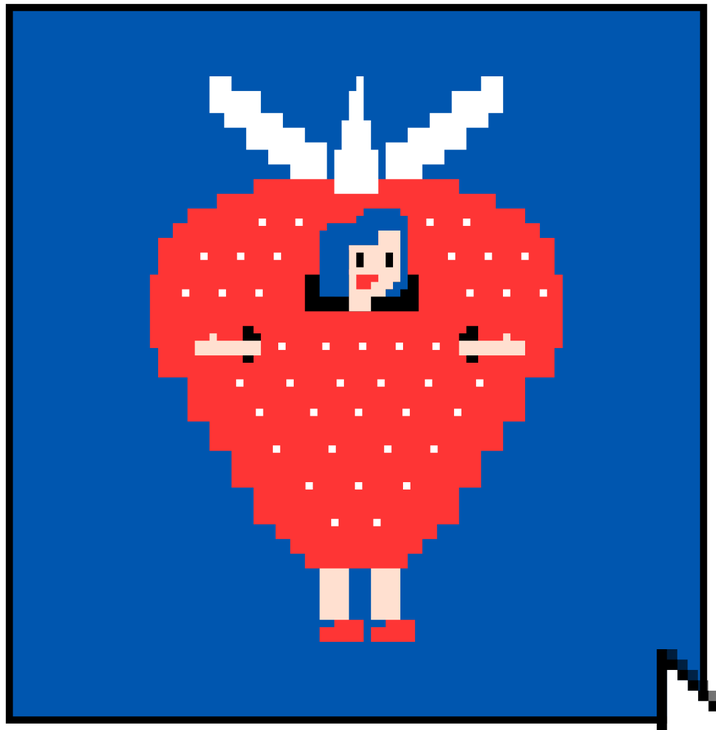
Help! I can't stop playing 2048

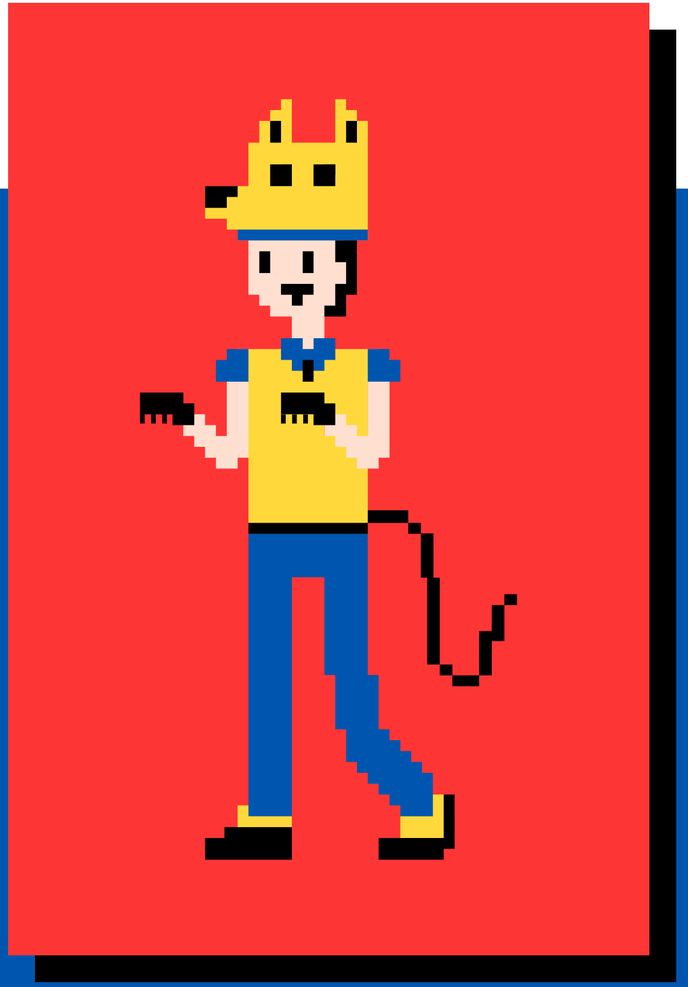


2048

EVEN IF YOU DONT LOVE NUMBERS,
YOU WILL LOVE THIS GAME.

TEAM:

- Choutapally Akshara
- Ekta Kumari
- Gaddam Swathi Reddy
- Priyankaa Sarkar
- Tejal Edulakanti



Step 1: Before You Start

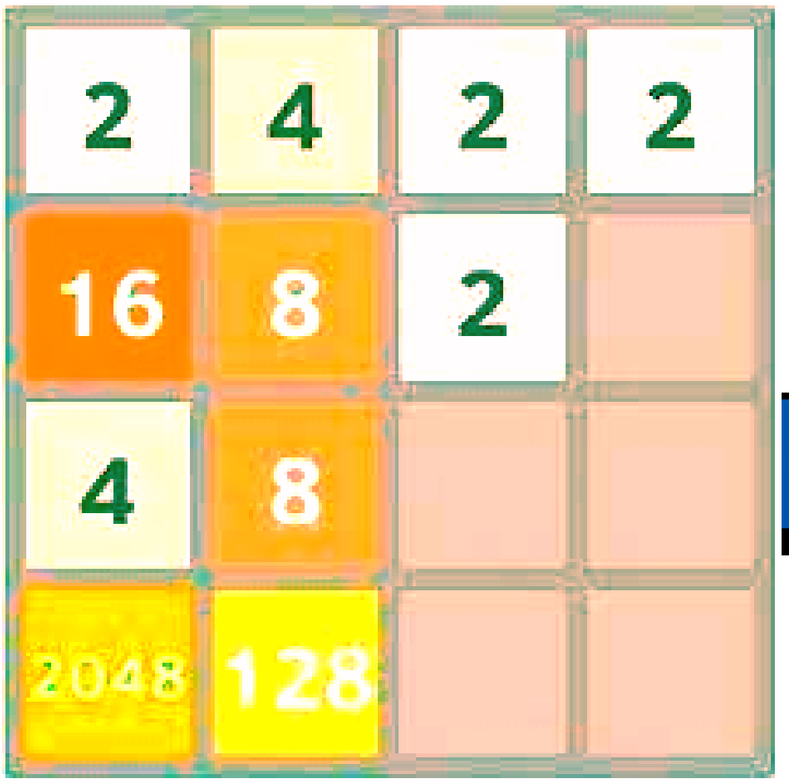
Things to note before you jump in

What is 2048?

2048 is a single-player sliding tile puzzle game played on a 4x4 game board. It is one of the most addictive mathematical puzzle games.The goal of the game is to combine equally-valued tiles to reach the 2048 tile without getting stuck.

Intro

The project we have made allows the user to access a few special features along with the basic 2048 game. Most importantly, we have added the undo function, which allows them to go back one step any time during the duration of the game. We have also added the feature of a leader-board, this enables the user to check their current scores along with their scores of preciously played games. They are given the choice of resetting this leader-board according to their preferences too



2048 video game

components
that we used

Libraries

Classes

Global Variables

Global functions

Private Variables

Public Variables

Member functions

Member functions

Step 2: Libraries Used :

<iostream>:

The iostream stands for standard input-output stream. This header file contains definitions to objects like cin, cout, cerr etc.

<ctime>:

The ctime() function in C++ converts the given time since epoch to a calendar local time and then to a character representation.

<unistd.h>:

The General Purpose Standard Library of C programming language which declares a variety of utility functions

Step 2: Libraries Used :

<cstdlib>:

Used for the General Purpose Standard Library of C programming language which declares a variety of utility functions for type conversions

<cstdio>:

The header file that contains all of the old C functions to print stuff and write to files

<cmath>:

The C++ <cmath> header file declares a set of functions to perform mathematical operations.

<conio.h>:

Presentations are communication tools that can be used as lectures, speeches, reports, and more.

Step 3: Classes Used :

Derived class inherits a base class publicly

01 Game_AI :

The derived class with access specifier as the public.

02 Game :

This is the derived class with access specifier as the public.

.

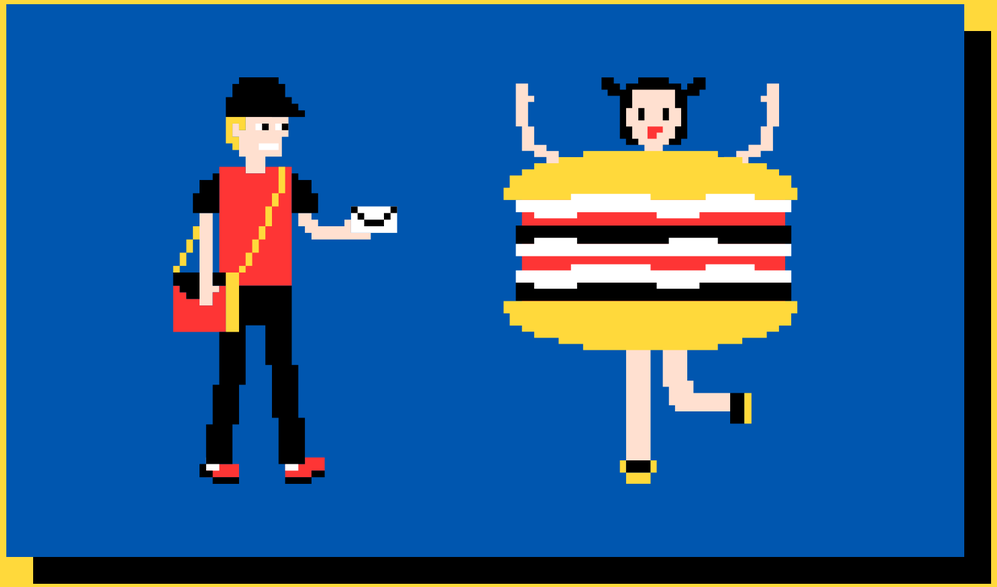
Step 4: Global variables :

```
int press_enter
```

Step 5: Global functions :

```
int random_index(int)
```

Step 6 :Private Variables :



```
int response
int apocalypse
char control
char option
char name[25]
```

Step 7:
Public Variables :

Accessed from everywhere

01 `int max;`

The derived class with access specifier as the public.

