|  |  |
| --- | --- |
| Onet connect  CW2-Group-45 | Abstract  Onet Connect is a puzzle game where we have to match similar tiles to eliminate them, the matching tiles should be in a reachable path between them.  Advance Web Technology |

# Table of Contents

|  |  |  |
| --- | --- | --- |
| INDEX | CONTENT | PAGE |
| 1 | Introduction | 3 |
| 1.1 | .NET Framework | 3 |
| 1.2 | Pros of .NET Framework. | 3 |
| 1.3 | Cons of .NET Framework. | 3 |
| 2 | What is an Onet Connect game? | 3 |
| 3 | Design of the Playing board. | 3 |
| 4 | Placing solvable number of colours randomly inside the gameboard. | 4 |
| 5 | Implementing functionalities of a cell (when clicked). | 5 |
| 6 | Implementing a method to find a path is available to match cells. | 6 |
| 7 | Win Detection and Timer Initialization. | 10 |
| 8 | Conclusion | 10 |
| 9 | Works Cited | 10 |

## INTRODUCTION

The implementation of Onet Connect puzzle by our team is done using .NET framework and C# programming language. .NET framework is a framework which is used for software development on Desktop, Websites, Server apps, etc. This framework is used for both frontend and backend development. It is the second most popular framework which is used to develop almost anything. Our application uses .NET framework for both frontend and backend. We chose C# as our programming language because it provides tons of libraries, UI components and other resources which would help in the game development. Both C# and .Net can be combined to develop a ton of application including cloud-based services, websites and applications. We will now discuss more about .NET framework and C# before jumping into explanation of our game.

## .NET Framework

.NET framework is created by Microsoft which initially supported only by windows application and originally released by 2002. The environment provided by .NET framework is enormous as it supports almost all kind of applications like cloud-based applications, windows desktop services and web-based services. (engineering, 2021)

## Pros of .NET Framework

* It uses OOP (Object Oriented Programming) concept.
* Visual Studio brings along .NET with it, which will allow developers to develop applications without much of stress.
* It is easy to deploy and maintain when .NET is used. (MELNICHUK, 2020)

## Cons .NET Framework

* Incurs Licencing cost if not a student or for personal use.
* With new releases comes stability problems. (MELNICHUK, 2020)

Let us now discuss the implementation of Onet Connect puzzle game,

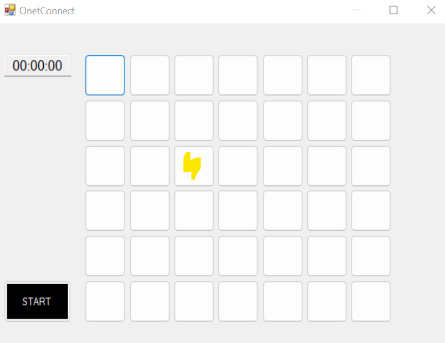
**What is an Onet Connect game?**

An Onet Connect is a single player puzzle game where any user has to eliminate all the tile by clicking 2 matching tiles when time is ticking. (apple.inc, n.d.)It can be implemented in several different ways, using several different framework or programming language. The ways of implementing tile can be differed, it can of colours, Animals, Fruits, Numbers etc. We have implemented using colours. In our implementation, a user should eliminate the tile by matching the tile of same colour with path available. There are several steps in creation of the Onet Connect game by our methodology and we have described each and every step below with functionalities and mathematical annotations. The steps are,

* Design of the playing board (UI).
* Placing solvable number of Colours inside board in a random order in all the cells.
* Implementing functionalities of each cell when clicked as first or second time after clicking another cell.
* If clicked second time matching the path, if it is solvable remove colour.
* Detection of win move and display win.
* Adding additional functionalities (Timer, Start button, etc.).

## Design of the Playing Board

There are many ways the cells can be arranged, but we have used a single pattern placing the cells in 6x7 matrix. Where it has 6 rows and 7 columns. Considering each and every cell has an address, we will be solving the path between cells. The address of a cell is denoted by a [i x j], where ‘a’ is a cell, ‘i’ is the row number and ‘j’ is the column where cell is present.

For example, In the image, a highlighted cell in yellow is present, the address of the cell is considered as,

a [3 x 3].

Here first number is row address, 2nd number is column address.

