2048

Introduction

Our main motive was to recreate the popular game “2048” and to revitalize the game in a new light by adding a couple of new features which may make the game even more fun to play.

Team Members:

1. Mohammad Yehya Hayati , 21K-3309
2. Muhammad Sufyan, 21K-3206

Background:

2048 is a single-player sliding tile puzzle video game written by Italian web developer Gabriele Cirulli and published on GitHub. The objective of the game is to slide numbered tiles on a grid to combine them to create a tile with the number 2048; however, one can continue to play the game after reaching the goal, creating tiles with larger numbers. It was originally written in JavaScript and CSS over a weekend, and released on 9 March 2014 as free and open-source software subject to the MIT License. Versions for iOS and Android followed in May 2014. But here we have created this game in MASM Assembly language implementing all Assembly Language concepts which includes arrays, stack, registers, loops, functions, labels, direct Offset addressing, jump and compare instructions.

Features:

A simple single player 2048 Game with some new features. The logic of the game is simple as we all we need to do is check if the same number is in its direction of movement and if its then combine the two numbers. If the whole board is full then game over. The player will be able to control the movement of the numbers using the arrow keys and that in turn will update a 2D grid. After every move, the game will check if the number 2048 exists on the board; if it does then the player won, however if the board is full with no numbers that can add, and there is no 2048 on the board then the player has lost. Here is a picture of the game:

This 2048 game strategy can be broken down into a few key elements:

1. Use only two directions (as much as possible)
2. Never move your tiles up.
3. Keep your tiles tidy.
4. Focus on your goal.
5. Aggressively combine downward and horizontally as needed.

Tools and Technologies:

The IDE that is used for the creation of this game is Visual Studio and we used MASM language using Irvine32 library.

Implementation & Testing

First we had to make the game itself which requires the random generations of numbers at any random empty spot. This was done by randomly selecting an empty space from the board array and putting a number that is randomly picked from an array filled with 2's and 4's and all of this is done while the board is not full. This was done using two user-defined functions IsFull and UpdateGrid.

```
245   ;-----------------------------------------------------
246   ;A function that puts a value in a random spot on the Grid; Value set in EAX
247   UpdateGrid PROC USES ebx
248   ;-----------------------------------------------------
249       call IsFull
250       cmp eax, 1
251       je UpdateGridRet
252   FindSpot:
253       mov eax, 16
254       call RandomRange
255       mov ebx, Grid[eax * TYPE DWORD]
256       cmp ebx, 0
257       je FoundSpot
258       jmp FindSpot
259   FoundSpot:  mov Grid[eax * TYPE DWORD], 2
260   UpdateGridRet:ret
261   UpdateGrid ENDP
```

Then we print the board on the console and all the numbers are color coded which was done so that it is easier to differentiate between numbers. Then we needed to make the most integral part of the game which is the moving of the numbers. We had to make sure that only two numbers could add not more than that and we also had to make sure that only the same numbers are added. This was done using a user-defined function keystroke .

```
678  PerformMoveRight:
679      push 1
680      mov ecx, 0
681      L1Right:
682          cmp ecx, 4
683          jge EndStroke
684          mov i, ecx
685          mov ecx, 2
686          L2Right:
687              cmp ecx, 0
688              jl EndL2Right
689              mov j, ecx
690              mov a, ecx
691              L3Right:
692                  cmp ecx, 3
693                  jge EndL3Right
694                  mov eax, ecx
695                  inc eax
696                  mov ebx, i
697                  IMUL ebx , row
698                  add eax , ebx
699                  shl eax, 2
700                  mov edx, a
701                  add edx, ebx
702                  shl edx, 2
703                  mov ebx, 0
704                  cmp [esi + eax], ebx
705                  jne elseifRight
706                  mov ebx, [esi + edx]
707                  mov [esi + eax], ebx
708                  mov ebx, 0
709                  mov [esi + edx], ebx
710                  inc a
711                  jmp LOOPRight
712              elseifRight:
713                  PUSH eax
714                  PUSH edx
715                  mov eax, [esi + eax]
716                  mov edx, [esi + edx]
717                  cmp eax, edx
718                  jne POPRightbreak
719                  POP edx
720                  POP eax
```

Next we needed to check if the user has won or lost the game. If the number 2048 exists on the board then the user has won. However, if there are no possible ways/combinations to win then the user has lost. This is done using two user-defined functions GameLost and GameWon .

