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. Asad Noor Khan , 21K-4678
3. Hamza Baig , 21K-3955

Background

The main research done was on the various different library functions and how to use them in our program. The libraries we used were:

- Stdio.h
- Stdlib.h
- Conio.h
- Time.h
- Windows.h

Stdio.h was used for using simple programming elements such as scanf () and printf ().

Stdlib.h was used for exit (), system (), srand () and rand () functions.

Conio.h was used for getch () and kbhit () functions.

Time.h was used for clock () and time () functions as well as the clock_t data type.

Windows.h was used for SetConsoleTextAttribute () function as well as using HANDLE.

Help was used from https://fresh2refresh.com/c-programming/ for the multiple functions.

Logic was built on our own (except for the Time.h usage which was fairly complicated) as there does not exist any reference code that corresponds to our level, therefore the program consists of basic syntax that we learnt in our classes and labs.

Solution Design

The program consists of multiple menus/panels from which it starts from rolling the credits or in simpler terms addresses the creator and the team of the project. It is then followed by a Menu which prompts the user to pick a choice out of the 4. The user can use the arrow keys to select a different choice. This is done by using Windows.h SetConsoleTextAttribute () and system (“CLS”) functions to perform an illusion to the user as if the choices are moving. The 4 choice are Play, Instructions, Credits, and Exit. If the user picks Exit the program prompts the user if he really wants to exit. If yes then we run the exit () function if

no we can go back to the menu. If the user chooses Credits then it will reshow the credits which were shown at the beginning. If the user chose Instructions then it will display all the instructions of the different modes of the game. If the user picks Play he will be prompted to pick a difficulty. There are three difficulties to pick from which are Normal, Hard, and Expert. The Normal mode consists of the regular 2048 game. The Hard mode has all the normal functionalities of the Normal mode but there is a chance of a bomb to generate, and if two bombs join then it is game over. The Expert mode has all the functionalities of the Hard mode but there is a timer which is adjustable, however if the timer runs out it is game over. If the user wants to exit the match and return to the main menu all he has to press is the escape key and he will be prompted with a menu. This is our take on the required innovation/new features as requested by the instructor. To make it professional we use colors and clearing screens which makes the illusion of the program moving.

Implementation & Testing

First and foremost we had to make the title which was made by using ASCII Art and is simply printed by using a user-defined function title ().

Next 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 checkisfull () and checkupdate ().

```
    }
        if (checkisfull() == 0)
        {
            do
            {
                temp = rand()%16;
            }
            while (emptycheck[temp] != 1);
        }
    k = 0;
    if (checkisfull() == 0)
    {
        for (i = 0 ; i < 4 ; i++)
        {
            for (j = 0 ; j < 4 ; j++)
            {
                if (k == temp)
                {
                    board[i][j] = numbers[rand()%n];
                    emptycheck[temp] = 0;
                }
                k = k + 1;
            }
        }
    }
}
int checkisfull()
```

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 ().

```
    case 72:
        a = 0;
        for (i = 0 ; i < 4 ; i++)
        {
            for (j = 1 ; j < 4 ; j++)
            {
                a = j;
                for (k = j ; k > 0 ; k--)
                {
                    if (board[k-1][i] == 0)
                    {
                        board[k-1][i] = board[a][i];
                        board[a][i] = 0;
                        a--;
                    }
                    else if(board[k-1][i] != board[a][i])
                    {
                        break;
                    }
                    else if (board[k-1][i] == board[a][i] && (added[k-1][i] == 0 && added[a][i] == 0))
                    {
                        board[k-1][i] *= 2;
                        board[a][i] = 0;
                        added[k-1][i] = 1;
                    }
                }
            }
        }
        for (i = 0 ; i < 4 ; i++)
        {
            for (j = 0 ; j < 4 ; j++)
            {
                added[i][j] = 0;
            }
        }
        return 1;
```

