track of marked hexes for the *Red player*.

Blue(11, 11) – array that keeps track of marked hexes for the Blue player.

Path(11, 11) – array that keeps track of hexes that were visited already. It is used by both players. After calling the *checkRed* of *checkBlue* subroutines, the members of *path(11,11)* are being reset to default values.

This subroutine is called, when the current player clicks on a hex. It is very important subroutine, because it calls all other subroutines that are vital for the game logic. Firstly, it checks the color of the hex: if the hex is either blue or red, it does nothing. That way it makes sure the color of the hex does not change over the duration of the game. If it is grey however, it extracts the number of the hex as an integer from its name and figures out the location of that cell in the 2-dimensional array in the format of *(x, y)*. Thirdly, it checks the player's turn by checking the value of the variable *turn* and increments it. After that, it calls the *playerTurnDisplay* to update the display box. Then, it assigns the Boolean value of the member of either *red(11,11)* array or *blue(11,11)* array to *True*. Lastly, it checks the first line of the hexes. If they are marked, the subroutine calls the *checkRed* or *checkBlue* for further computation.

3. startGame

This subroutine sets up the board for a new game by resetting all the hexes to default grey color and the all global Boolean arrays to *False*. It is by far the simplest subroutine in this game.

4. checkRed and checkBlue

These subroutines are called to check whether the current player (red or blue) has won by connecting one of their hexes to the opposite edge of the board. If a player has won, a message box is displayed to indicate this.

Firstly, it checks the *path(11, 11)* member at *(x, y)* coordinate that was passed to subroutine as parameters *x* and *y*. If the hex was visited, it exits the subroutine. If it was not, then, it indicates that this hex was visited and checks for the *win condition* by checking, if the hex is on the opposite side of the box. If it is not on the other side of the board then it checks for its neighbors as well as their range to always operate within the board game. Once it reaches the neighbor that is marked by one of the players, then it calls the subroutine on the neighbor. Those subroutines are recursive, and they will run until the *win condition* is satisfied.

Determining The Winner

The game is won when a player has connected one of their hexagons to the opposite edge of the board. The *checkRed* and *checkBlue* subroutines are called to check for this win condition. They use recursion to check all the adjacent hexagons to the current one and if it's the same color as the current player, it will call the function again with that hexagon as the current one. If the function reaches the opposite edge of the board with a player's color, it will display a message box indicating that the player has won.

Challenges Faced

1. Translating The Position Of The Hex To The 2-dimensional Array

One of the biggest challenges was to find out how to let the program know which shape I clicked and establishing its place in the 2-dimensional array. Initially, I knew that I can have access to the shape that is clicked by using *Application.Caller*, and change some of its properties like changing its color. It wasn't of much use back then, until I found out that I could access the name of the shape too by using *Application.Caller*. So, I hard-codedly changed the name of each shape on the board and numbered them from 1 to 121. Next, I split their names as strings to get access to the number of the shape and changed it to the integer type to make further calculations. Using simple formula, I figured out the exact location of the shape in the 2-dimensional array. And now, I was able to change not only color, but the value of its corresponding arrays' members. Now it was left for me to write an algorithm to determine the winner.

2. The Algorithm For Determining The Winner

It was hard for me to determine the winner, when I was contemplating on that matter and not coding. I searched a lot on the internet for possible algorithms. They seemed very complicated like

Djikstra algorithm, *Depth-First Search*, *Breadth-First Search*. I did not get how to implement those algorithms in code, so I gave up on them and started from scratch. It was essential to find the relationship of hexes in the 2-dimensional array and slightly deformed board that apparently didn't look like a 2-dimensional array. I found out that each hex could possibly have 6 neighbors, and I wrote code to get the indexes of those neighbors. Next, I needed to make sure that the hexes that have less than 6 neighbors were not trying to reach the invalid member of the array, so I made a range-check as well. Now, with having access to the neighbors of every hex I could call the *checkRed* or checkBlue recursively and checking for *win condition*. And Voila! Game done!