02 `int win;`

This is the derived class with access specifier as the public.
.

03 `int plus;`

The derived class with access specifier as the public.

04 `int score;`

The derived class with access specifier as the public.

Step 7:
Public Variables :

Accessed from everywhere

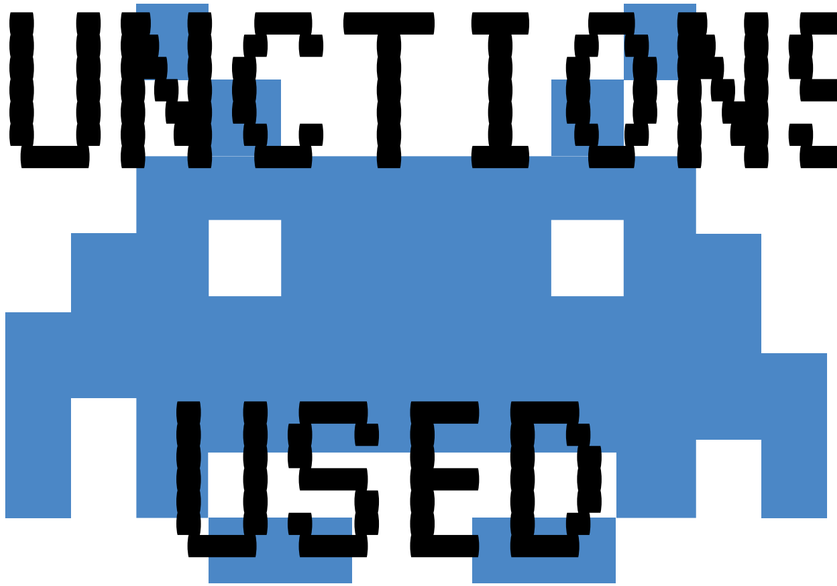
05 `int grid[4][4];`

The grid to store the current state of the game

06 `int bgrid[4][4];`

The backup grid ,for the undo functionality and to check if the grid moves.

FUNCTIONS USED



void logic_flow(Game *execute) :

```
he fuctioning of the game is carried out with the help of this function
d Game_AI::logic_flow(Game *execute)


 switch (control)
 {
 case 'w':
 case 'W':
 case 'a':
 case 'A':
 case 's':
 case 'S':
 case 'd':
 case 'D':
     execute->backup_grid();
     execute->fill_space();
     execute->update_grid();
     execute->fill_space();
     execute->find_greatest_tile();
     execute->display_grid();
     usleep(1000 * 160);

     if (execute->full() && apocalypse)
     {
         response = -1;
         break;
     }
     else if (execute->block_moves())
```

The controls of this grid i.e, up, down, left, right as w, s, a, d are accepted in this function and the functions to be further carried out after are called too

```
    else if (execute->block_moves())
    {
        execute->spawn();
        break;
    }
    else
    {
        response = 0;
        break;
    }

case 'u':
case 'U':
    if (execute->block_moves())
        score -= plus;
    execute->undo();
    break;

case 'r':
case 'R':
    execute->start_grid();
    score = 0;
    plus = 0;
    break;

case 'q':
case 'Q':
    response = -1;
    break;
```

```
    case 'h':
    case 'H':
        execute->display_help_screen();
        press_enter = getchar();
        execute->display_grid();
        break;
    }
}
```

The options of undoing your move, restarting your game, quitting your game and looking at the rules/help are also provided through this function

void game_end_check(Game*)

```
d Game_AI::game_end_check(Game *screen)
 if (max == win)
 {
     screen->display_win_screen();

     if (screen->display_try_again_screen(0) == 'n' || screen->display_try_again_screen(0) == 'N')
     {
         response = -1;
     }
     else
         win *= 2;
 }
 else if (response == -1)
 {
     screen->display_loser_screen();

     if (screen->display_try_again_screen(1) == 'y' || screen->display_try_again_screen(1) == 'Y')
     {
         screen->start_grid();
         response = 0;
     }
 }
 if (response == -1)
 {  system("cls");
     cout << "\n\n\t\t             > Thank you for playing. "
          << "\n\n\n\t\t\t\t\t\t\t    [-_-] \n\n";
     press_enter = getchar();
     exit(0);
 }
```

What is displayed on the screen after the completion of the game is determined through this function

void key_press()

```
void Game_AI::key_press()
{
    system("stty raw");
    cin >> control;
    system("stty cooked");
}
```

This function allows us to take in a character that will be used in various switch-case conditions later on

void display_first_menu(Game*)

```
void Game_AI::first_menu(Game *execute)
{

    switch (control)
    {

    case 's':
    case 'S':
        system("cls");
        cout << "\n\n\n\n\n\n\t\t\t\tPlease enter your name \n\t\t\t\t\t";
        cin >> name;
        execute->start_grid();
        while (1)
        {
            execute->display_grid();
            execute->key_press();
            execute->logic_flow(execute);
            execute->game_end_check(execute);
        };
        break;

    case 'q':
    case 'Q':
        cout << "Thank you for playing!" << endl
             << endl
             << endl;
        response = -1;
        break;
```

The main menu and its basic actions are displayed and carried out through this function, start, quit and help which displays the rules are the actions which will be shown.

```
    case 'h':
    case 'H':
        execute->display_help_screen();
        press_enter = getchar();
        system("cls");
        cout << "\n\n\n\n\n\n\t\t\t\tPlease enter your name \n\t\t\t\t\t\t";
        cin >> name;
        start_grid();
        execute->start_grid();
        while (1)
        {
            execute->display_grid();
            execute->key_press();
            execute->logic_flow(execute);
            execute->game_end_check(execute);
        }
        break;
    }
}
```

The two digits are either 2 or 4 and they appear at random indexes

void start_grid()

```
void Game_AI::start_grid()
{
    int i, j;

    plus = 0;
    score = 0;
    max = 0;

    for (i = 0; i < 4; i++)
        for (j = 0; j < 4; j++)
            grid[i][j] = 0;

    i = random_index(4);
    j = random_index(4);

    grid[i][j] = 2;

    i = random_index(4);
    j = random_index(4);

    grid[i][j] = 2;
}
```

Through this function the initial grid is generated

The initial grid is a 4*4 grid with two of its places containing digits and the rest are empty spaces

void update_grid()

```
void Game_AI::update_grid()
{
    plus = 0;
    apocalypse = 1;

    switch (control)
    {
    case 'w':
    case 'W':
        for (int i = 0; i < 4; i++)
            for (int j = 0; j < 3; j++)
            {
                if (grid[j][i] && grid[j][i] == grid[j + 1][i])
                {
                    apocalypse = 0;
                    grid[j][i] += grid[j + 1][i];
                    grid[j + 1][i] = 0;
                    plus += (((log2(grid[j][i])) - 1) * grid[j][i]);
                    score += (((log2(grid[j][i])) - 1) * grid[j][i]);
                }
            }
        break;
```

With the help of this function we update the places of the grid after every move the user makes, we increase the value of an index by double whenever two like blocks collide with one another and also add up the score and

```
case 's':
case 'S':
    for (int i = 0; i < 4; i++)
        for (int j = 3; j > 0; j--)
        {
            if (grid[j][i] && grid[j][i] == grid[j - 1][i])
            {
                apocalypse = 0;
                grid[j][i] += grid[j - 1][i];
                grid[j - 1][i] = 0;
                plus += (((log2(grid[j][i])) - 1) * grid[j][i]);
                score += (((log2(grid[j][i])) - 1) * grid[j][i]);
            }
        }
    break;
case 'a':
case 'A':
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 3; j++)
        {
            if (grid[i][j] && grid[i][j] == grid[i][j + 1])
            {
                apocalypse = 0;
                grid[i][j] += grid[i][j + 1];
                grid[i][j + 1] = 0;
                plus += ((log2(grid[i][j])) - 1) * grid[i][j];
                score += ((log2(grid[i][j])) - 1) * grid[i][j];
            }
        }
    break;
```

```
    case 'd':
    case 'D':
        for (int i = 0; i < 4; i++)
            for (int j = 3; j > 0; j--)
            {
                if (grid[i][j] && grid[i][j] == grid[i][j - 1])
                {
                    apocalypse = 0;
                    grid[i][j] += grid[i][j - 1];
                    grid[i][j - 1] = 0;
                    plus += ((log2(grid[i][j])) - 1) * grid[i][j];
                    score += (((log2(grid[i][j])) - 1) * grid[i][j]);
                }
            }
        break;
    }
}
```

how much the score increases by with each move and display it with the help of this function

void fill_space()

```
void Game_AI::fill_space()
{

    switch (control)
    {
    case 'w':
    case 'W':
        for (int i = 0; i < 4; i++)
            for (int j = 0; j < 4; j++)
            {
                if (!grid[j][i])
                {
                    for (int k = j + 1; k < 4; k++)
                        if (grid[k][i])
                        {
                            grid[j][i] = grid[k][i];
                            grid[k][i] = 0;
                            break;
                        }
                }
            }
        break;

    case 's':
    case 'S':
        for (int i = 0; i < 4; i++)
            for (int j = 3; j >= 0; j--)
            {
```

