

Software Requirement Specifications(SRS)

Project 2048

Group no: 12

Team Members—

- 1) RaseshSaraiya – 145060003
- 2) AmalVats – 14D260008
- 3) SanketChirame—14D260003
- 4) Sharang Bhagdikar - 14D100018

Purpose:

The purpose of this document is to give software specifications of the 2048 coding project. This document briefly overviews the structure of code decided up to first stage of project. It also outlines the scope of project and user interface provided for game. Document states brief specifications of functions designed for basic moves. The game is exclusively designed for windows interface.

Scope:

2048 is a single player puzzle game which went viral earlier this year on PlayStore. It was created by a 19 year old Italian web designer, Gabriele Cirulli. The objective is to slide numbered tiles on a rectangular grid and combine them in accordance with certain rules, to create a tile with the number 2048.

The 2048 game is played on 4x4 grid, with all the tiles but two, initially blank. The player may have either of the numbers 2 or 4. Then, the user (player) uses the arrow keys to move the entire filled tiles left, right, up or down till the point that the last column or row (whatever the case may be) meets the boundaries. The tiles containing similar numbers add up, for eg- 2+2 will give a resultant tile containing 4 and so on.

The player wins when a tile containing 2048 is formed. But if the entire grid is filled with numbers without any of them being 2048, the player loses.

A scoreboard will keep track of the player's moves. The score is added consistently with the numeric content of any tile that is formed. Also, whenever a new tile is formed by addition, a random tile will also be generated containing 2 or 4.

User Controls- The entire filled tiles are moved in the 4x4 grid by the use of arrow keys-

- 1) R key – Move right
- 2) L key – Move left
- 3) U key – Move up
- 4) D key – Move down

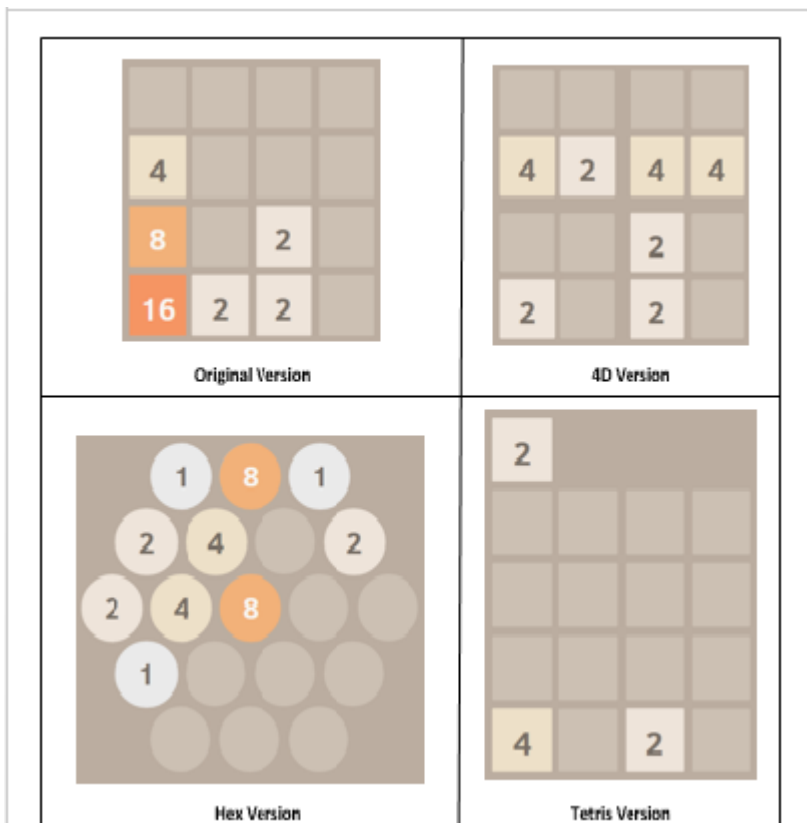
Our gaming app goes ahead providing the four different versions of the game in one place. After starting the game, user will swipe numbers according to rules of particular game mode to get the tile of '2048'.