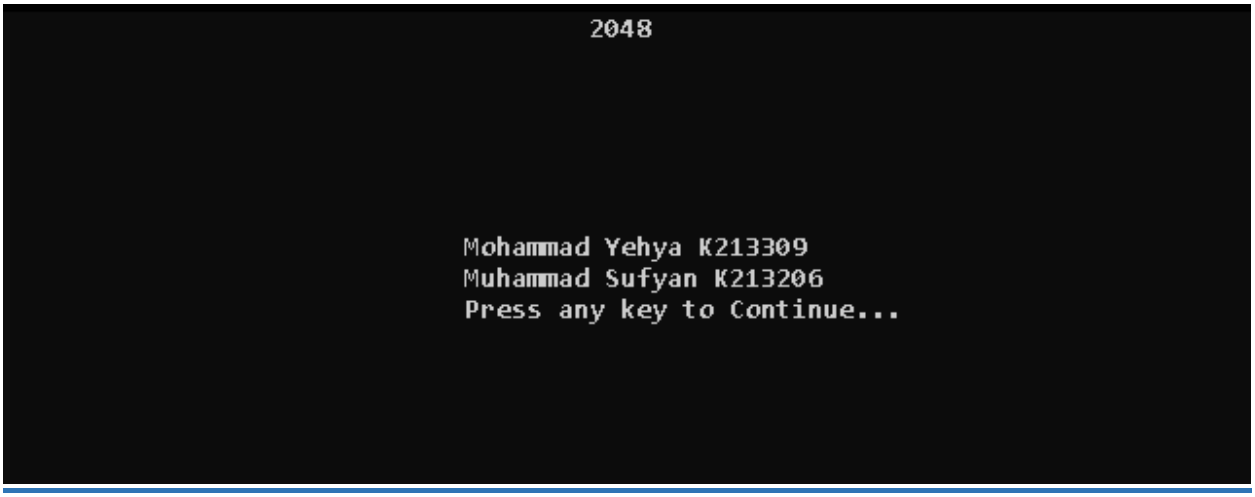
```
;-----------------------------------------------------
;A simple function that checks if there is a '2048' on the grid; Value set in EAX
GameWon PROC
;-----------------------------------------------------
    mov ecx, 0
GameWin:
    cmp ecx, 16
    jge GameNotWin
    cmp Grid[ecx * TYPE DWORD], 2048
    je Win
    inc ecx
    jmp GameWin
GameNotWin:
    mov eax, 0
    jmp GameWinRet
Win:    mov eax, 1
GameWinRet:ret
GameWon ENDP
;-----------------------------------------------------
;A funuction that checks if there are no more possible combinations; Value set in EAX
GameLost PROC
;-----------------------------------------------------
    call IsFull
    cmp eax, 0
    je LostRet
    mov ecx, 0
    mov ebx, 0
LostHorizontal:
    cmp ecx, 16
    jge LostHorizontalFin
    mov edx, ecx
    inc edx
    cmp ebx, 3
    je resetHorizontalEBX
    mov eax, Grid[ecx* TYPE DWORD]
    cmp eax, Grid[edx * TYPE DWORD]
    je NotLost
    inc ebx
contHorizontal: inc ecx
    jmp LostHorizontal
LostHorizontalFin:
    mov ecx, 0
    mov ebx, 0
```

