**My MINEFIELD GAME**

**My C# game looks like this:**

**A picture containing chart

Description automatically generated**

**Summary:**

In the C# minefield game inspired by 'Finding Nemo', players assume the role of Marlin, navigating perilous waters to reunite with his son, Nemo. The game features multiple difficulty levels that adjust the number of jellyfish mines and impose time constraints. A helpful mine detector assists players by indicating nearby mines. Gameplay entails maneuvering Marlin using arrow keys while avoiding mines; collisions result in game over. Successfully guiding Marlin to Nemo secures victory. Players can restart at the same or a new difficulty level, with elapsed time tracking. Keyboard shortcuts allow players to reveal or hide mines.

**Game Mechanics Overview:**

Themed after the movie 'Finding Nemo', the C# minefield game sets the player as Marlin, the father, situated at the bottom left corner, while Nemo, his son, awaits at the top.

Objective:

Guide Marlin through the perilous waters to reunite with Nemo. However, numerous jellyfish mines obstruct their path, and time is limited, the duration varying with the chosen difficulty level.

Difficulty Levels:

Upon launching, the game defaults to the Moderate difficulty setting. Players can adjust the challenge by selecting from Easy, Moderate, Hard, or Impossible modes. Each difficulty level features a different quantity of jellyfish mines and distinct time constraints.

Gameplay:

Initiate the game by pressing "START" after selecting the desired difficulty level. Navigate Marlin using the arrow keys on either the screen or keyboard. Be cautious not to collide with the jellyfish mines, as doing so halts the game, reveals the mines, displays a corresponding consequence image, and results in failure. Likewise, running out of time ends the game unfavorably.

Victory Conditions:

Successfully guiding Marlin to Nemo results in a win, accompanied by an image of their heartwarming reunion.

Restarting the Game:

If unsuccessful, players can restart by clicking "Try again," which maintains the chosen difficulty level, or by selecting a new level and pressing "Start."

Elapsed Time Tracking:

The game also records the elapsed time of each session.

Controls:

Navigate: Arrow keys (on-screen or keyboard)

Reveal Mines: Blue button (or "M" on the keyboard), usable twice per game for a limited duration.

Hide Mines: Optional red button (for immediate concealment)

Restart: "Space" key on the keyboard or dedicated restart button

**What I did to achieve this:**

In my minefield game, I incorporated the theme of the movie 'Finding Nemo' by introducing characters representing Nemo and his father. Using jellyfish PNG images as the mines further reinforced the thematic connection. These images, along with others, were seamlessly integrated into the game through project resources, and I adjusted label properties accordingly.

To enhance the immersive experience, I unified the colour scheme by setting all panel labels to shades of blue to mirror the ocean. Additionally, I utilized lighter shades of blue for various elements such as arrows, trails, background during mine reveals, and the form border upon game completion.

A key feature I implemented was a 2 time only sneak peek button (in blue) to show the Mines with an optional hide Mines button (in red). I used an int counter and set a condition to limit the number of times the button could be used for and a timer so the mines would only be shown for a couple of seconds before being hidden again.

For the game timer I used another timer from toolbox and set a condition so it would countdown if my “int timeleft” was > 0 or else to stop the game. This resets with every game, coded amongst the difficulty levels (Game time varies with each difficulty level)

As for the difficulty levels I used radio buttons, coded within “Private void start ()” for when one radiobutton was selected it would load the selected difficulty level and reset the game after the button “Start”, “Try again” has been pressed or when the form is loaded (loads with moderate level being already selected)

Furthermore, I added another timer for the time Elapsed by adding “using System.Diagnostics” and the help of a string, timer & a label.

Lastly, I enriched the gaming experience by implementing keyboard controls via a protected override bool method, facilitating button clicks. Additionally, I customized the form icon to further personalize the interface.

In summary, the program operates efficiently, incorporating both simple and advanced features, with careful attention to detail evident in the custom layout, meaningful object names, and extensive commenting throughout the codebase.

Code:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Diagnostics;

namespace MINEFIELD\_GAME\_0001

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

//size of grid

const int rows = 20;

const int cols = 20;

//set start location of sprite

int atCol = 0;

int atRow = 19;

//Boolean Array for Mine locations!

bool[,] mines = new bool[rows, cols];

private void panel1\_Paint(object sender, PaintEventArgs e)

{

}

private void label1\_Click(object sender, EventArgs e)

{

}

private void Start() //Restarts the game at the same diffculty that was selected

//(reset start point, hide mines and previous steps, Plant more mines in new random locations etc..)