This function adds space in all the suitable indexes of the grid

```
case 's':
case 'S':
    for (int i = 0; i < 4; i++)
        for (int j = 3; j >= 0; j--)
        {
            if (!grid[j][i])
            { for (int k = j - 1; k >= 0; k--)
                    if (grid[k][i])
                    { grid[j][i] = grid[k][i];
                        grid[k][i] = 0;
                        break;
                    }
            }
        }
    break;
case 'a':
case 'A':
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
        {if (!grid[i][j])
            {
                for (int k = j + 1; k < 4; k++)
                    if (grid[i][k])
                    { grid[i][j] = grid[i][k];
                        grid[i][k] = 0;
                        break;
                    }
            }
        }
    break;
```

```
    case 'd':
    case 'D':
        for (int i = 0; i < 4; i++)
            for (int j = 3; j >= 0; j--)
            {
                if (!grid[i][j])
                {
                    for (int k = j - 1; k >= 0; k--)
                        if (grid[i][k])
                        {
                            grid[i][j] = grid[i][k];
                            grid[i][k] = 0;
                            break;
                        }
                }
            }
        break;
    }
}
```

void spawn()

```
void Game_AI::spawn()
{
    int i, j, k;

    do
    {
        i = random_index(4);
        j = random_index(4);
        k = random_index(10);

    } while (grid[i][j]);

    if (k < 2)
        grid[i][j] = 4;

    else
        grid[i][j] = 2;
}
```

This function allows us to fill up any random space on the grid with either a 2 or a 4 after every move

void find_greatest_tile()

```
void Game_AI::find_greatest_tile()
{
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            if (grid[i][j] > max)
                max = grid[i][j];
}
```

By this function we check the greatest tile present on the grid after every move and if 2048 is present then we declare the user as a winner

void backup_grid():

```
void Game_AI::backup_grid()
{
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            bgrid[i][j] = grid[i][j];
}
```

This function enables us to carry out the undo function, allows us to store the current grid in a variable array called bgrid

void undo():

```
void Game_AI::undo()
{
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            grid[i][j] = bgrid[i][j];
}
```

The undo function allows us to obtain the grid before our previous move.

int full()

```
int Game_AI::full()
{
    int k = 1;

    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
        {
            if (!grid[i][j])
                k = 0;
        }
    return k;
}
```

We check if the grid is full through this function, also meaning if there is no like tiles to further collide with one another, with the help of the variable k

int block_moves()

```
int Game_AI::block_moves()
{
    int k = 0;

    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            if (bgrid[i][j] != grid[i][j])
            {
                k = 1;
                break;
            }
    return k;
}
```

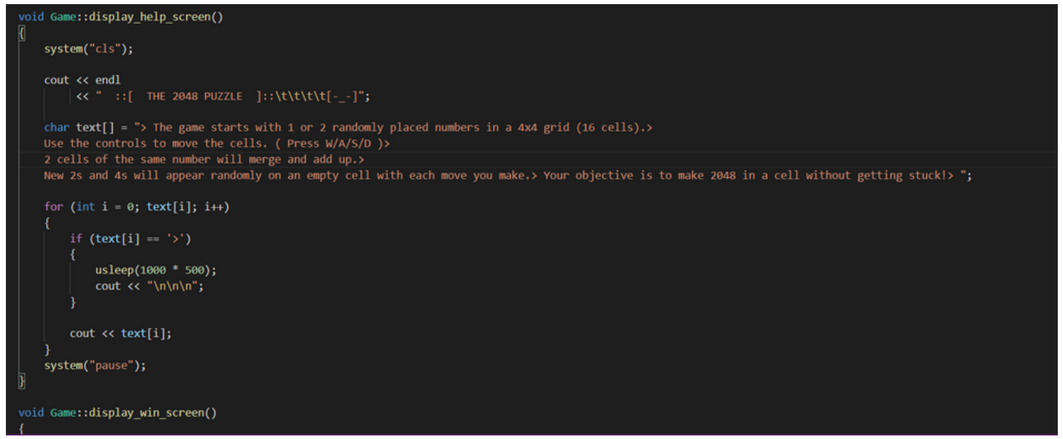
Function to check whether undo is possible or not

void display_grid() :

```
void Game::display_grid()
{

    system("cls");

    cout << "\n  ::[  THE 2048 PUZZLE  ]::\t\t\t\t [-_-]\n\n\t";

    if (plus)
        cout << "+" << plus << "!";
    else
        cout << "   ";

    cout << "\t\t\t\t\t\tSCORE::" << score << " \n\n\n\n";

    for (int i = 0; i < 4; i++)
    {
        cout << "\t\t    |";

        for (int j = 0; j < 4; j++)
        {
            if (grid[i][j])
                printf("%4d    |", grid[i][j]);
            else
                printf("%4c    |", ' ');
        }

        cout << endl
             << endl;
    }

    cout << "\n\n Controls (+ :: o)\t\t\t\tu - undo\tr - restart\n\n\tW\t\t    ^\t\t\th - help\tq - quit\n\t\t\t\t\t\t\t\t"
         << " \n   A    S    D\t\t<    v    >\t\t\t    ."
         << " \n\t\t\t\t                              ";
}
```

This displays the main game interface to the user, the On this interface the controls are specified, W to move the contents of the grid upwards, A to move them leftwards, S to move them downwards and D to move them rightwards.
The various other options of the game the user can access are U for undoing a move, R for restarting the game, H enables the display of the help screen with the rules of the game and Q allows the user to quit the game.

void display_help_screen() :



```
void Game::display_help_screen()
{
    system("cls");

    cout << endl
         << "  ::[  THE 2048 PUZZLE  ]::\t\t\t\t[-_-]";

    char text[] = "> The game starts with 1 or 2 randomly placed numbers in a 4x4 grid (16 cells).>
    Use the controls to move the cells. ( Press W/A/S/D )>
    2 cells of the same number will merge and add up.>
    New 2s and 4s will appear randomly on an empty cell with each move you make.> Your objective is to make 2048 in a cell without getting stuck!> ";

    for (int i = 0; text[i]; i++)
    {
        if (text[i] == '>')
        {
            usleep(1000 * 500);
            cout << "\n\n\n";
        }

        cout << text[i];
    }
    system("pause");
}

void Game::display_win_screen()
{
```

This function allows us to display the rules of the game whenever the user seeks for help

void display_win_screen()

```
void Game::display_win_screen()
{
    system("cls");

    cout << endl
         << endl;
    cout << "\n\t\t\t  :: [  YOU MADE " << win << "!  ] ::"
         << "\n\n\t\t\t  :: [ YOU WON THE GAME ] ::"
         << "\n\n\n\n\t\t\t TILE\t     SCORE";
    printf("\n\n\t\t\t %4d\t    %6d\t    ", max, score);

    cout << "\n\n\t\t> The maximum possible tile is 65,536 ! So go on :)";
    register_score();
}
```

Function which enables us to display what the winner sees on winning the game, reaching the 2048 tile

void display_loser_screen()

```
void Game::display_loser_screen()
{
    system("cls");

    cout << "\n\n\n\t\t\t  :: [ G A M E  O V E R ] ::"
         << "\n\n\n\n\t\t\t TILE\t     SCORE\t  ";
    printf("\n\n\t\t\t %4d\t    %6d\t    ", max, score);

    cout << "\n\n\t\t> The maximum possible score is 3,932,156 ! So close :p";

    register_score();
}
```

Function which enables us to display what the winner sees on not being able to reach the 2048 tile, or on choosing to quit the game after the generation of the grid

char display_try_again_screen(int) ...

```
char Game::display_try_again_screen(int lose)
{
    if (lose)
        cout << "\n\n\n\t    > Would you like to try again " << name << " (y/n) ? :: ";
    else
        cout << "\n\n\n\t    > Would you like to continue playing " << name << " (y/n) ? :: ";

    cin >> option;

    return option;
}
```

This function is displayed at various stages, it allows the user to choose whether he wishes to quit the game or take a shot at playing it again.

void register_score()

```
void Game ::register_score()
{
    FILE *ptr;
    ptr = fopen("record_round.txt", "a");
    fprintf(ptr, "\n%s --> %d", name, score);
    fclose(ptr);
    cout << "\n\n\t\tTo see the score press Z\n\t\tPress T to restart\n\t\tPress any key to exit\n\t\t";
    cin >> option;
        if (option == 'Z' || option == 'z'){
            show_score();
        }
        else if (option == 'T' || option == 't'){
            start_grid();
            score = 0;
            plus = 0;
        }
        else
            exit(0);
}
```

Function that enables us to store the name and score of the user in a file after each turn of theirs

void reset_score()

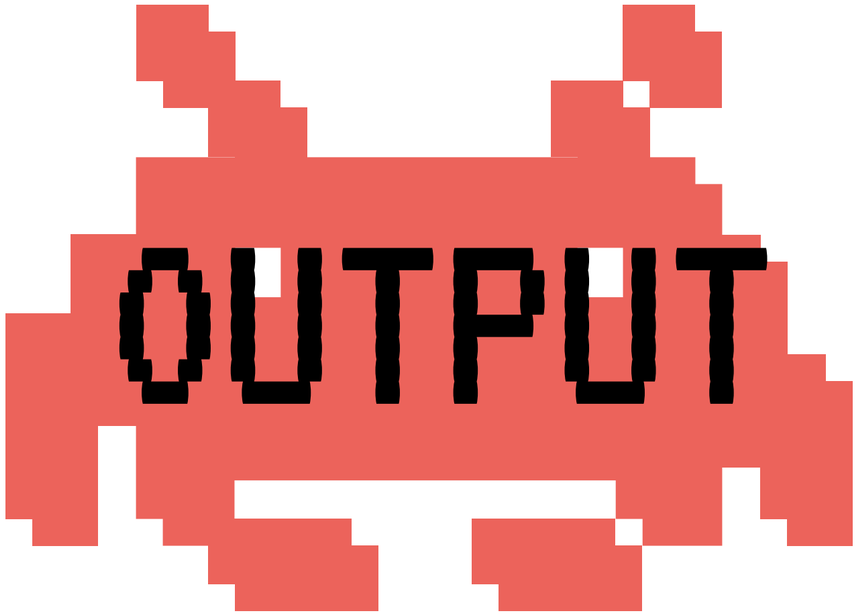
```
void Game ::reset_score()
{
    system("cls");
    FILE *fptr;
    fptr = fopen("record_round.txt", "w");
    cout << "\n\n\t\tPress Y to proceed \n\t\tPress any key to exit\n\n\t\t";
        char res = toupper(getch());
        if (res == 'Y' || res == 'y' )
        {
            remove("record_score.txt");
            printf("\n\t\tSuccesfully cleared the record\n");
            return;
        }
        else
        {
            exit(0);
        }
        fclose(fptr);
}
```

Function that enables the resetting of the said scores

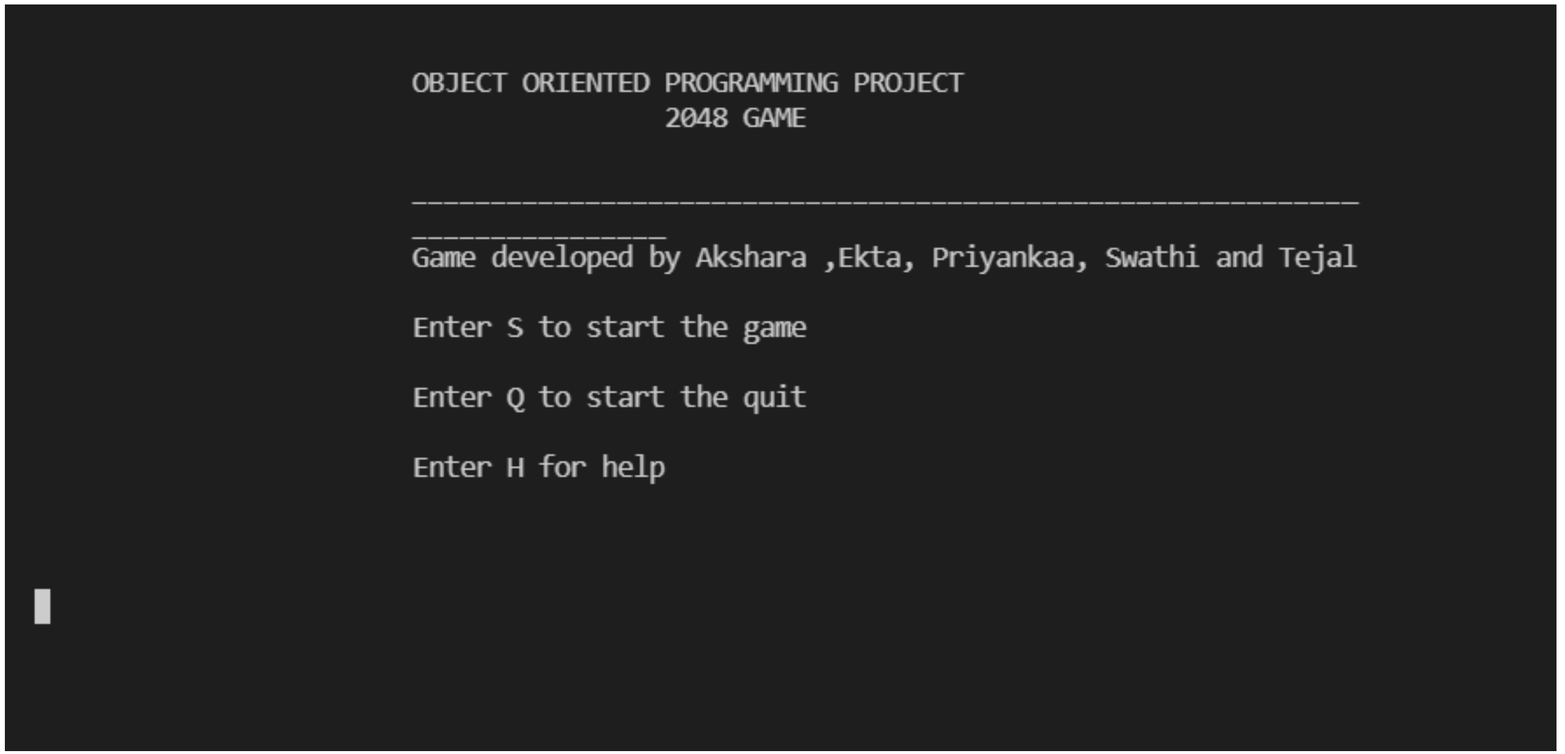
void show_score() :

```
void Game ::show_score()
{
    system("cls");
    FILE *fptr;
    fptr = fopen("record_round.txt", "r");
    if (fptr == NULL)
    {
        printf("No file exist,probably you haven't played yet\n");
        printf("Press Y to go to main home\n");
    }
    else
    {
        char c = getc(fptr);
        printf("your  : \n");
        printf("Name\t\tscore\n");
        while (c != EOF)
        {
            printf("%c", c);
            c = getc(fptr);
        }
        cout << "\n\n\t\tTo reset the score press R\n\t\tPress T to restart\n\t\tPress any key to exit\n\t\t";
        cin >> option;
        if (option == 'R' || option == 'r')
        {
            reset_score();
        }
        else if (option == 'T' || option == 't')
        {
            start_grid();
            score = 0;
            plus = 0;
        }
        else
        {
            exit(0);
        }
    }
    fclose(fptr);
}
```

Function that enables the showing of these scores

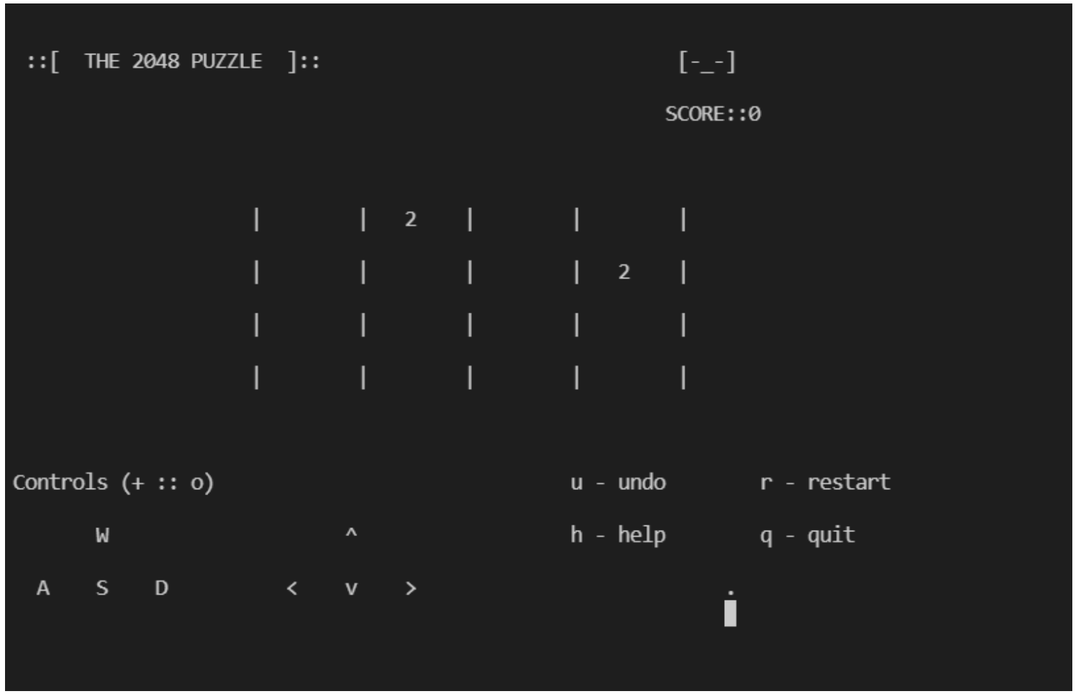

OUTPUT

This is the first page that will come onto the user's screen upon launching the game.This page provides the user the option to start the game by pressing S, to quit the game by pressing Q and take a look at the instructions of the game by pressing H.

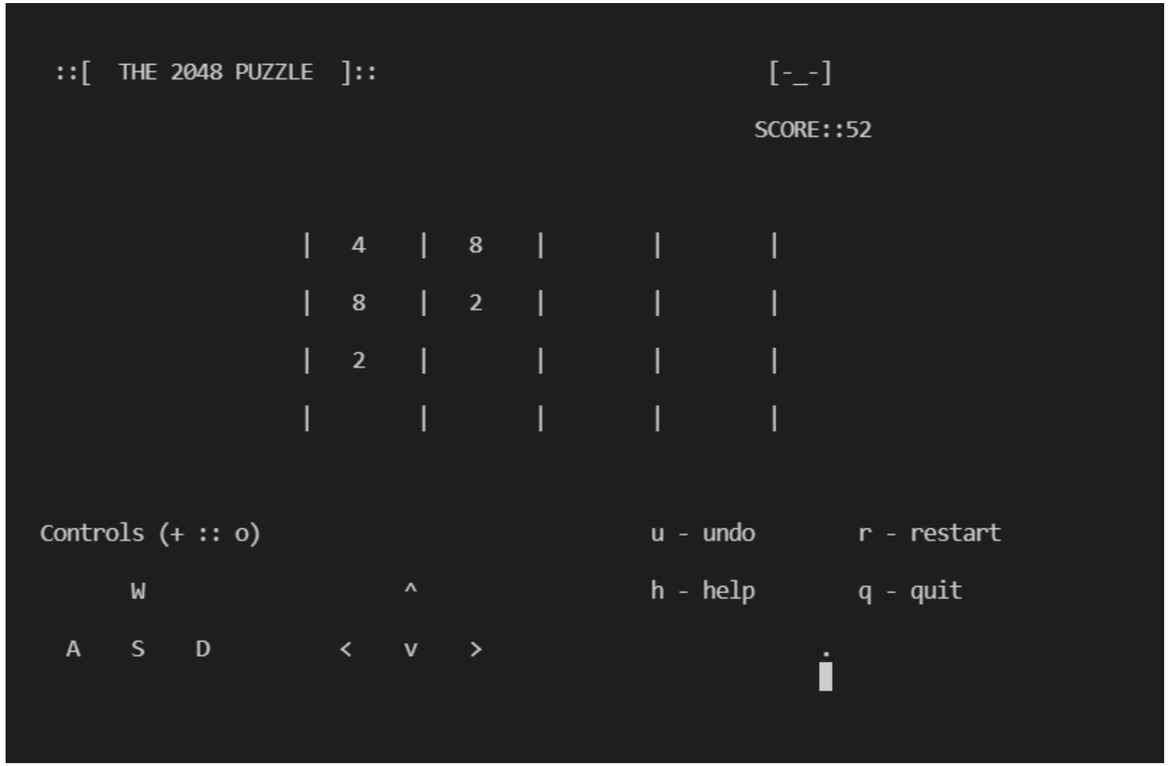


Upon the user pressing S the below screen is displayed in which they are prompted to enter the name they want to continue forward with during the duration of the game.The same name will be displayed on the leader-board.

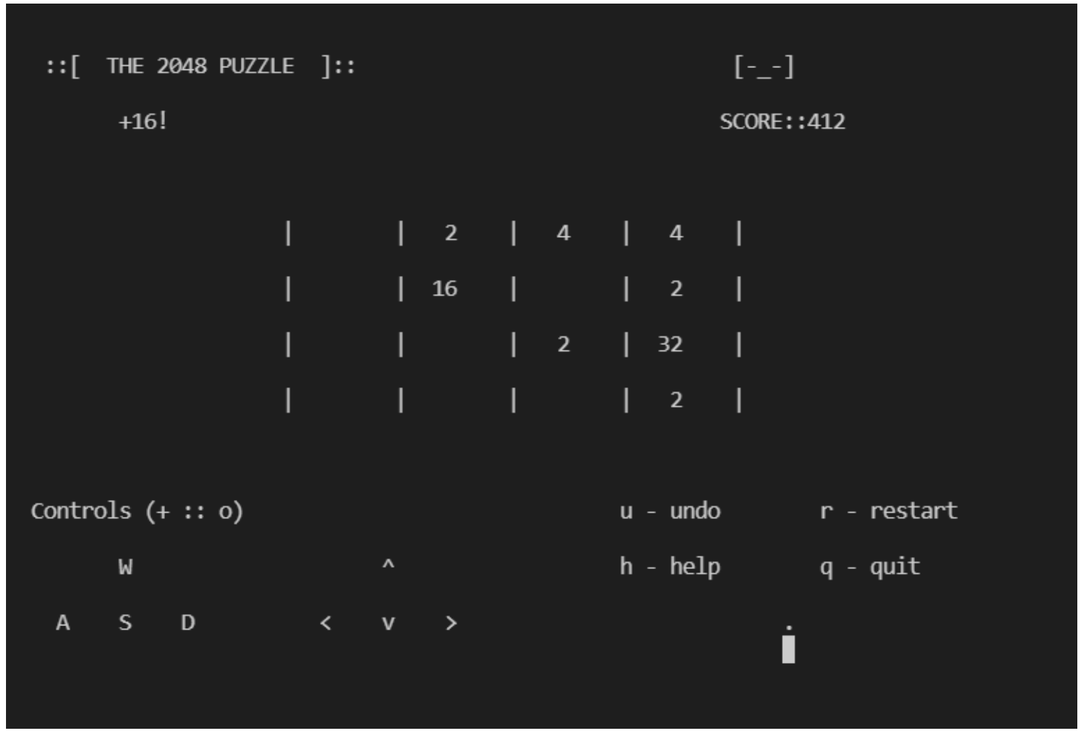
After entering your name the main game interface will be displayed on the user's screen as shown in the figure.On this interface the controls are specified, W to move the contents of the grid upwards, A to move them leftwards, S to move them downwards and D to move them rightwards.The various other options of the game the user can access are U for undoing a move, R for restarting the game, H enables the display of the help screen with the rules of the game and Q allows the user to quit the game.



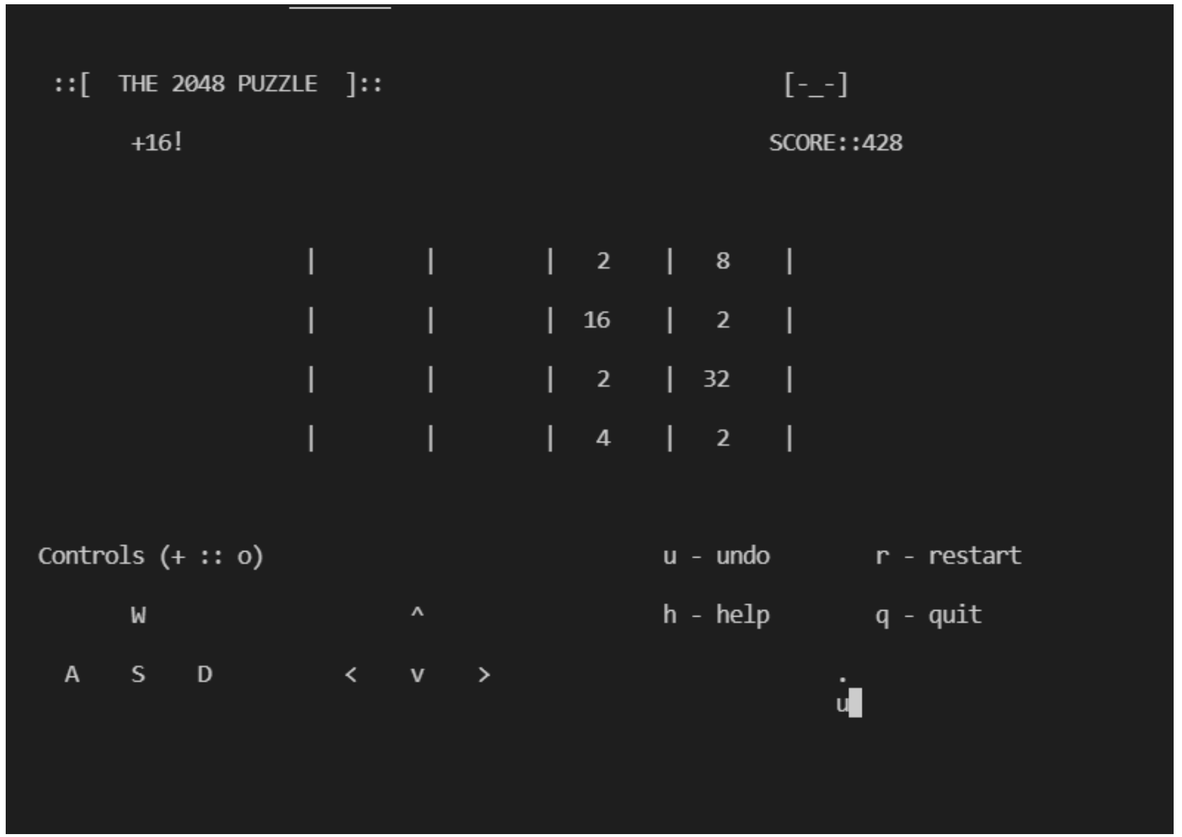
After every move by the user, two like tiles on the board that collide with one another combine and become a single tile double their previous value.For example: two tile of value 2 and 2 collide and become a single 4 tile



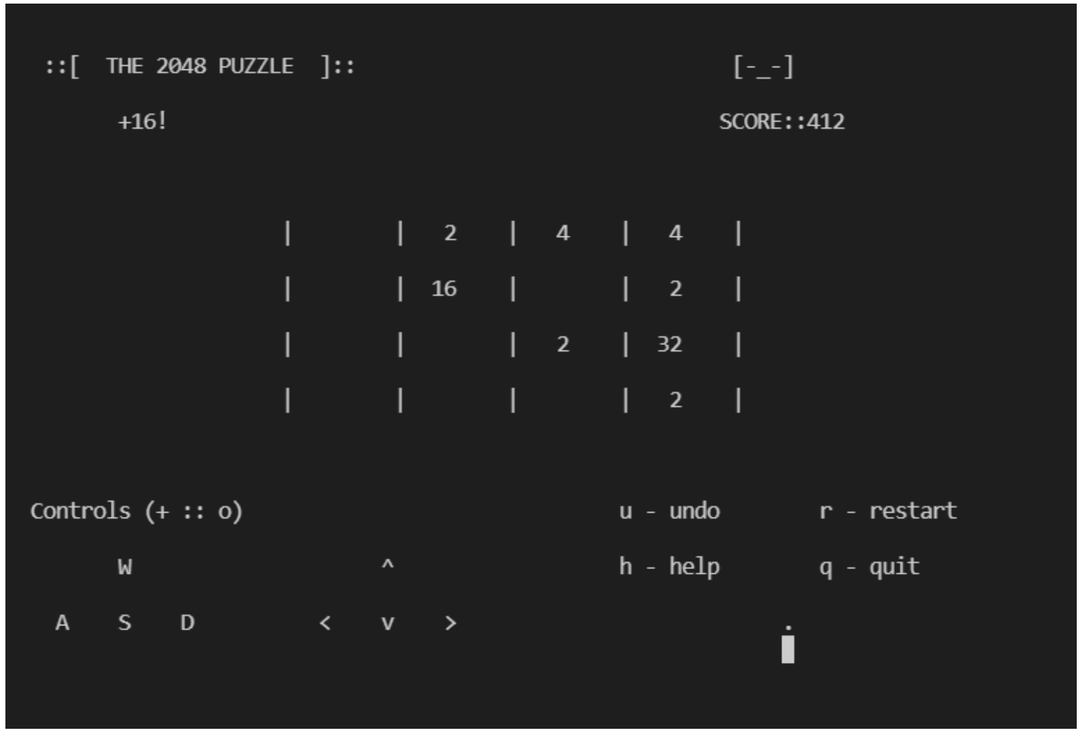
After every move played by the user the increase in the score after that particular move will be displayed on the left of the screen.For example: combining two 8s will add an increase of 16 to the score and it will be shown on the gaming interface as displayed in the picture below.The increased total score after every move is displayed on the right of the screen



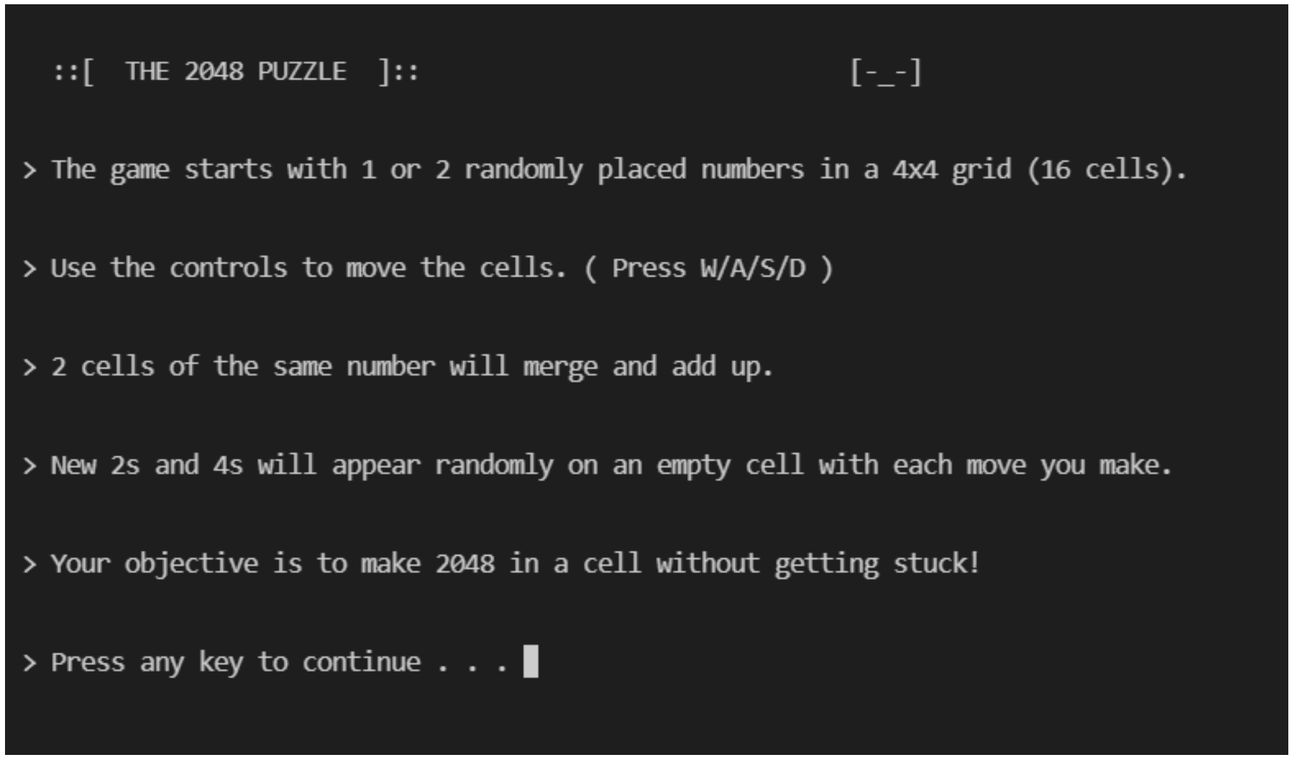
Now let us see what happens on entering u/U hence enabling the undo function.The screen in below image only differs from the previous image by one D/rightwards move



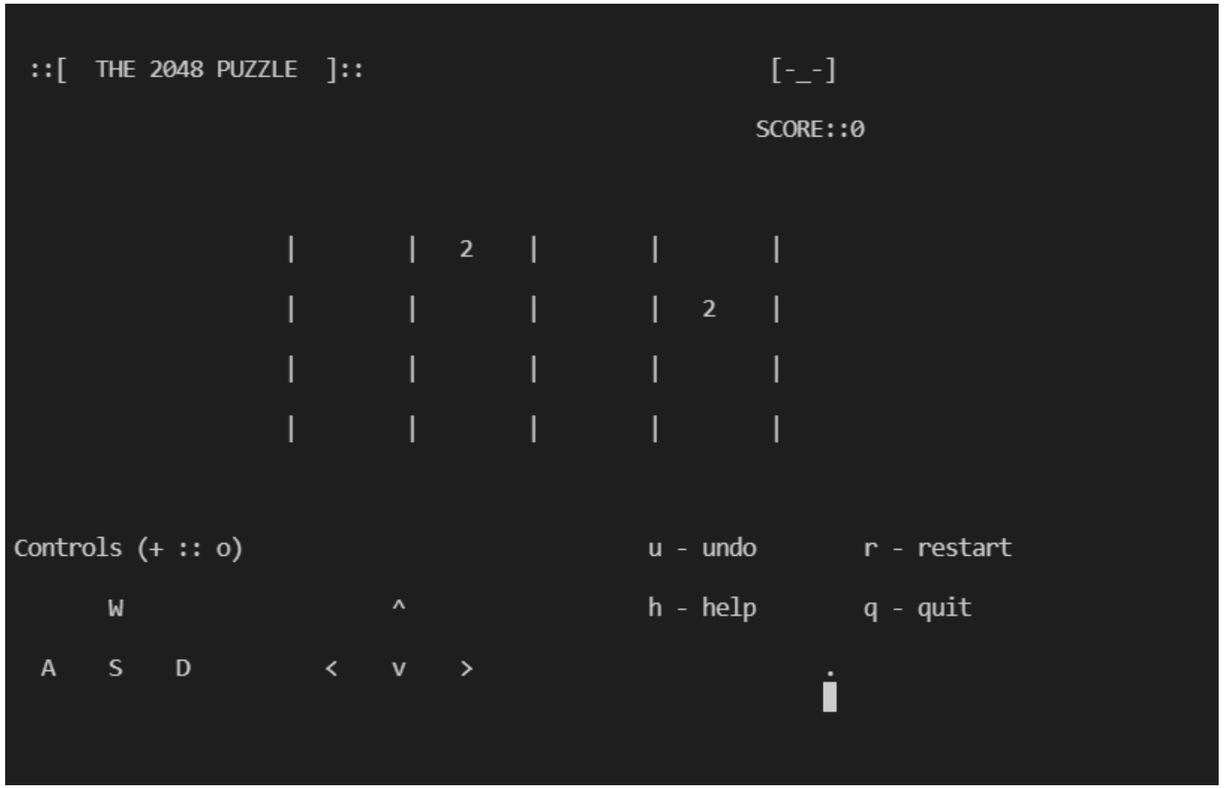
Hence the undo function successfully allows us to obtain the gird in its previous state which differs from the current by any one move.



Now let us see what is observed on pressing the h/H key.The following set of rules begin to appear on the user's screen one after the other.This allows the user to read back on the rules whenever any confusion or doubt arises in their minds at any moment while playing the game.After the help screen is displayed, the user can press any key to be redirected back to their game screen and in the same state the left it in.



On pressing the r/R key at any time during the game, we are brought back to a fresh grid with the score being reset to 0.



The following screen is displayed in two instances

1) When the user loses the game: that is when the grid becomes full and he can no longer move about the tiles in his grid to collide them

2) When the user presses the q/Q key at any moment of the game

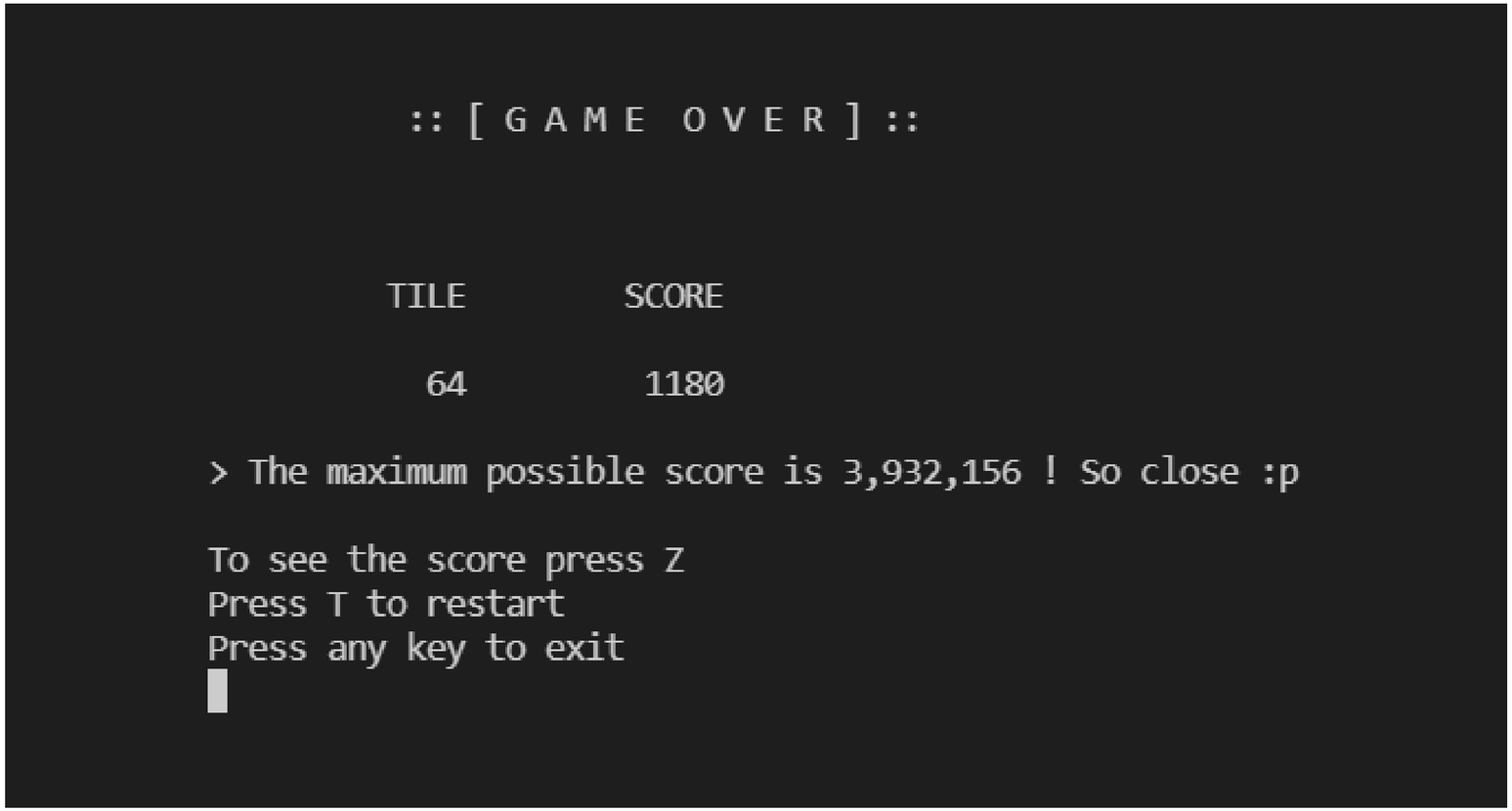
On this page the user's max tile is displayed and their overall score

The options further provided are

1) Z/z to see the leader-board, which will include the scores of all the previously played games along with this game

2) T/t allows the user to begin the game once again, it brings them to the main game interface

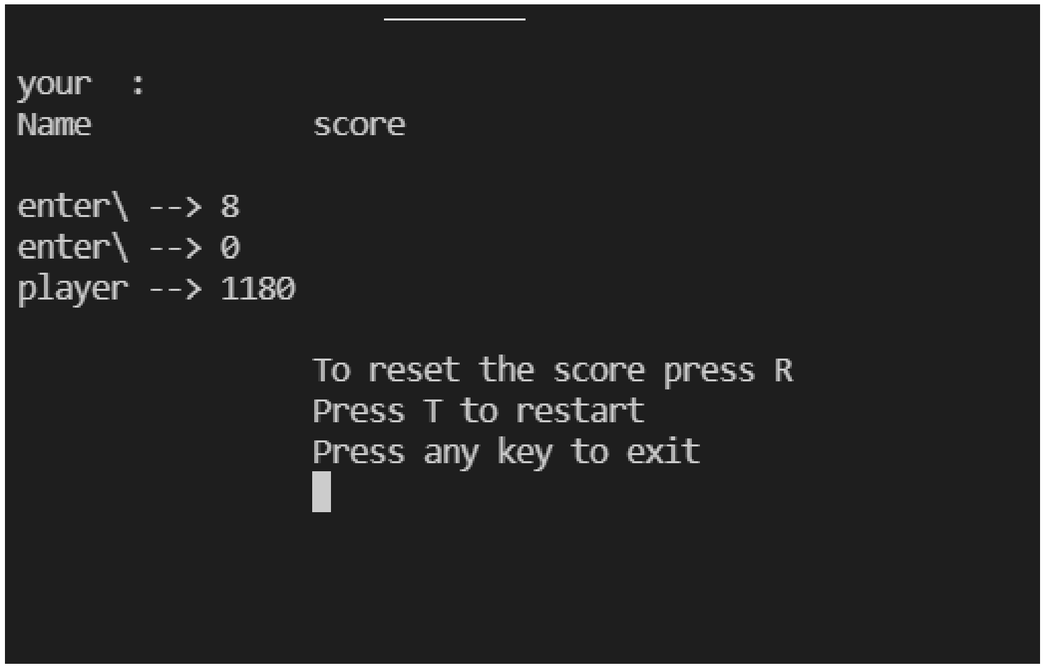
3) Pressing any other key will immediately cause the user to exit the game


:: [G A M E O V E R] ::
TILE
SCORE
64
1180
> The maximum possible score is 3,932,156 ! So close :p
To see the score press Z
Press T to restart
Press any key to exit

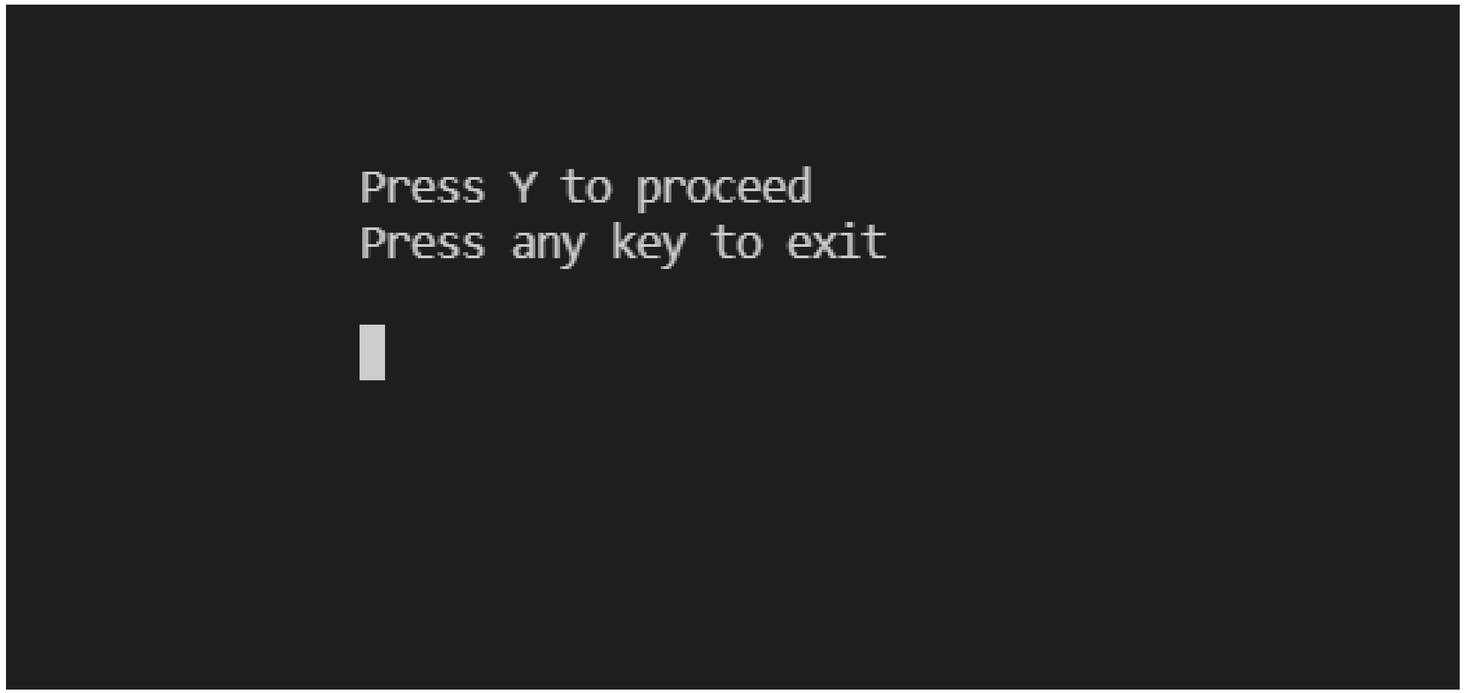
In this way the leader-board is displayed, in the leader-board the name of the player along with their score is shown as displayed in the picture.The following options are provided to the user: 1)R/r to reset the records

2) T/t allows the user to begin the game once again, it brings them to the main game interface

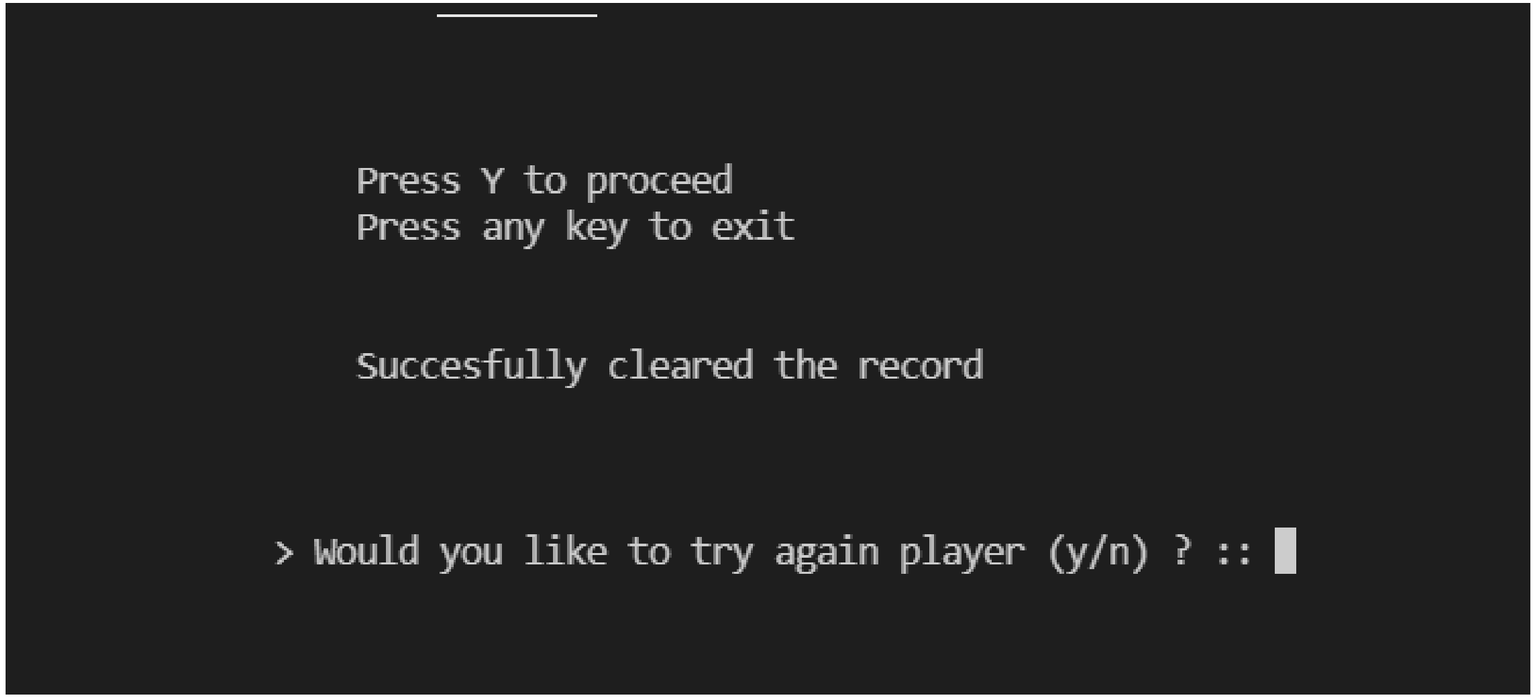
3) Pressing any other key will immediately cause the user to exit the game



On clicking the r/R key in the above menu case the following screen is displayed where the user is asked again if they want to reset the scores of the game or just exit the game completely.

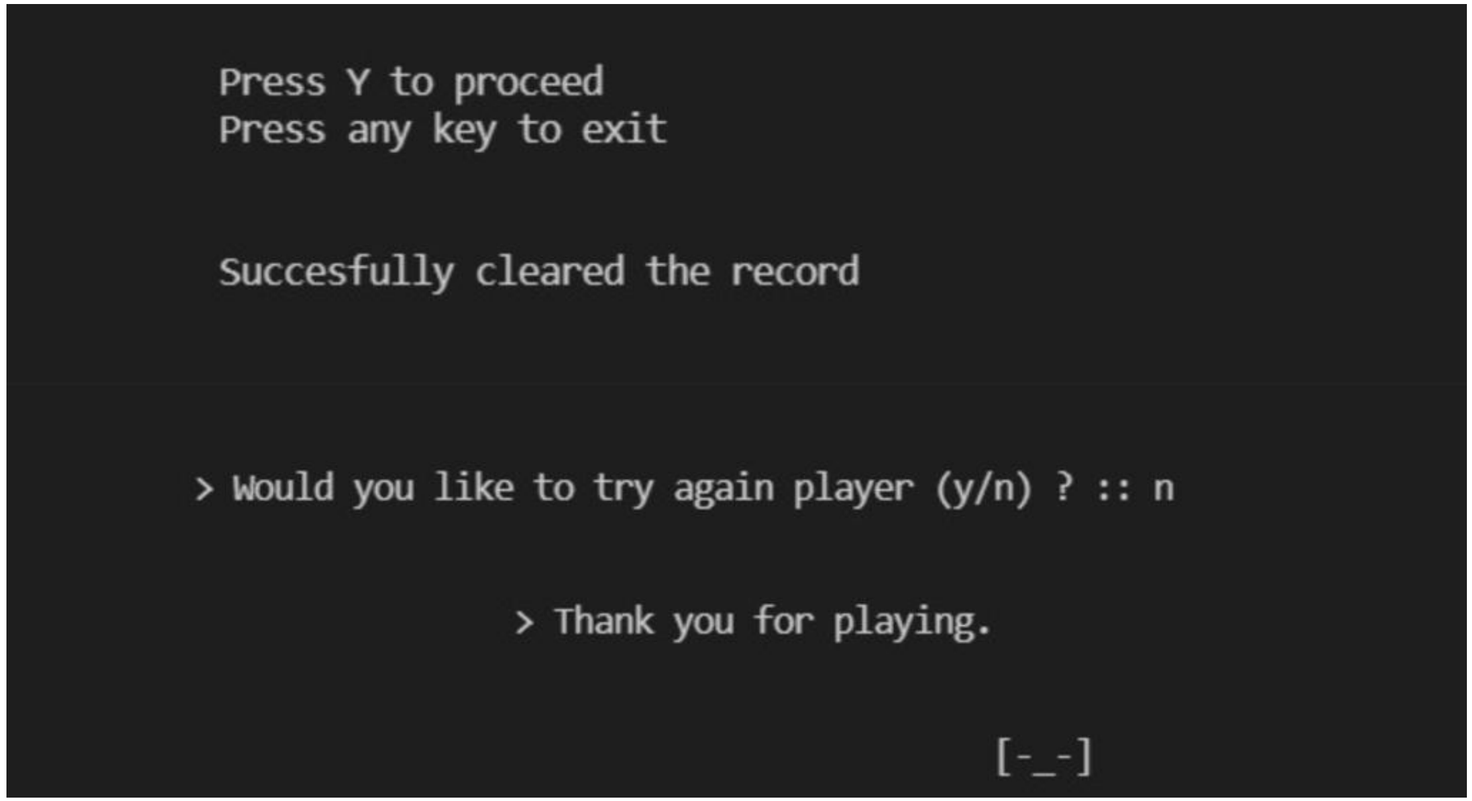


On pressing y/Y above the leader-board's scores are reset and erased.After which the user is given the option to choose whether to play again or exit the game.



On pressing y/Y the user is once again brought to the main game interface and on entering n/N the user exits the game.

The following thank you message is displayed on pressing n/N



THANK YOU!

I hope you learned something new!