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 GameOver () and GameWon ().

```
int GameOver()
{
    int i,j,flag = 1;
    for (i = 0 ; i < 4 ; i++)
    {
        for (j = 0 ; j < 3 ; j++)
        {
            if (board[i][j] == board[i][j+1])
            {
                flag = 0;
                return flag;
            }
        }
    }
    for (i = 0 ; i < 4 ; i++)
    {
        for (j = 0 ; j < 3 ; j++)
        {
            if (board[j][i] == board[j+1][i])
            {
                flag = 0;
                return flag;
            }
        }
    }
    return flag;
}
```

```
int GameWon()
{
    int i,j,flag = 0;
    for (i = 0 ; i < 4 ; i++)
    {
        for (j = 0 ; j < 4 ; j++)
        {
            if (board[i][j] == 2048)
            {
                flag = 1;
                return flag;
            }
        }
    }
    return flag;
}
int bombexplode()
```

Here is an example of losing the game.

Then for the hard mode we had to adjust the array that carries 2's and 4's and extend it to add and "X" which denotes the symbol for a BOMB. We also had to add another condition on top of the existing losing condition that if two bombs join then the game is over. This is done using a user-defined function called bombexplode ().

```
875  int bombexplode()
876  {
877      int i,j,flag=0;
878      for (i = 0 ; i < 4 ; i++)
879      {
880          for (j = 0 ; j < 4 ; j++)
881          {
882              if (board[i][j] == 176)
883              {
884                  flag = 1;
885                  return flag;
886              }
887          }
888      }
889      return flag;
890  }
```

Here is an example of two bombs joining at the top-right corner.

For the expert mode we needed to add a timer which counts for 2 seconds (is changeable) and the user will lose the game if he does not make a move in that time limit. This was done by using the functions from Time.h and some user-defined functions called delay () and displaytimer ().

```
912     }
913     void delay(int ms)
914     {
915         clock_t timeDelay = ms + clock();
916         while (timeDelay > clock());
917     }
918     void displaytimer()
919     {
920         printf("\n%s",space);
921         printf("\t\t\t\t\t\t\t\t\t\t\b\bTIMER");
922         printf("\n%s",space);
923         printf("\t\t\t\t\t\t\t\t\t\t\b\b%d : %d" , second , msecond);
924     }
925
```

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.

```
290  int menu()
291  {
292      int b = 0,i,z;
293      char menu [4][50] = {"Play" , "Instructions" , "Credits" , "Exit"};
294      do
295      {
296          title();
297          for (i = 0 ; i < 4 ; i++)
298          {
299              if (b == i)
300              {
301                  SetConsoleTextAttribute(color, 4);
302                  printf("\n%s",space);
303                  printf("\t\t\t\t\t\t\t\t\t%s\n",menu[i]);
304                  SetConsoleTextAttribute(color, 15);
305              }
306              else
307              {
308                  printf("\n%s",space);
309                  printf("\t\t\t\t\t\t\t\t\t%s\n",menu[i]);
310              }
311          }
312          z = getch();
313          if (z == 27)
314          {
315              excheck(0);
316          }
317          if (z == 224)
318          {
319              z = getch();
320              if (z == 72)
321              {
322                  if (b > 0)
323                  {
324                      b--;
325                  }
326              }
327              else if(z == 80)
328              {
329                  if (b < 3)
330                  {
331                      b++;
332                  }
333              }
334          }
335          system("CLS");
336      }
337      while (z != 13);
338      return b;
339  }
```

Project Breakdown Structure

We started our project on the 15th of October 2021. We decided to split the project into three phases:

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

The main game was further split into three parts which consisted of the three difficulty levels. The main Normal version of the game was easy to implement as was done by 25th of October 2021. However the next two difficulties were a little tough to implement due to the advanced level programming

techniques used and was done around 8th of November 2021 and was not fully functional as it had some bugs and glitches. These problems were dealt with at around 17th of November 2021. 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 29th of November 2021. Phase three includes the fixing of all 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 6th of December 2021.

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. Instead, we introduced elements which gives the users a challenge like the bomb and the timer. 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.