{

stopWatch.Reset();

stopWatch.Start();

tmrGameTime.Enabled = true;

//Difficulty levels:

if (rbEasy.Checked == true) // Easy (25 mines, 100 seconds)

{

HideMinesRestart();

this.BackColor = Color.WhiteSmoke;

hideSpriteAt(atRow, atCol);

atRow = +19; //reset location to Row 19

atCol = +0; //reset location to Col 0

showSpriteAt(atRow, atCol);

label20.Image = Properties.Resources.nemo03;

btnUp.Image = Properties.Resources.ARROW\_UP02;

btnDown.Image = Properties.Resources.ARROW\_DOWN;

btnLeft.Image = Properties.Resources.ARROW\_LEFT;

btnRight.Image = Properties.Resources.ARROW\_RIGHT;

btnDown.Enabled = true;

btnUp.Enabled = true;

btnLeft.Enabled = true;

btnRight.Enabled = true;

btncounter = 0;

lblMineCount.Text = "Jellyfish Nearby:" + mineCount(atRow, atCol);

lblEnd.Image = null;

timeleft = 100;

plantMines(25);

}

if (rbMod.Checked == true) // Moderate (35 mines, 100 seconds)

{

HideMinesRestart();

this.BackColor = Color.WhiteSmoke;

hideSpriteAt(atRow, atCol);

atRow = +19; //reset location to Row 19

atCol = +0; //reset location to Col 0

showSpriteAt(atRow, atCol);

label20.Image = Properties.Resources.nemo03;

btnUp.Image = Properties.Resources.ARROW\_UP02;

btnDown.Image = Properties.Resources.ARROW\_DOWN;

btnLeft.Image = Properties.Resources.ARROW\_LEFT;

btnRight.Image = Properties.Resources.ARROW\_RIGHT;

btnDown.Enabled = true;

btnUp.Enabled = true;

btnLeft.Enabled = true;

btnRight.Enabled = true;

btncounter = 0;

lblMineCount.Text = "Jellyfish Nearby:" + mineCount(atRow, atCol);

lblEnd.Image = null;

timeleft = 100;

plantMines(35);

}

if (rbHard.Checked == true) //Hard (50 mines, 80 seconds)

{

HideMinesRestart();

this.BackColor = Color.WhiteSmoke;

hideSpriteAt(atRow, atCol);

atRow = +19; //reset location to Row 19

atCol = +0; //reset location to Col 0

showSpriteAt(atRow, atCol);

label20.Image = Properties.Resources.nemo03;

btnUp.Image = Properties.Resources.ARROW\_UP02;

btnDown.Image = Properties.Resources.ARROW\_DOWN;

btnLeft.Image = Properties.Resources.ARROW\_LEFT;

btnRight.Image = Properties.Resources.ARROW\_RIGHT;

btnDown.Enabled = true;

btnUp.Enabled = true;

btnLeft.Enabled = true;

btnRight.Enabled = true;

btncounter = 0;

lblMineCount.Text = "Jellyfish Nearby:" + mineCount(atRow, atCol);

lblEnd.Image = null;

timeleft = 80;

plantMines(50);

}

if (rbImp.Checked == true) //Impossible (75 mines + only 50 seconds!)

{

HideMinesRestart();

this.BackColor = Color.WhiteSmoke;

hideSpriteAt(atRow, atCol);

atRow = +19; //reset location to Row 19

atCol = +0; //reset location to Col 0

showSpriteAt(atRow, atCol);

label20.Image = Properties.Resources.nemo03;

btnUp.Image = Properties.Resources.ARROW\_UP02;

btnDown.Image = Properties.Resources.ARROW\_DOWN;

btnLeft.Image = Properties.Resources.ARROW\_LEFT;

btnRight.Image = Properties.Resources.ARROW\_RIGHT;

btnDown.Enabled = true;

btnUp.Enabled = true;

btnLeft.Enabled = true;

btnRight.Enabled = true;

btncounter = 0;

lblMineCount.Text = "Jellyfish Nearby:" + mineCount(atRow, atCol);

lblEnd.Image = null;

timeleft = 50;

plantMines(75);

}

}

private void Form1\_Load(object sender, EventArgs e)

{

showSpriteAt(atRow, atCol);

label20.Image = Properties.Resources.nemo03;

btnUp.Image = Properties.Resources.ARROW\_UP02;

btnDown.Image = Properties.Resources.ARROW\_DOWN;

btnLeft.Image = Properties.Resources.ARROW\_LEFT;

btnRight.Image = Properties.Resources.ARROW\_RIGHT;

plantMines(35);

rbMod.Checked = true; // Default difficulty game level checked

stopWatch = new Stopwatch();

stopWatch.Start();

lblStopWatch.Image = Properties.Resources.stopwatch;

}

//subroutine to plant minefield!

private void plantMines(int toBeSet)

{

Random r = new Random();// Random number generator

//Variables

int setSoFar = 0;

int tryRow, tryCol;

//clear all current mines array( RESET?)

Array.Clear(mines, 0, mines.Length);

//Loop fill array

do

{

tryRow = r.Next(0, rows);

tryCol = r.Next(0, cols);

//prevent mines at these locations:

if (tryRow == -1 && tryCol == 0)

continue;

if (tryRow == rows - 1 && tryCol == 1)

continue;

if (tryRow == rows - 2 && tryCol == 0)

continue;

if (tryRow == rows - 2 && tryCol == 1)

continue;

if (tryRow == 0 && tryCol == 19)

continue;

if (tryRow == 19 && tryCol == 1)

continue;

if (tryRow == 18 && tryCol == 2)

continue;

if (tryRow == 18 && tryCol == 1)

continue;

if (tryRow == 18 && tryCol == 0)

continue;

if (!mines[tryRow, tryCol])// if it doesnt(not) already have a mine(true)

{

mines[tryRow, tryCol] = true; // place mine

setSoFar++;

}

} while (setSoFar < toBeSet);

}

//function to show sprite at any grid location

private void showSpriteAt(int Row, int Col)

{

Label lbl = getLabel(Row, Col);

lbl.BackColor = Color.LightBlue;

lbl.Image = Properties.Resources.nemoDAD03;

}

//function to show sprite at any grid location

private void hideSpriteAt(int Row, int Col)

{

Label lbl = getLabel(Row, Col);

lbl.Image = null;

}

private int mineCount(int atR, int atC) //Counts number of Jellyfish nearby

{

int counted = 0;

if (atCol > 0)

{

if (mines[atR, atC - 1])

counted++;

}

if (atC < cols - 1)

{

if (mines[atR, atC + 1])

counted++;

}

if (atR > 0)

{

if (mines[atR - 1, atC])

counted++;

}

if (atR < rows - 1)

{

if (mines[atR + 1, atC])

counted++;

}

return counted;

}

//Show all Mines

private void showMines()

{

Label lbl;

for (int Row = 0; Row < 20; Row++)

{

for (int Col = 0; Col < 20; Col++)

{

lbl = getLabel(Col, Row);

if (mines[Col, Row])

{

lbl.Image = Properties.Resources.jellyfish;

}

else if (lbl.BackColor == Color.LightBlue)

{

continue;

}

else

{

lbl.BackColor = Color.LightSkyBlue;

}

}

}

}

private void HideMines() // hide all mines but not previous steps

{

Label lbl;

for (int Row = 0; Row < 20; Row++)

{

for (int Col = 0; Col < 20; Col++)

{

lbl = getLabel(Col, Row);

if (mines[Col, Row])

{

lbl.Image = null;

lbl.BackColor = Color.Blue;

}

else if (lbl.BackColor == Color.LightBlue)

{

continue;

}

else

{

lbl.BackColor = Color.Blue;

}

}

}

}

private void HideMinesRestart() // hide all mines and previous steps

{

Label lbl;

for (int Row = 0; Row < 20; Row++)

{

for (int Col = 0; Col < 20; Col++)

{

lbl = getLabel(Col, Row);

if (mines[Col, Row])

{

lbl.Image = null;

lbl.BackColor = Color.Blue;

}

else

{

lbl.BackColor = Color.Blue;

}

}

}

}

//Did I hit a Jellyfish?

private void amIDead(int Col, int Row)

{

if (mines[Col, Row])//Yes, you die and lose the game

{

this.BackColor = Color.Red;

btnDown.Enabled = false;

btnUp.Enabled = false;

btnLeft.Enabled = false;

btnRight.Enabled = false;

showMines();

lblMineCount.Text = "Oops!! You died!!";

lblEnd.Image = Properties.Resources.Deadfish03; //Dislays image of your fish dead after being killed by Jellyfish

ShowMineTimer = 2;

tmrShowMines.Enabled = false;

tmrGameTime.Enabled = false;

stopWatch.Stop();

}

else if (atCol == 19 && atRow == 0) //You find nemo and win the game!

{

this.BackColor = Color.SkyBlue;

btnDown.Enabled = false;

btnUp.Enabled = false;

btnLeft.Enabled = false;

btnRight.Enabled = false;

showMines();

lblMineCount.Text = "You found Nemo!! ";

lblEnd.Image = Properties.Resources.NemoNDad02;

ShowMineTimer = 2;

tmrShowMines.Enabled = false;

tmrGameTime.Enabled = false;

stopWatch.Stop();

}

else //no-Carry on

{

lblMineCount.Text = "Jellyfish Nearby:" + mineCount(atRow, atCol);

}

}

//function to return a Label at a given grid location

private Label getLabel(int Row, int Col)

{

// k wikll be the label number we are seeking

int k = Row \* 20 + Col + 1;

string s = "label" + k.ToString();

foreach (Control c in panel1.Controls)

{

if (c.Name == s)

{

return (Label)c;

}

}

return null;

}

private void btnStart\_Click(object sender, EventArgs e) //Start game at difficulty level selected

{

Start();

}

private void btnUp\_Click(object sender, EventArgs e) // Move Up

{

if (atRow > 0)

{

hideSpriteAt(atRow, atCol);// delete current location

atRow--; // update location(up a row)

showSpriteAt(atRow, atCol); // display at current location

amIDead(atRow, atCol);

}

}

private void btnDown\_Click(object sender, EventArgs e) // Move Down

{

if (atRow < 19)

{

hideSpriteAt(atRow, atCol);

atRow++;

showSpriteAt(atRow, atCol);

lblMineCount.Text = "Jellyfish Nearby:" + mineCount(atRow, atCol).ToString();

amIDead(atRow, atCol);

}

}

private void btnRight\_Click(object sender, EventArgs e) //Move Right

{

if (atCol < 19)

{

hideSpriteAt(atRow, atCol);

atCol++;

showSpriteAt(atRow, atCol);

amIDead(atRow, atCol);

}

}

private void btnLeft\_Click(object sender, EventArgs e) // Move Left

{

if (atCol > 0)

{

hideSpriteAt(atRow, atCol);

atCol--;

showSpriteAt(atRow, atCol);

amIDead(atRow, atCol);

}

}

private void btnHideMines\_Click(object sender, EventArgs e) // Red Hide Mine button

// (in case user doesnt want to wait the full ShowMine Time after using the showMine sneak peak button)

{

HideMines();

}

private void btnRestart\_Click(object sender, EventArgs e) // Try again button to restart game at the same diffulty lvl selected

{

Start();

}

//ShowMines button counter

int btncounter = 0;

//ShowMine Time counter

int ShowMineTimer = 0;

private void btnShowMines\_Click(object sender, EventArgs e) // Show Mines(max 2 times) for limited time

{

tmrShowMines.Enabled = true;

ShowMineTimer = 1; // set showMines time limit to 1 once btnShowMines has been pressed.

btncounter++;

if (btncounter < 3)

{

showMines();

}

}

private void tmrShowMines\_Tick(object sender, EventArgs e) // ShowMines countdown

{

if (ShowMineTimer > 0)

{

ShowMineTimer = ShowMineTimer -= 1;

}

else

{

HideMines();

}

}

private void lblStopWatch\_Click(object sender, EventArgs e)

{

}

int timeleft = 100; // default game time

private void tmrGameTime\_Tick(object sender, EventArgs e)

{

if (timeleft > 0)

{

timeleft = timeleft -= 1;

lblMyTime.Text = timeleft + "seconds";

}

else

{

this.BackColor = Color.Red;

btnDown.Enabled = false;

btnUp.Enabled = false;

btnLeft.Enabled = false;

btnRight.Enabled = false;

showMines();

lblMineCount.Text = "Out of Time!!";

lblEnd.Image = Properties.Resources.Time03;

ShowMineTimer = 2;

tmrShowMines.Enabled = false;

stopWatch.Stop();

tmrGameTime.Enabled = false;

}

}

private Stopwatch stopWatch; //Time Elapsed

private void tmrTimeElapsed\_Tick(object sender, EventArgs e)

{

lblTimeElapsed.Text = string.Format("{0:hh\\:mm\\:ss}", stopWatch.Elapsed);

}

protected override bool ProcessCmdKey(ref Message msg, Keys keyData) // Buttons to react to Keys on Keyboard

{

if (keyData == Keys.Up)

{

btnUp.PerformClick();

}

if (keyData == Keys.Down)

{

btnDown.PerformClick();

}

if (keyData == Keys.Right)

{

btnRight.PerformClick();

}

if (keyData == Keys.Left)

{

btnLeft.PerformClick();

}

if (keyData == Keys.Space)

{

btnRestart.PerformClick();

}

if (keyData == Keys.M)

{

btnShowMines.PerformClick();

}

return base.ProcessCmdKey(ref msg, keyData);

}

private void rbEasy\_CheckedChanged(object sender, EventArgs e)

{

}

private void rbMod\_CheckedChanged(object sender, EventArgs e)

{

}

private void gbDif\_Enter(object sender, EventArgs e)

{

}

private void rbHard\_CheckedChanged(object sender, EventArgs e)

{

}

private void rbImp\_CheckedChanged(object sender, EventArgs e)

{

}

}

}