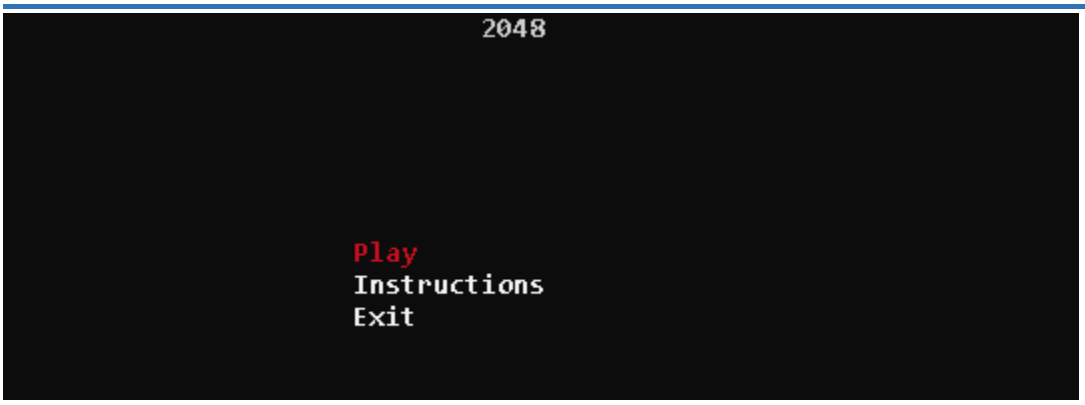
Here is an example of losing the game.

You can see that no same values are adjacent to each other either vertically or horizontally and that where the game ends because it needs same adjacent values for addition to continue the game.

As for the menus, they were created with a simple yet clever tactic of disguising a variable which is the cause for the aforementioned illusion effect. The variable's value is updated as the user presses on of the arrow keys, which then prints the corresponding text in color. This cycle repeats itself until the user presses the enter key. Then the value of variable is checked and the appropriate function is then called.

```
CreditStrings   BYTE "Mohammad Yehya K213309",0,"Muhammad Sufyan K213206",0,"Press any key to Continue..."
Instructions1 BYTE "The aim of the game is simple. You have to get the number form the number 2048.",0
Instructions2 BYTE "You can only add same numbers, for example 2 can only add with 2 and 4 can only add with 4.",0
Instructions3 BYTE "If there are no more numbers that can be added on the board then it is Game over!",0
MenuStrings BYTE "Play",0
            BYTE "Instructions",0
            BYTE "Exit",0
MenuArr DWORD 0,5,13
```





Project Breakdown Structure

We started our project on the 15th of October 2022. We decided to split the project into three phases:

1. Main Game
2. Extra Features (Menus/Instructions/Exit Program)
3. Clean Up of Program

The project was further split into two parts which consisted of the main game and other small functionalities for having a better user experience. The major part of the game was easy to implement as was done by 25th of October 2022. To implement all functionalities fulfilling the rules of the game was done around 8th of November 2022 and it was not fully functional as it had some bugs and glitches. These problems were dealt with at around 17th of November 2022. Then we arrived in the second phases which includes the quality of life functions. These are the menus and instruction manual which were carefully and delicately made in order for the user to have a smooth experience. These changes were implemented as soon as 26th of November 2022. At the end, we made some tiny changes which includes the addition of color and joining of all the functions which leads to the final product of "*2048 Re-Made*" and this was achieved on 1st of December 2022.

Results

The result is extremely satisfactory as it shows the amount of hard work and devotion put into this project when the final product comes to bloom. Unfortunately, the idea that was proposed of introducing power ups which could help the user in achieving his goal, was scrapped as it defeated the purpose and challenge of the game. We also scrapped the idea of the high score as we deemed it unnecessary and we felt like there was no need to add it as the user would only play it once, achieve his highest score and forget the rest; we do not desire the game to die out that easily, hence the removal of the high score system.

Conclusion

In conclusion we are extremely happy of the finished product and are sure that it will be evaluated and rated highly among the other projects in our batch.