References:

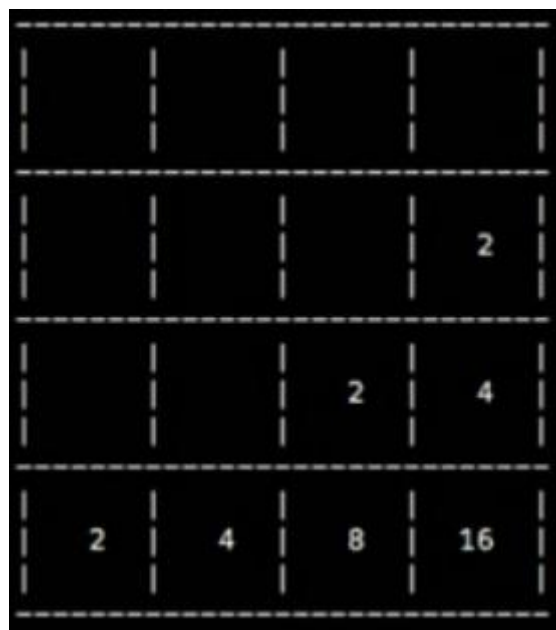
- *Normal 2048-* <http://gabrielecirulli.github.io/2048>
- *2048 4D-* <http://huonw.github.io/2048-4D/>
- *2048 Hex-* http://rudradevbasak.github.io/16384_hex/ (*slightly modified*)
- *2048 tetris -* <http://prat0318.github.io/2048-tetris/>

Interface requirements: Playing the game requires only a keyboard as an input device and a monitor to display the output.

Graphic Interface: (Tentative; colors and other design details may vary in the final submission):



Text-Based Interface: If we run into a problem implementing the graphic version, then the following version will be implemented-



Working of the game –

The game, as mentioned, would work with arrow keys.

1)U Key-

Whenever the up arrow key is pressed during the game, the numbers would move up till the boundary is approached, and similar numbers would add along the way. For a demonstration,

Before pressing the U key, let the grid be of this form:-

		4	2
		4	4
2	2	2	2
8	16	4	2

After pressing the arrow key, the grid would look something like:-

2	2	8	2
8	16	2	4
		4	4
2			

 (Randomly generated tile containing the number 2 in this case)

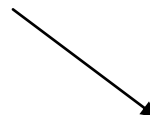
2) D key-

In the newly created grid,

2	2	8	2
8	16	2	4
		4	4
2			

If we press the D key, then the grid would look something like-

(The randomly generated number tile, 4 in this case)



			4
2		8	
8	2	2	2
2	16	4	8

3)In the grid formed,

			4
2		8	
8	2	2	2
2	16	4	8

If we press the R key, the grid would be-

(The randomly generated number tile, 2 in this case)



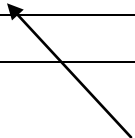
2			4
		2	8
	8	2	4
2	16	4	8

4) If we press the L key now,

2			4
		2	8
	8	2	4
2	16	4	8

The grid would be-

2	4		
2	8		
8	2	4	2
2	16	4	8



(The randomly generated number tile, 4 in this case)

Through the above grids and their content, we see that if two tiles containing the same number are located next to each other and if the arrow key is pressed in the same direction, the numbers get added up and through this way, we have to create 2048 in a tile in order to win the game.

Algorithms-

For implementation of all the project contents, we are firstly developing functions to complete the original 2048, which shall in turn be used in further variations of the game.

So far, we have completed 7 functions –

1) Display Board- This function asks for the initial input which is displayed in a 4x4 grid format. However, by default the game has only 2 random tiles with a 2 or a 4 in them.

2)Random Tile Generator- Whenever the user presses the arrow keys, and some two tiles add up, a new tile containing a number is generated.

This function contains an array, which stores all the entries of the grid as 0 or 1. 0 means blank tile and 1 means that the tile contains a number. Thus it identifies the blank tiles in a 2048 grid. Then, it inserts a number in any of the tiles with equal probability. However, the probability of insertion of a 2 and a 4 is different. The number 2 is inserted with a 90% probability and the number 4 is inserted with a 10% probability.

3) Collapse Down Function- This function comes into play whenever the user presses a key. If the user presses the D key, then this function, with the help of the **Swapper Function**, identifies if the numbers in the particular column are filled or not and according to that, the numbers are rearranged in that particular column. This works through a loop and after the first column, the second column rearranges and so on through another loop.

4

Becomes

2
4

4) Collapse Up Function-This function comes into play whenever the user presses an arrow key. If the user presses the U , then this function, with the help of the **Swapper Function**, identifies if the numbers in the particular column are filled or not and according to that, the numbers are rearranged in that particular column. This function picks up the top tile and goes progressively down. This works through a loop and after the first column, the second column rearranges and so on through another loop.

2
4

Becomes

2
4

5) Collapse Left Function- This function comes into play whenever the user presses an arrow key. If the user presses the L key , then this function, with the help of the **Swapper Function**, identifies if the numbers in the particular column are filled or not and according to that, the numbers are rearranged in that particular column. This function picks up the

tile from the leftmost column and goes right. This works through a loop and after the first row, the second row rearranges and so on through another loop.