The Cells are made of series of buttons arranged in the order of 6 x 7 matrix. When each button click is made, the address of the cell is calculated and different functions are called. Below is the snippet to find the address of the cell,

public int findLocationOfAButton (int buttonIndex)

{

int loc,i,j;

if (buttonIndex % 7 != 0)

{

i = (buttonIndex / 7) + 1;

j = (buttonIndex % 7);

}

else

{

i = (buttonIndex / 7);

j = 7;

}

loc = (i \* 10) + j;

Console.WriteLine("location " + loc);

return loc;

# }

In the above code we have used 7, to find mod value and division value because the game board we are using has 7 columns.

## Placing Solvable number of Colours randomly inside the Game board

Since our game board is of size 6x7, there are 42 cells in our game board. And the colours placed inside the game board should have suitable other pair to get eliminated and the game can be won. So, we decided to add 7 colours inside the game board (colour count can be increased if needed) and each cell should be coloured randomly from those 7 colours, therefore created a switch case statement or if condition with 7 colours, then generated a random number and then assigned it to a variable and passed it to the statement to choose the colour. And whenever a cell is coloured from first 21 cells, a suitable other pair from second half is painted with same colour so that every cell has a matching pair (i.e.) any colour should be present even number of times inside the game board.

Colour selection using random number logic,

Random rnd = new Random();

number = rnd.Next(1, 7);

if (number == 1)

{

color = Color.SkyBlue;

colorName = color.Name;

}

else if (number == 2)

{

color = Color.DarkRed;

colorName = color.Name;

}

else if (number == 3)

{

color = Color.Orange;

colorName = color.Name;

}

else if (number == 4)

{

color = Color.Yellow;

colorName = color.Name;

}

else if (number == 5)

{

color = Color.Green;

colorName = color.Name;

}

else if (number == 6)

{

color = Color.DarkViolet;

colorName = color.Name;

}

else

{

color = Color.Pink;

colorName = color.Name;

}

## Implementing functionalities of a cell (when clicked)

There are 42 cells in our game board, which is created using 42 buttons. All the buttons do have similar functionality; therefore, we can create functionality for a button and then use same structure for all other cells. To have our cell working there are pointer variable which needed to be initialised, which are

* currentButton
* previousButton
* currentColor
* previousColor

These pointer variables help us to match the cells with colours and find the path available to match them. Each time a button is selected the button number is updated in the pointer ‘currentButton’ and colour of the button is stored in ‘currentColor’. And when again a button is selected, current values are transferred to ‘previousButton’ and ‘previousColor’, to update ‘currentButton’ and ‘currentColor’ with new selection. When ‘currentColor’ and ‘previousColor’ matches, ‘ifPossible’ method is called to find out if path is available to connect cells. When the colours are not matched or when path is not available, the current values and previous values are set to null. And when colour and path is matched, the colour of the cells are changed to white denoting elimination. The main functionality of this method code is given below,

currentButton = "button1";

currentColor= button1.BackColor.Name;

if (previousButton != "" && previousColor!="" && currentButton!=previousButton)

{

if (previousColor == currentColor)

{

int buttonNum1=getButtonNum(previousButton);

int buttonNum2 = getButtonNum(currentButton);

int location1=findLocationOfAButton(buttonNum1);

int location2=findLocationOfAButton(buttonNum2);

if (ifPossible(location1, location2))

{

this.Controls[previousButton].BackColor = Color.White;

this.Controls[currentButton].BackColor = Color.White;

}

}

currentColor = "";

previousColor = "";

previousButton = "";

}

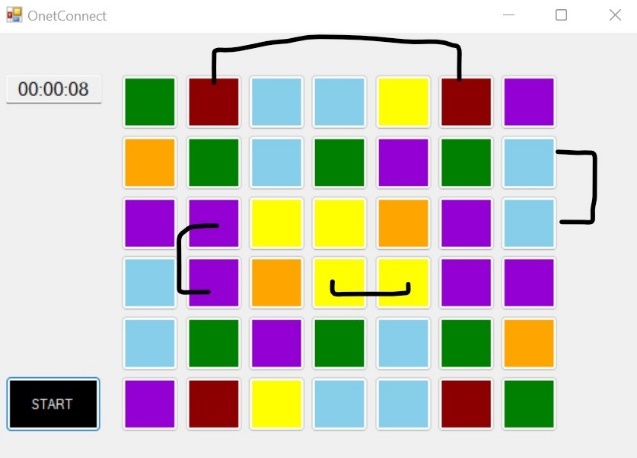
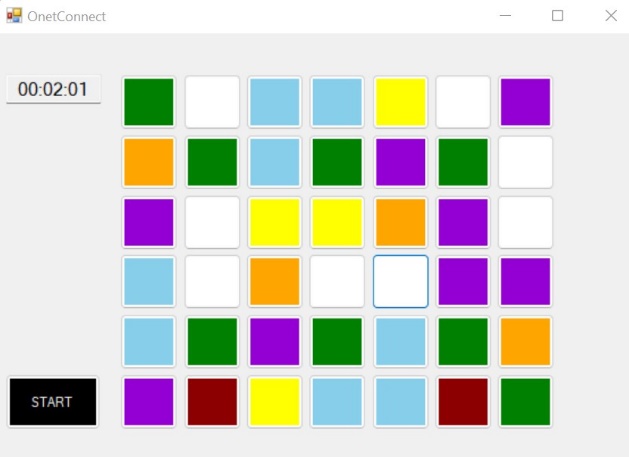
else

{

previousColor = currentColor;

previousButton = "button1";

}

Before Eliminating After Eliminating

## Implementing a method to find if path is available to match cells

When two different cells are matched with colour, their location is found and sent to this method. We have put this method’s return type as Boolean, therefore it will return true if path is available or false if path is not available. While creating a method, we have to set default return type as false and then we can use if statements inside the method to breakdown the function to different small conditions to find the path and return true if path available. The path should be no more than two right angled turns to connect cells and there should not be any cell blocking the matching path. This condition can be simplified to several small conditions in terms of row x column value, and they are listed as below,

Here a[i1 x j1] is location of cell 1 and a[i2 x j2] is location of cell 2.

* The matching cells can be next to each other in the same column or same row.

That is when i1==i2 or j1==j2

When i1==i2, should be j1+1==j2 or j1-1==j2.

When j1==j2, should be i1+1==i2 or i1-1==i2.

* The matching cells can be in same row provided there is no coloured cells in between them,

That is i1==i2 or j1==j2

Now fetch colour of next available cell between cells.

for(int a = j1 + 1; a < j2; a++)

{

buttonNum1 = ((i1 - 1) \* 7) + a;

String button=Convert.ToString(buttonNum1);

String tempButtonName = "button" + button;

String middleColor = this.Controls[tempButtonName].BackColor.Name;

if (middleColor == "White")

{

if (a == j2 - 1)

{

return true;

}

continue;

}

if (middleColor != "White")

{

break;

}

}

If blocked break the loop.

If it is white, move on to next cell until you reach desired cell.

* The cells can be in same row or column and be blocked by middle cells when it is present in outer box,

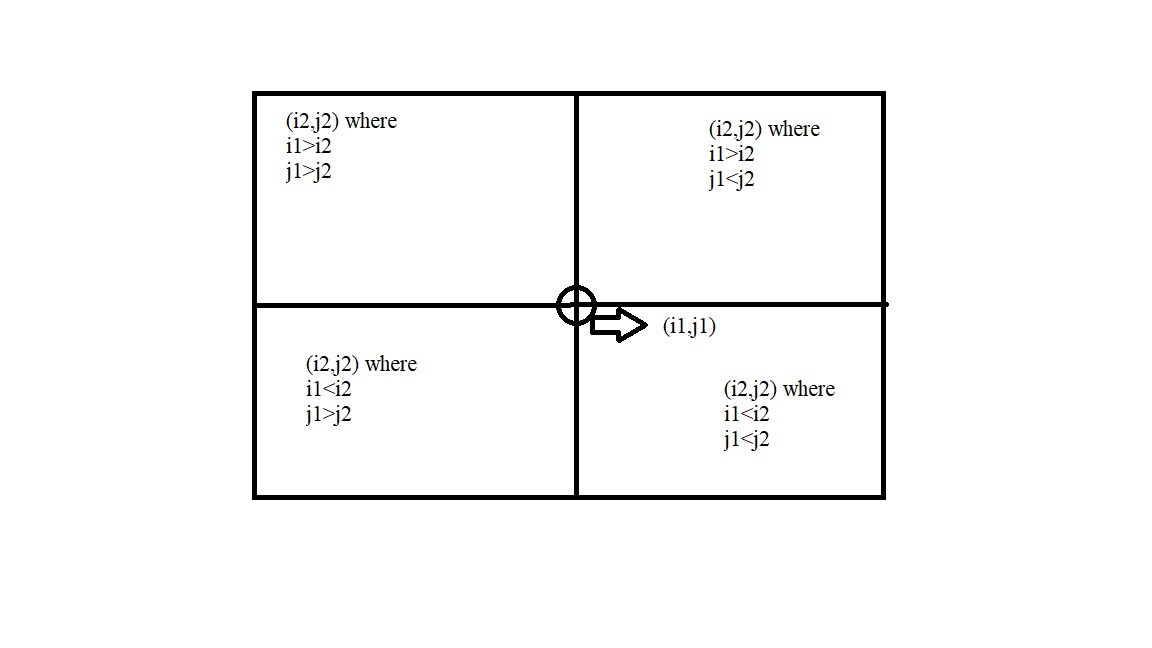
When i1==1 and i2==1

Or j1==1 and j2==1

Or i1==6 and i2==6

Or j1==7 and j2==7

* Implementing the L connection, we have explained it using a diagram below,



Considering a position for 1st cell a[i1 x j1], there are four different possibilities for location of 2nd cell a[i2 x j2].

And they are,

* Top right quadrant
  + Where i1>i2 and j1<j2.
* Top left quadrant
  + Where i1>i2 and j1>j2.
* Bottom right quadrant
  + Where i1<i2 and j1<j2.
* Bottom left quadrant
  + Where i1<i2 and j1>j2.
* Using above conditions, additional pointers are created to identify the path. They are,
  + Boolean blocked
  + Boolean turnRight
  + Boolean turnLeft.
* Using the condition above and the pointer, below is the solution for two of the quadrants (top right and bottom left),

Boolean blocked = false, turnLeft = false, turnRight = false;

int i=i1,j=j1,k=i2,l=j2;

if (i2 > i1)

{

e= i1;

f = i2;

}

if (i1 > i2)

{

e= i2; f = i1;

}

if (j2 > j1)

{

g= j1;

h = j2;

}if(j1> j2)

{

g = j2;

h = j1;

}

j2 = h;

j1 = g;

i1 = e; i2=f;

if ((j2 > j1 && i2 > i1)||(j2 < j1 && i2 < i1))

{

for(int a = j1 + 1; a <= j2; a++)

{

buttonNum1 = ((i1 - 1) \* 7) + a;

String button = Convert.ToString(buttonNum1);

String tempButtonName = "button" + button;

String middleColor = this.Controls[tempButtonName].BackColor.Name;

if (middleColor == "White")

{

if (a == j2)

{

turnRight = true;

blocked = false;

}

continue;

}

if (middleColor != "White")

{

blocked = true;

turnRight=false;

break;

}

}

if (turnRight == true&& blocked == false)

{

if (i1 + 1 == i2)

{

return true;

}

for (int a = i1 + 1; a <= i2; a++)

{

buttonNum1 = ((a - 1) \* 7) + j2;

String button = Convert.ToString(buttonNum1);

String tempButtonName = "button" + button;

String middleColor = this.Controls[tempButtonName].BackColor.Name;

if (middleColor == "White")

{

if (a == i2-1)

{

return true;

}

continue;

}

if (middleColor != "White")

{

blocked = true;

turnRight = false;

break;

}

}

}

if (blocked == true && turnRight == false)

{

for (int a = i1 + 1; a <= i2; a++)

{

buttonNum1 = ((a - 1) \* 7) + j1;

String button = Convert.ToString(buttonNum1);

String tempButtonName = "button" + button;

String middleColor = this.Controls[tempButtonName].BackColor.Name;

if (middleColor == "White")

{

if (a == i2)

{

turnLeft = true;

blocked = false;

}

continue;

}

if (middleColor != "White")

{

blocked = true;

turnLeft = false;

break;

}

}

if (blocked == false && turnLeft == true)

{

if (j1 + 1 == j2)

{

return true;

}

for (int a = j1 + 1; a <= j2; a++)

{

buttonNum1 = ((i2 - 1) \* 7) + a;

String button = Convert.ToString(buttonNum1);

String tempButtonName = "button" + button;

String middleColor = this.Controls[tempButtonName].BackColor.Name;

if (middleColor == "White")

{

if (a == j2-1)

{

blocked = false;

return true;

}

continue;

}

if (middleColor != "White")

{

blocked = true;

break;

}

}

}

}

Similarly, the by changing values of i1,j1,i2,j2,turnLeft to right and right to left, solution for other 2 quadrants can be created.

## Win Detection and Timer Initialization

For the user to win, all the cells in the game board should be white. We can write a simple method, which can be called in side every button clicks. On redirecting to this particular method, it will run a loop to fetch colours of all cells. If every cell is in white colour, we can throw a dialogue of win to the user. Below is the code and image of implemented win move. And timer is made of text box and a start button. Start button is connected to generate board and text box, once it is clicked, timer starts running from 0 and board is generated.

Boolean flag=false;

for(int i=1; i <= 42; i++)

{

String buttonName = "button" + i;

String C = this.Controls[buttonName].BackColor.Name;

if(C != "White")

{

break;

}

if (i == 42 && C == "White")

{

flag = true;

}

} if (flag == true)

{

timer.Stop();

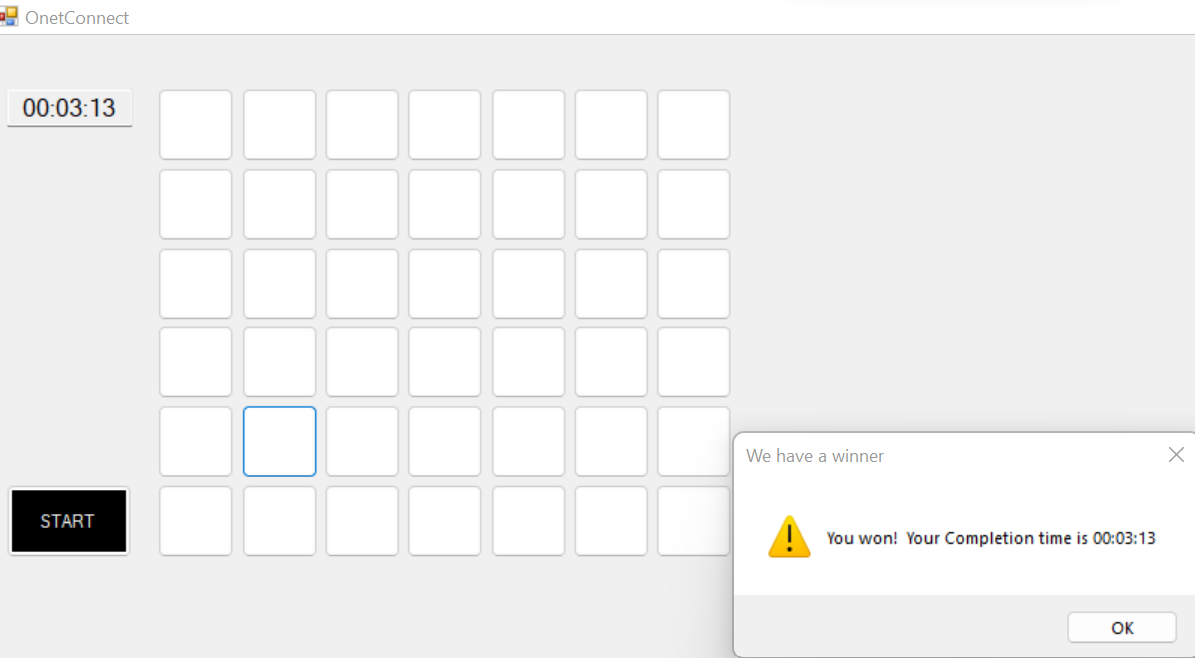
String completionTime = finishTime.Text;

MessageBox.Show("You won! Your Completion time is "+completionTime, "We have a winner", MessageBoxButtons.OK,MessageBoxIcon.Exclamation);

generateBoard();

finishTime.Text = "00:00:00";

}



## Conclusion

We have attached much more code snippets and video to the zip file submitted. Thus, we have implemented the game of Onet Connect from our coursework-2 successfully. We have learned a lot through the process of this creation and are eagerly waiting for any upcoming challenges.

# Works Cited

apple.inc. (n.d.). *apple*.

engineering, A. s. (2021, December). *.NET Framework Programming*. Retrieved from https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-net-framework-programming/

MELNICHUK, A. (2020, July 24). *Advantages and Disadvantages of .NET Framework*. Retrieved from https://ncube.com/blog/pros-and-cons-of-net-framework