	4	16	8
--	---	----	---

Becomes

4	16	8	
---	----	---	--

6) Collapse Right Function- This function comes into play whenever the user presses the R key. If the user presses the R key, then this function, with the help of the **Swapper Function**, identifies if the numbers in the particular row are filled or not and according to that, the numbers are rearranged in that particular column. This function starts from the rightmost tile and goes left. This works through a loop and after the first row, the second row rearranges and so on through another loop.

4	16	8	
---	----	---	--

Becomes

	4	16	8
--	---	----	---

7) Swapper Function- This function is called inside the Collapse functions. When an empty tile is identified, this function looks for the next filled tile in the particular row or column(according to the function call) and when it finds one, it swaps them and in this way it helps rearranging the entire row or column (according to the function call).

8)Merge Function- Since we have not completed it so far, we shall not be defining it much here and won't be submitting it's cpp file. But we have begun work over it and this function would basically be used to add tiles containing same numbers next to each other and which would combine with the other functions to complete the original version of the 2048 game.

8) Merge Up Function- This function would add up the adjacent tiles if the numbers present in them are same. If the U key is pressed, and the numbers are collapsed together as well as if the numbers adjacent are same, they will add up.

9) Merge Down Function- This function would add up the adjacent tiles if the numbers present in them are same. If the D key is pressed, and the numbers are collapsed together as well as if the numbers adjacent are same, they will add up.

Post Stage 1 Functions:

10) Undo function: On pressing 'u', the user can go back to the previous state of the board. It was noticed that pressing 'u' as soon as the game started (i.e. before any move was made) resulted in the entire board being filled up with garbage values. Hence a provision has been made for the same. (Message displayed is : No move made to undo)

11) Bomb function: The user is provided with 5 so called 'bombs' at the start of the game which he can use to remove an unwanted tile at his discretion. On pressing 'b', the user is asked for the row and column of the tile to be set to 0. If the range of the entered coordinates exceeds 4 or is below 1, and error message is displayed. Similarly, 'bombs exhausted' is displayed if bombs = 0.

12) display_board: This is the final interface the user sees. It consists of the board, the score and the number of bombs left.

13) reset : Firstly, empty_index is array that has 16 rows and 2 columns with the values of all the elements set to -1. When reset is called the entire array is set to -1. Then, the array board is traversed and all the zero elements's coordinates are stored in each row of empty array. On storage of one set of coordinates, the variable **counter** is incremented by 1. By the end of the iteration, counter stores the total number of empty tiles and empty_array stores the coordinates of the same.

14) copy_board: This function copies the entire contents of 'board' to another array called 'temp_board'. This function seems trivial but is indispensable when it comes to other functions that involve invalid move checking and undo.

15) check_array: The return type of this function is bool. It returns **true** if any element of temp array does not match with the corresponding element of board array and **false** otherwise.

16) game_over: In 2048, the game is over when a move made in any direction does not cause a change in the configuration of the board. To check this, in the game over function, each move is made and the configuration of the board is compared with temp array. If no change is detected in any move, the game is over.

17) save function: The game is saved in a file named 'SAVED_GAME.txt'. This is done by initializing a file pointer and using the function **fprintf** to extract the values stored in the board array and stored in new lines in the file. Once four tiles of a given row are stored, another new line is inserted for readability. The last figure represents the score at the time of saving the game.

18) openfile: This functions is used to resume a saved game. The tiles are stored in the format mentioned in the previous paragraph. Now this same file is opened in read mode and the data is extracted using the function **fscanf** and stored in the board array. The last figure represents the score and is stored in the variable **score**.

19) delete_savedgame: This results in the existing figures of the file 'SAVED_GAME.txt' being replaced by zeroes. This is done by using **fprintf** with parameter (fpout,"%1d",k), where k=0.

20) display_4d_board: This is similar to the display_board function, the only difference being that it contains four boards of four cells each instead of one board of sixteen cells.

21)left, right, up, down move of 4d board: This is similar to the functions used in the regular version of the 2048 game. It executes as followed by the reset function which populates empty_array with the coordinates of the empty tiles. Function check_array is used later to check if an invalid move is being made. If not, a random tile is generated using generate_tile. The board is then displayed with the tiles shifted appropriately and copied to array temp.

Graphics- For graphics, though initially we thought upon using the simple cpp library, but later we agreed to use EZWindows for implementing our graphics. But in the project, we have so far not delved much into the graphics part.

Structure of Hex 2048 (Devised by Sharang, Documented by Rasesh):

From the photo added above, we have a hexagonal system consisting of rows of length 3, 4, 5, 4, 3 stacked one on top of the other. The moves are slightly different here with diagonal moves permitted. They are: North-east, North-West, South-East and South-West. Crow stands for copied row which stores the same values for processing later on in the program.

Store_coordinates: This is similar to reset function of the simple version. The coordinates of the empty cells are

New functions added in Hex: