

Sapper (Minesweeper)

A-Level Computer Science NEA [Summer 2022]

Andrew Smith (8274)
POOLE HIGH SCHOOL (55227)
COMPUTER SCIENCE NEA PROJECT SUMMER 2022

Table of Contents

<u>Section 1: Analysis</u>	3
1.1 Problem Statement	3
1.2 Client Information	3
1.3 Background to the problem	3
1.4 Interview Summary	4
1.5 Client Objectives	4
1.6 Research	6
<u>Section 2: Design</u>	8
2.1 Modelling the problem	8
2.1.1 System Flow Chart	8
2.2 Class Diagram	9
2.3 Algorithms	9
2.4 Data Structures	13
2.5 Human-Computer Interaction (HCI)	15
<u>Section 3: Implementation</u>	16
<u>Section 4: Testing</u>	22
4.1 Approach to testing	22
4.2 Test Plan	22
4.3 Requirement Testing	58
<u>Section 5: Evaluation</u>	59
5.1 Evaluation of the objectives	59
5.2 Client Feedback & Review	62
5.3 Self Evaluation	63
5.4 Future Improvements	63
<u>Appendix A: Client Interview</u>	64
<u>Appendix B: Client Feedback</u>	65
<u>Appendix C: Code Listing</u>	67

Section 1: Analysis

1.1 Problem Statement

I am creating a windows form application that replicates the popular game from the nineties 'minesweeper'. This is because my client has asked me to produce this as they have a minesweeper tournament but do not want to pay the royalties to the actual game. Having me produce this for them means that my client can focus their resources (money) on other aspects of the tournament

1.2 Client Information

His name is Adam Bailey. My client is the owner of the Poole Retro Gaming Society. He has recently acquired this company as he used to be a regular customer at the society, but the previous owner had to sell it due to personal and financial reasons. It is noteworthy to add that Adam will be partaking in the tournament as he is an avid gamer who loves competition.

1.3 Background to the problem

The name of the organisation that has asked for my help is 'Poole Retro Gaming Society', that specialises in collecting retro antiques from gaming and auctioning them every year. There are only 5 employees, as they only reside in a medium sized unit on an industrial estate. I have been informed by my client that the company really only sees the same 75-100 people use their shop per week. They hold regular tournaments on retro games, and struggle to get licensing from major companies to host a financial based event using the game that was developed by them. By this, I mean that licensing can cost over £1000, which would be unreasonable to pay financially, seeing as the tournament is giving out £100 alone for the winner with a £5 entry. This is why I have been contacted, as I will be developing a simulation of minesweeper to get around this. Also, having a locally produced program to promote ticks the criteria for the company to show the public that they support local businesses. I have been strictly commissioned to create the game without any special rules, therefore I have been restricted to regular features of minesweeper.

1.4 Interview Summary (See [Appendix A](#) for interview)

My client Adam Bailey is a very keen lover of video games, and has always had strong passions for gaming from as early as he remembers. Adam took over the society so he could find like-minded people who share the same passion as him and feel as competitive as him, thus he has asked me to create a minesweeper like game to adhere to the nostalgic but competitive nature of the society. Adam also would like to save the initialisation of the game, so that the society can look back at the game they just played and try to further their knowledge of patterns that may arise. This would be ideal for the culture of the society, because each member can offer help to another member by going over games that they have played. My client has asked me personally to create this project with the program being computer based, as they only mainly have a couple of display computers that will be hosting the tournament. One final aspect that my client requested was that the program will be able to remove ONLY the adjacent fields if a zero is revealed, rather than removing all of the linked zeros at once, as it would increase the difficulty of the game, as it would require more user input which allows for an increase in the chance of a user errors (in game).

1.5 Client Objectives

1. The system must allow players to mark specific fields as flags (bomb spots)

If the player can identify where the bombs are, they would spend more time playing the game as part of the game is attempting to identify the bombs without taking too much time. This would make the game universally inviting to competition, as there would always be a time to improve on.

2. The system must be playable on an ordinary computer device (laptop, desktop) at home.

This feature should be used because if the club members love the game, it would be a good idea to make it playable at home so that they are not limited to playing at the club. This would also then allow for my client to promote the usage of local developers, and entice more local programmers to approach them or even work for them.

3. The system must allow the user to properly start and complete the game.

If the system fails to do so, the project will not even start and will fail to satisfy the client's fundamental requirements. The project must run smoothly from start to finish, with no errors or crashes.

4. The system must use minimal memory

If the system uses minimal memory, then the program can execute quickly, keeping the tournament standard and removing potential time issues from impacting the time. If you were to restart the game constantly, there wouldn't be much of an impact on memory usage, as there would only need to be up to 100 comparisons needed, as each space would need to compare if a bomb is near.

5. The system must be able to run multiple times within the same instance

This is needed for my client, because many games will be played on the same device, so having replay-ability means that the running of the tournament can be smoother. This also allows the society to use fewer machines, ultimately bringing down the cost of the tournament, which is essential for a small local club.

6. The system must have the ability to save the state of the game

Saving the location of the bombs and the values assigned allows a player to go back over the game they just had and can identify patterns where they went wrong/ could be quicker. This is needed for my client, as they would like to be able to create a community feeling around this tournament, and players can come together to identify patterns, and further improve their skill.

7. The system must allow the player to reveal a field

This is important, because the only way to actually play the game is to reveal the fields and spot where the 'bomb' spots are. This is important for my client, as they cannot actually have the game run correctly, without the initial foundations of it being developed.

8. The system must allow the user to save their name

This is important because the tournament would need a system to record and save the players names who complete a game. This would save the users first names and last names into a text file, which then also takes the score and difficulty that is able to be saved.

9. The system must allow the user to adjust difficulty

The system must do this, so the client can hold multiple stages. For example, the first round can be on a 5x5 grid; then the second round is on a 7x7 grid, etc... This is important for my client, because they may want to redistribute my software, and promote the fact that they accept local developers into their business / organisation.

10. The system must remove adjacent squares

The system must do this, because my client referenced in our interview that he wants the game to be as challenging as possible, so the instant removal of all zeroes if one is found would make the game way “too easy”.

1.6 Research

Minesweeper:

My client has requested specifically that the program is a near copy of the popular game ‘Minesweeper’ which works in a way that takes a square based grid and assigns certain spots within the grid a “bomb”. For each bomb that is present on the grid, each adjacent grid space has its counter incremented by one. This creates a grid that has different values on each grid space that represents ‘bombs’ adjacent.

Nonogram FRVR:

A nonogram works in a way that uses a predetermined grid like sudoku, but nonograms are picture logic puzzles in which cells in a grid must be colored or left blank according to numbers at the side of the grid to reveal a hidden picture. In this puzzle type, the numbers are a form of binary art that measures how many unbroken lines of filled-in squares there are in any given row or column. For example, a clue of "4 8 3" would mean there are sets of four, eight, and three filled squares, in that order, with at least one blank square between successive sets. This algorithmically works using RLE, where black squares would be determined by a ‘1’ and accumulated, and the ‘0’ are ignored. This isn’t suitable for my client, because it requires less skill, and wouldn’t make for a competitive environment due to the attention needed.

		1	5	2	5	² 1	2
2	1						
1	3						
1	2						
3							
4							
1							

SUDOKU:

Sudoku works in a way that already has a predetermined grid, so that the only algorithm that needs to be completed is a comparison if the number selected is in the correct space according to the predetermined grid order. This helped my project, because my project included creating a grid with 10 bombs in random places of the grid, and the bomb places assigned a value to the adjacent squares, creating a pre-determined grid that the player would have to use. However, this solution falls down in places, because it starts with giving the user hints, which isn’t ideal for my client as you would want to see players’ full cognitive ability without any help from the game itself in a tournament environment.

5	3		7				
6			1	9	5		
	9	8				6	
8			6				3
4			8	3			1
7			2				6
	6				2	8	
			4	1	9		5
			8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

BATTLESHIPS:

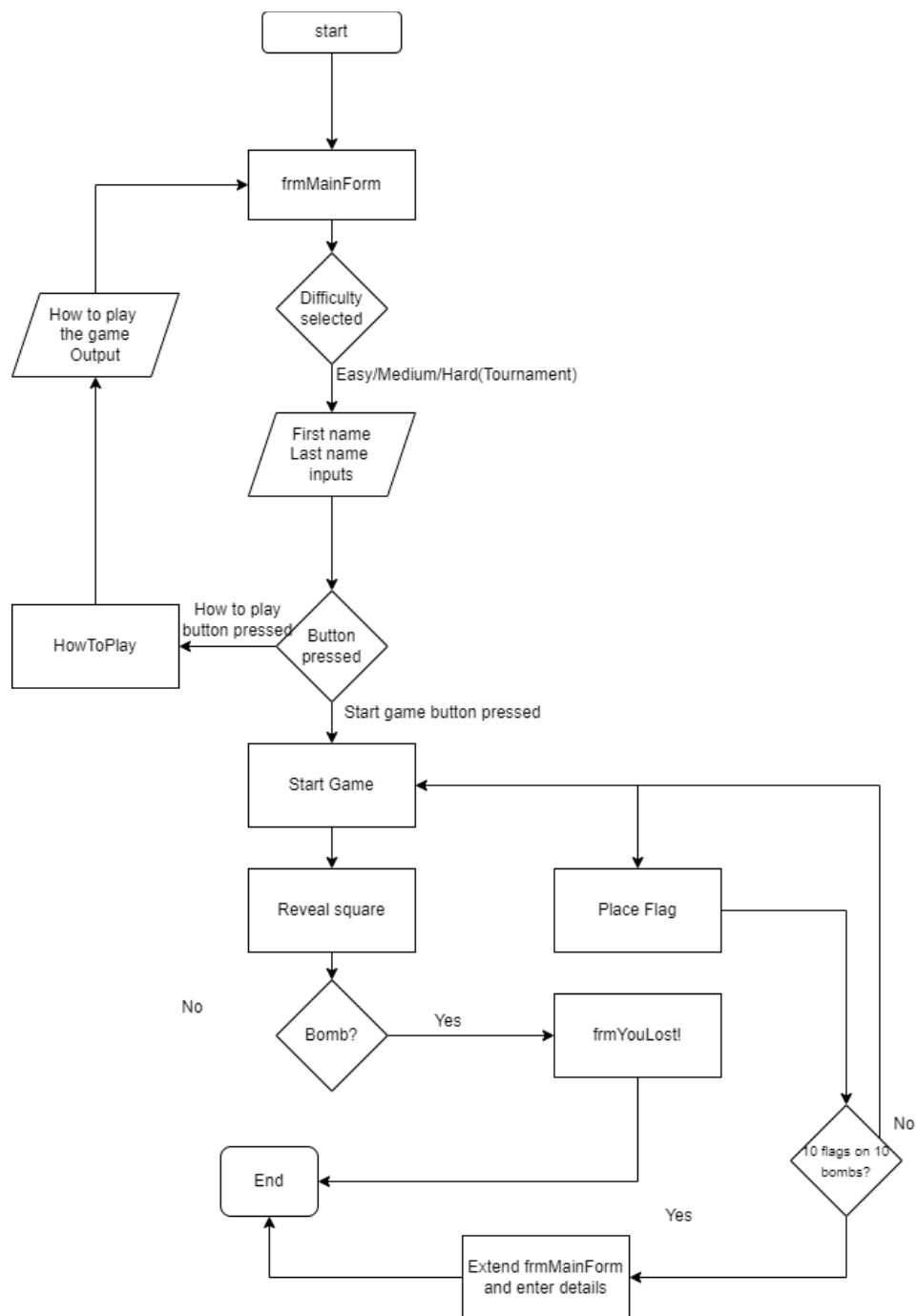
Battleships (solo) uses a randomised grid too, in the sense of the AI randomly assigns the ships which in this case would be a (1) to a space in a grid. However, in battleships you cannot see the value that is assigned to the grid until the grid piece is selected and “shot at”. This helped my solution, because it allowed me to hide the actual raw values of the grid until the grid space had been interacted with (whether that's removal of adjacent squares when a zero is found or an actual value is found). This creates almost two grids on top of each other, as the grid seen by the player covers the grid that has all values on it, until the grid space is removed. Battleships fall short of Sapper, because battleships have the opposite objective of Sapper, in that Battleships aim to hit the ‘mines’(ships), rather than just find their location. This shows that Sapper is more ideal for my client, because my client wants to see how players are able to work logically rather than mathematically

	A	B	C	D	E	F	G	H	I	J
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										

Section 2: Design

2.1 Modelling the problem

2.1.1 System Flow Chart

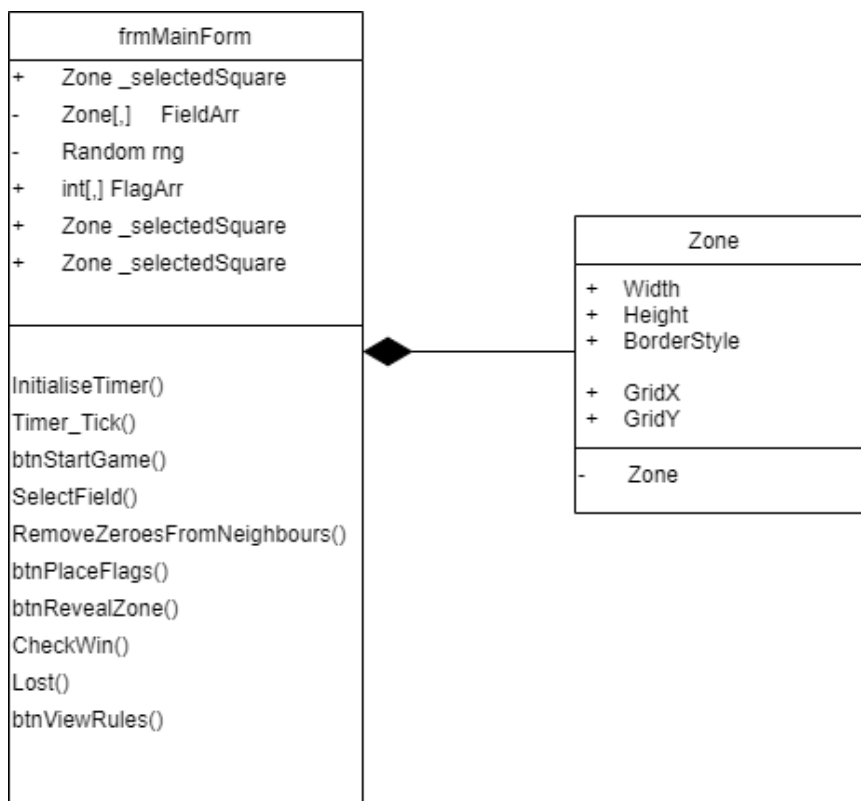


This flowchart represents my program as a whole. When you execute the program, frmMainForm opens, and you are forced to enter your name and select a difficulty. Then you are given the option to press one of two buttons: How to play, or start a game. Pressing the 'how to play' button, displays a messagebox with how Sapper is meant to be played. If you are to click the start game button, then the program will execute the correct procedures to display the game. Then, you choose whether you would like to reveal the square that you have selected or place a flag on the square as you feel there is a bomb underneath. Whenever you place a flag, the program checks whether there has been a flag placed on every bomb location. If this is true, then the program saves the players first name and last name to a text file.

2.2 Class Diagram

2.2.1 Class Diagram

My program only makes use of one class 'Zone' which creates the basic frameworks of the game. Each field within my grid is instantiated as a "zone" with the properties of gridx and gridy being used. Each instantiation of my class uses the GridX and GridY properties so that the GridArr array is able to locate each field according to their coordinates. The width and height properties relate to how wide each field will be, with the borderstyle being set to "fixedsingle" which allows me to show the user the disparity between a selected field and a non selected field.



2.3 Algorithms

2.3.1 Field reveal algorithm

This algorithm works by taking what grid space the user has clicked to reveal on the visual grid, and then printing the value associated using the picture file name in the debug folder (using the system.drawing library).

```
SquareBorderStyle ← 3D
SelectedSquare ← Image
NumToDisplay ← GridArr[GridX, GridY]
IF NumToDisplay == 0
    RemoveZeroesFromNeighbours #Removes adjacent squares
ELSEIF NumToDisplay == 9
    Lost
    #losing condition
ENDIF
```

2.3.2 Empty space reveal algorithm

This algorithm works in a way that checks what grid space has been clicked on, and if the value of the grid space clicked equates to '0', then this algorithm will execute by definition in terms of itself, removing each space adjacent. This is needed, so this is more of a quality of life (QoL) algorithm, as if you reveal a zero, it will be pointless and time wasting to go and reveal every single zero, so removing only the adjacent fields will help the 'smoothness' of the game.

```
IF Reveal Field button clicked THEN
    IF GridArr[GridX,GridY == 0) THEN
        RemoveZeroesFromNeighbour
    ENDIF
```

```
ENDIF
```

```
RemoveZeroesFromNeighbour
```

```
FOR i = -1 to 1
```

```
    FOR j = -1 to 1
```

```
        Load Image "number-" + GridArr[GridX + i, GridY + j]
```

```
    END FOR
```

```
END FOR
```

2.3.3 Minefield generation algorithm

This algorithm works in a way that inherits from class 'Zone'. Each grid space is an object of class Zone. At the same time, bombs are being randomly allocated an x and y position, to which these are then stored in the GridArr[]. This uses an int variable called 'bombsNeeded' which is assigned to the number that corresponds to the difficulty. Then, a condition controlled loop is used to repeatedly create the "bomb" locations. I have used condition controlled iteration rather than count controlled iteration, because the random generation of the 'GridX' and 'GridY' integers can result in a repeated location, so the game would attempt to place two bombs on the same field, creating an error in which there would not be enough bombs for the user to play the game the way it is intended. This problem is solved in my code, because the selection statement 'if (GridArr[GridX, GridY] != 9)' makes sure that the randomly assigned "location" doesn't already have a 'bomb' assigned to it. The reason that I have used the value '9' to compare values, is because the algorithm randomly generates between 1 and 9, and the algorithm accepts '9' as a bomb, so comparing the array value at the specified index with the value 9 tells the statement if there is a bomb at that index. This then means that my solution is more memory efficient, as the only way to get around this problem with count controlled iteration, is to iterate through an exponential amount of times, whereas my algorithm only iterates a necessary amount.

```
WHILE bombsNeeded != 0
```

```
    GridX ← Random(10)
```

```
    GridY ← Random(10)
```

```
        IF GridArr[GridX, GridY] != 9
```

```
GridArr[GridX, GridY] ← 9

bombsNeeded -= 1

bombNum = bombNum + 1

Label bombNum= bombNum

ENDIF

END WHILE
```

2.3.4 Grid Generation algorithm

This algorithm will use nested for loops to assign each index within the array an object that is instantiated from the class zone, so each index is occupied by an instance of 'Zone'. Each of these indexes are outputted in the form of a grid, size depending

```
Loop 1 to 10
  Loop 1 to 10
    If GridArr[I,J] == 9 THEN
      GridArr[I-1, J-1] += 1
    ENDIF
  EndLoop
End Loop
```

2.3.5 File saving algorithm

This algorithm stores the value of each grid into a file. This is done by taking the value of each grid space, converting it to a string, and then outputting it into a text file with the same size of the grid. For example, a bomb in grid space (6,3) would correspond to the value '9' in the text file at position (6,3).

```
Loop 1 to 11
    Loop 1 to 11
        IF FieldArr[i, j] != null
            ValueToSave ← GridArr[j, i];
            WriteToFile ValueToSave+ ","
        ENDIF
    ENDLOOP
ENDLOOP
```

2.4 Data Structures

GridArr will be a 2d array whose size is determined by the difficulty chosen. Each index is going to be instantiated with objects 'newGridSpace' which is an instance of class 'Zone'. Each index in GridArr holds a value that determines the entirety of the game. By this, I mean that the values represent the game itself. For example, GridArr[1,5] could contain the value '9' which signals a 'bomb' to be in this zone. Then GridArr[1,6] would have the value '1', because there is a '9' adjacent to the [1,5] location.

GridX = → GridY = ↓

0	1	2	3	4	5	6	7	8	9	10
1	1	1	0	1	9	1	0	0	0	0
2	9	2	1	2	1	1	1	1	1	0
3	1	2	9	1	0	0	1	9	1	0
4	0	1	1	1	1	1	2	1	1	0
5	0	0	0	0	1	9	1	0	0	0
6	1	1	1	0	1	1	1	0	0	0
7	1	9	1	0	0	0	0	1	1	1
8	1	1	1	0	1	1	1	1	9	1
9	0	0	0	0	1	9	2	2	2	1
10	0	0	0	0	1	1	2	9	1	0

FieldArr will be a 2d array which holds no true purpose to the game playing. This instead, makes sure that each and every field within the game actually has a location that is usable.

0	1	2	3	4	5	6	7	8	9	10
1	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)	(1,8)	(1,9)	(1,10)
2	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)	(2,8)	(2,9)	(2,10)
3	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)	(3,8)	(3,9)	(3,10)
4	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)	(4,8)	(4,9)	(4,10)
5	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)	(5,8)	(5,9)	(5,10)
6	(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)	(6,8)	(6,9)	(6,10)
7	(7,1)	(7,2)	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)	(7,8)	(7,9)	(7,10)
8	(8,1)	(8,2)	(8,3)	(8,4)	(8,5)	(8,6)	(8,7)	(8,8)	(8,9)	(8,10)
9	(9,1)	(9,2)	(9,3)	(9,4)	(9,5)	(9,6)	(9,7)	(9,8)	(9,9)	(9,10)
10	(10,1)	(10,2)	(10,3)	(10,4)	(10,5)	(10,6)	(10,7)	(10,8)	(10,9)	(10,10)

2.5 Human-Computer Interaction (HCI)

FrmMainForm

Difficulty:

Start game

Reveal

Place flag

How to play

Time taken:

First name

Last name:

I decided that this design would probably be best for the user, because it is very simplistic and has minimal chance for the user to find an undiscovered bug, as each button is disabled/enabled corresponding to the state of which the game is in. For example, if you have just loaded up the game, you are unable to press the reveal or place flag button until the start game button has been pressed. I have also made it so the user has to click the reveal button in order to reveal a field, as it brings in the factor of having to second guess yourself, as the actual button itself is designed to make you somewhat nervous about clicking it, as a simple click can ruin your game. This button also removes the factor of just randomly clicking on random fields and hoping you get lucky, as you would have to constantly move the cursor back and forth and isn't *really* worth it in relation to time. I have also designed the UI to be larger than usual to compensate for each difficulty taking up different sizes, so there are no overlapping issues with the buttons and the game itself. In addition to this, the reason I decided to include the time at the bottom right of the screen (along with the first name and last name boxes), is because having the timer tick right next to the game and the players names, is very ominous and increases the pressure aspect of the game, especially in the tournament scenario that my client

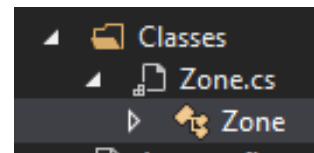
has based this project around. The Last name textbox works in a way in that when there is any input given, the start game button is enabled. This is to make sure that the program has the player's details to save when a game is won.

Section 3: Implementation

Object-oriented Programming

I have decided to use the Object Oriented Programming Paradigm (OOP) to be able to accurately identify each field within my panel grid. Although my program may contain one class, it is essential to my program, and through encapsulation, the implementation of them within my class improves my programs efficiency through having my user defined object 'newGridSpace' means that each field can be handled individually, corresponding to the state of the game and the user's inputs.

Essentially, each instance of my class will be adjusted accordingly throughout the process of Sapper being played using aggregation. I have implemented Object-Oriented Programming techniques such as encapsulation, which allow me to access specific parts of my class when instantiating an object (an object is a user defined blueprint using properties of a class). Within objects, you can have different access levels: Public, Private and Protected. Most of my properties within my object are public, with the size and width.



```
namespace Sapper
{
    public class Zone : PictureBox
    {
        public Zone()
        {
            this.Width = 50;
            this.Height = 50;
            this.BorderStyle = BorderStyle.FixedSingle;
        }

        public const int ZoneWidth = 50;
        public const int ZoneHeight = 50;
        public int GridX { get; set; }
        public int GridY { get; set; }
    }
}
```

```
public class Zone : PictureBox
{
    public Zone()
    {
        this.Width = 50;
        this.Height = 50;
```



```
        this.BorderStyle = BorderStyle.FixedSingle;
    }

    public const int ZoneWidth = 50;
    public const int ZoneHeight = 50;
    public int GridX { get; set; }
    public int GridY { get; set; }
    public bool isVisited;
}
```

Field Reveal Algorithm

The 'Field Reveal algorithm' is a custom made algorithm that is needed to actually play the game. This algorithm calculates the value that has been assigned to the corresponding grid space and then loads an image that is named in accordance to all values possible within the game. I have decided to program this algorithm because it was one of the shorter solutions and does exactly what is required from the client; so this algorithm allowed me to quickly program the base of the game because of its somewhat simplicity code wise, but was very difficult in the sense of logic. This algorithm uses two different subroutines, as there are two different procedures that must be followed logically, one for revealing all adjacent (remove zeros from neighbours), and the other procedure removes the exact square that was clicked on, as long as the value is not '0'.

```
private void btnRevealZone_Click(object sender, EventArgs e)
{
    _selectedSquare.BorderStyle = BorderStyle.Fixed3D;
}
```

```

//MessageBox.Show(Convert.ToString(GridArr[_selectedSquare.GridX,
_selectedSquare.GridY])); // shows number behind grid testing purposes
only!

        _selectedSquare.SizeMode = PictureBoxSizeMode.StretchImage;

        NumToDisplay = GridArr[_selectedSquare.GridX,
_selectedSquare.GridY].ToString();

        _selectedSquare.Load("U:\\Sapper\\Sapper\\bin\\Debug\\number-"
+ NumToDisplay + ".jpg");//Determines what image should be outputted in
reference to the number of bombs adjacent

        if (Convert.ToInt32(NumToDisplay) == 0)
        {
            RemoveZeroesFromNeighbours();

        } //Removes adjacent squares

        else if (Convert.ToInt32(NumToDisplay) == 9)
        {
            Lost();

        } //losing condition
    }
}

```

Adjacent field removal algorithm

This algorithm only ever executes when a zero is the value of the field/grid space that has been clicked. Within this algorithm, nested FOR loops are used with counters starting at -1 and finishing at 1, so the grid space that has been revealed can check all adjacent grid spaces, guaranteeing that no bombs will be located, as the initial 'zero' value shows that there are no bombs nearby. This works for my client, as one of the requirements was to have the program reveal all adjacent fields, but not all fields that are linked by a '0'. I could have also coded this recursively to remove every adjacent field, however my client strictly asked for a simple removal

algorithm; so the simplicity of this algorithm mainly comes down to my client's requests / requirements.

```
private void RemoveZeroesFromNeighbours()
{
    Zone Square = new Zone();
    for (int i = -1; i <= 1; i++)
    {
        for (int j = -1; j <= 1; j++)
        {
            Square = FieldArr[_selectedSquare.GridY + i,
            _selectedSquare.GridX + j];
            Square.SizeMode = PictureBoxSizeMode.StretchImage;
            NumToDisplay = GridArr[Square.GridX,
            Square.GridY].ToString();
            //MessageBox.Show(GridArr[Square.GridX,
            Square.GridY].ToString());

            Square.Load("C:\\Users\\andre\\Desktop\\Sapperpog\\Sapper\\bin\\Debug\\numb
            er-" + NumToDisplay + ".jpg");
        }
    } //removes a 3x3 square with the field clicked in the middle
} //
```

Minefield generation algorithm

This algorithm works using variables: bombsNeeded, GridX, GridY, GridArr[,] (all int declared variables). bombsNeeded is the value needed for the comparison in the condition. The value of bombsNeeded is set when the difficulty is chosen. Within the condition controlled loop, GridX and GridY are assigned random values from 1-9, as these are all the values possible with adjacent squares whilst playing the game; but the value '9' represents bombs, so I used selection to compare whether the index in array GridArr equals 9 or not. If the index does not have the value '9', then the index will be assigned the value '9' and therefore is now a designated "bomb" location. The bombs needed variable is then decremented, so the condition

controlled loops is one iteration closer to returning FALSE. The complexity of this algorithm is quite simple, as it only requires some iteration and comparative code, but this is essential to my project, as the values of the grid must be determined before playing the game, so this algorithm builds on the foundation of the grid being set up.

```
while (bombsNeeded != 0)
{
    GridX = rng.Next(10) + 1;
    GridY = rng.Next(10) + 1;
    if (GridArr[GridX, GridY] != 9)
    {
        GridArr[GridX, GridY] = 9; // 9 = bomb
        bombsNeeded -= 1;
        bombNum = bombNum + 1;
        lblBombNum.Text =
        Convert.ToString(bombNum);
    }
} //This While generates bombs.
```

Grid Value Calculation Algorithm

This algorithm executes at the end of every creation of a grid space, and compares all adjacent indexes within the 2D array 'GridArr' with the value 9, as a 9 represents a bomb location mentioned previously, and then increments the bombs around variable by '1', because there is 1 bomb adjacent to the current grid location. The value of bombsAround is then assigned to the index that is currently being created. This value is then further used when the player clicks on the grid and has the image corresponding to this value display. The variables BombAroundX and BombAroundY are the values that are used within the nested count controlled loop to access the indexes adjacent to the grid space that is being assigned a value. This algorithm works in a very logical way, having to access indexes of arrays using values that are incremented.

```
BombAroundX = i + 1;
BombAroundY = j + 1;
for (int X = -1; X <= 1; X++)
{
    for (int Y = -1; Y <= 1; Y++)
    {
        if (GridArr[BombAroundX, BombAroundY] != 9)
        {
            if (GridArr[BombAroundX + X, BombAroundY + Y] == 9)
            {
                bombsAround = bombsAround + 1;
                GridArr[BombAroundX, BombAroundY] = bombsAround;
            }
        }
    }
}
} //sets the numbers of bombs adjacent to the field
```

Section 4: Testing

4.1 Approach to testing


I plan to test each and every button that the user can click on each difficulty, and make sure that the fundamentals of the game works on each difficulty (Easy, Medium, Hard Tournament). My aim is to test the robustness of my project so that it meets my clients requirements and needs.

4.2 Test Plan

Test #	Description	Test Data	Expected Outcome	Actual Outcome
1	Testing that the grid size is displayed for the 'easy' difficulty	Select the easy difficulty and then press start game [Normal Test]	A 5x5 grid will be displayed	A 5x5 grid is displayed The test has passed.

<div><div>Sapper</div><div>Difficulty: Easy (5x5 Grid 5 Bombs) ▾</div><div><div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div></div></div><div><div>Start Game</div><div>Reveal</div><div>Place Flag</div><div>How To Play</div><div>Time taken: 2 seconds</div><div>First Name: Andrew</div><div>Last Name: Smith</div></div></div>				
2	Testing that the grid size is displayed correctly when the medium difficulty is clicked.	Select the medium difficulty and then click the start game button	The correct grid size will be displayed (7x7)	A 7x7 grid was displayed. Test passed

The screenshot shows the Sapper game window. At the top, the title bar says 'Sapper' with standard window controls. Below the title bar, there's a 'Difficulty:' label followed by a dropdown menu showing 'Medium (7x7 Grid 7 Bombs)'. To the left of the buttons is a 7x7 grid of squares. To the right of the grid are four buttons: 'Start Game' (highlighted with a red dashed border), 'Reveal', 'Place Flag', and 'How To Play'. Below these buttons, there's a 'Time taken: 1 seconds' display. At the bottom, there are two text input fields: 'First Name:' with 'Andrew' entered, and 'Last Name:' with 'Smith' entered.

 Sapper

Difficulty: Hard (Tournament Mode: 10x10 Grid 10 bombs)

Start Game

Reveal

Place Flag

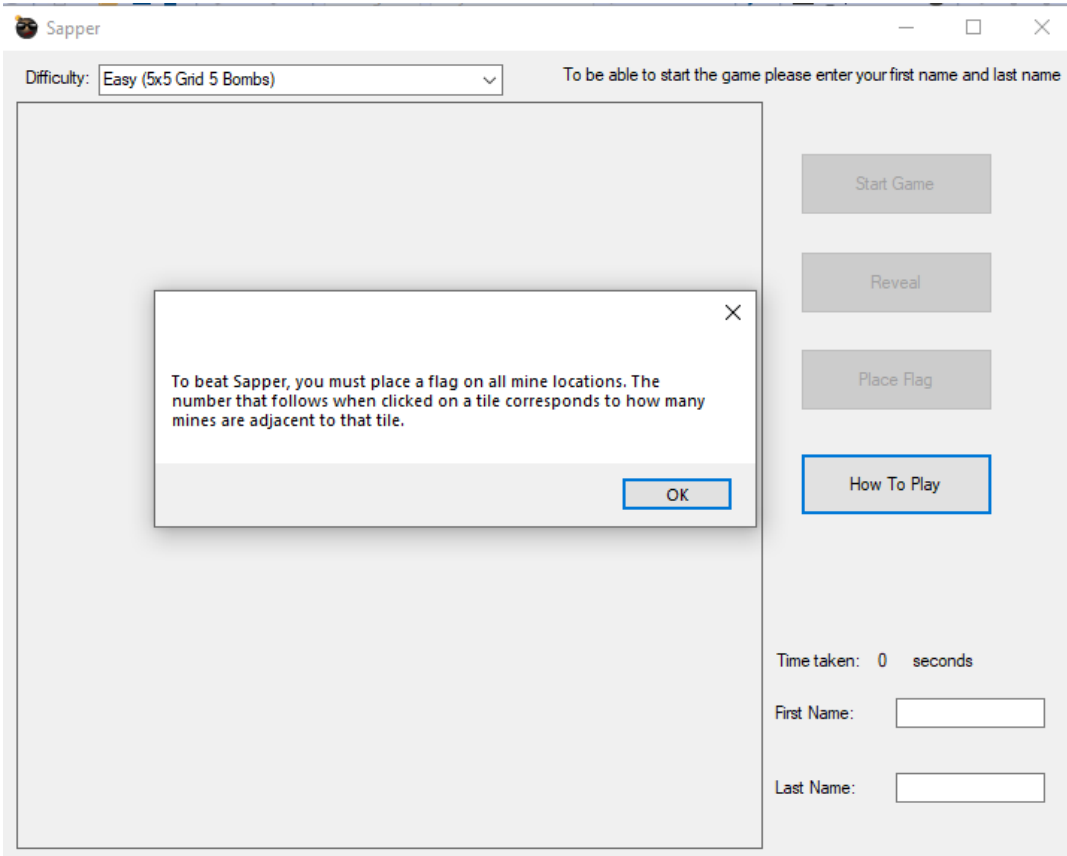
How To Play

Time taken: 17 seconds


First Name:

Last Name:

4	Test that the timer ticks correctly	Press start game button and take two samples ten seconds apart	The timer will tick at correct intervals.	The timer ticked correctly. Test passed (shown in video)
5	I will test that the game plays correctly and displays the correct amount of bombs	I will play the game as intended on the 'hard' difficulty and count the amount of bombs that have been displayed and assess whether it is correct	I expect the game to play correctly and for 10 bombs.	The game displayed correctly and displayed 10 total bombs, showing the tournament difficulty to have been the only initial working difficulty. Test passed (shown in video)

6	Test whether the 'How To Play' button works when the "easy" difficulty is selected	I will click on the "how to play" button after selecting the easy difficulty	A messagebox will show with the instructions	How to play instructions were displayed in a messagebox Test passed
				
7	Test that the reveal button does not work when there is no field selected on the easy difficulty	Run the program and attempt to press the reveal button without selecting a field on the easy difficulty	Will not work as the button has been disabled within the code and is only enabled when a field has been clicked	The button was unable to be pressed Test passed

<div><div>Sapper</div><div>Difficulty: Easy (5x5 Grid 5 Bombs) ▾</div><div><div><div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div></div></div><div><div>Start Game</div><div>Reveal</div><div>Place Flag</div><div>How To Play</div></div><div>Time taken: 2 seconds</div><div>First Name: <input type="text" value="Andrew"/></div><div>Last Name: <input type="text" value="Smith"/></div></div></div>				
8	Test that the reveal button works when a field is selected on the easy difficulty	I will attempt to press the reveal button when a field is selected on the easy difficulty	A value will be shown or a bomb will be shown.	The value '0' was shown. Test passed

 Sapper

Difficulty: Easy (5x5 Grid 5 Bombs)

			2	

Start Game

Reveal

Place Flag

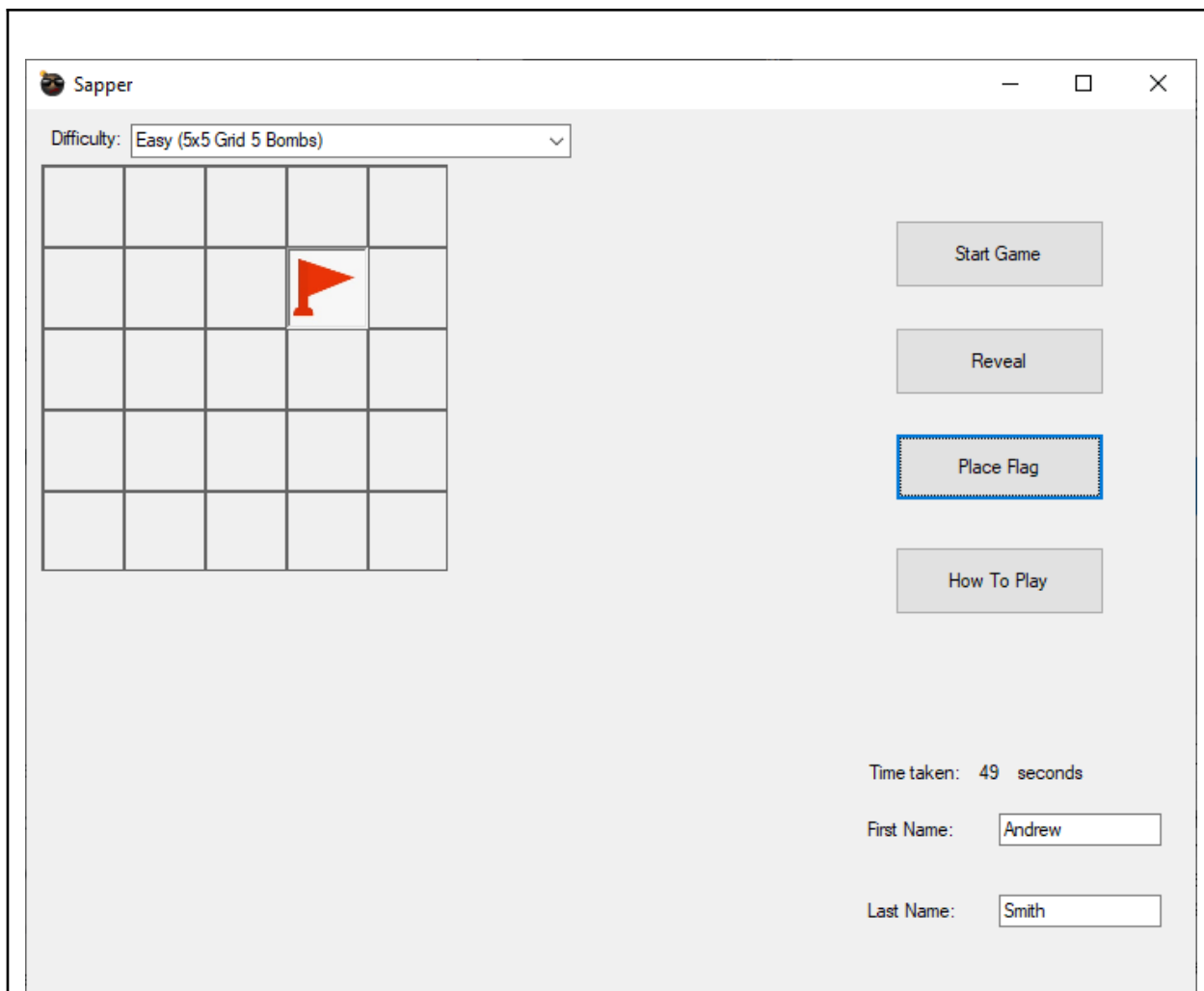
How To Play

Time taken: 14 seconds


First Name:

Last Name:

9	Test whether the place flag button can't be pressed when there is no field selected on the easy difficulty	I will start the game and then try to press the place flag button on the easy difficulty	It should not allow me to press the place flag button.	It did not let me press the place flag button without selecting a field Test passed
10	Test whether the game can place a flag when a field is selected on the easy difficulty	I will start the game and then place a flag on a random field on the easy difficulty	I expect the button to work, and a flag to appear on the field i clicked	A flag was placed at the field that was selected Test passed



11	Test whether the program can still reveal values where a flag already is when on the easy difficulty	Place a flag on a random field and then reveal it whilst in the easy difficulty	The flag will be removed and the value will be shown.	The value was shown correctly, and the flag image was replaced by the value image Test Passed
12	Test that the how to play button works when the medium difficulty is selected	Select the medium difficulty, start game and then press the how to play button	The how to play messagebox will show up	How to play messagebox shown up Test passed

 Sapper

Difficulty: Medium (7x7 Grid 7 Bombs)

Start Game

Reveal

Place Flag

How To Play

Time taken: 5 seconds

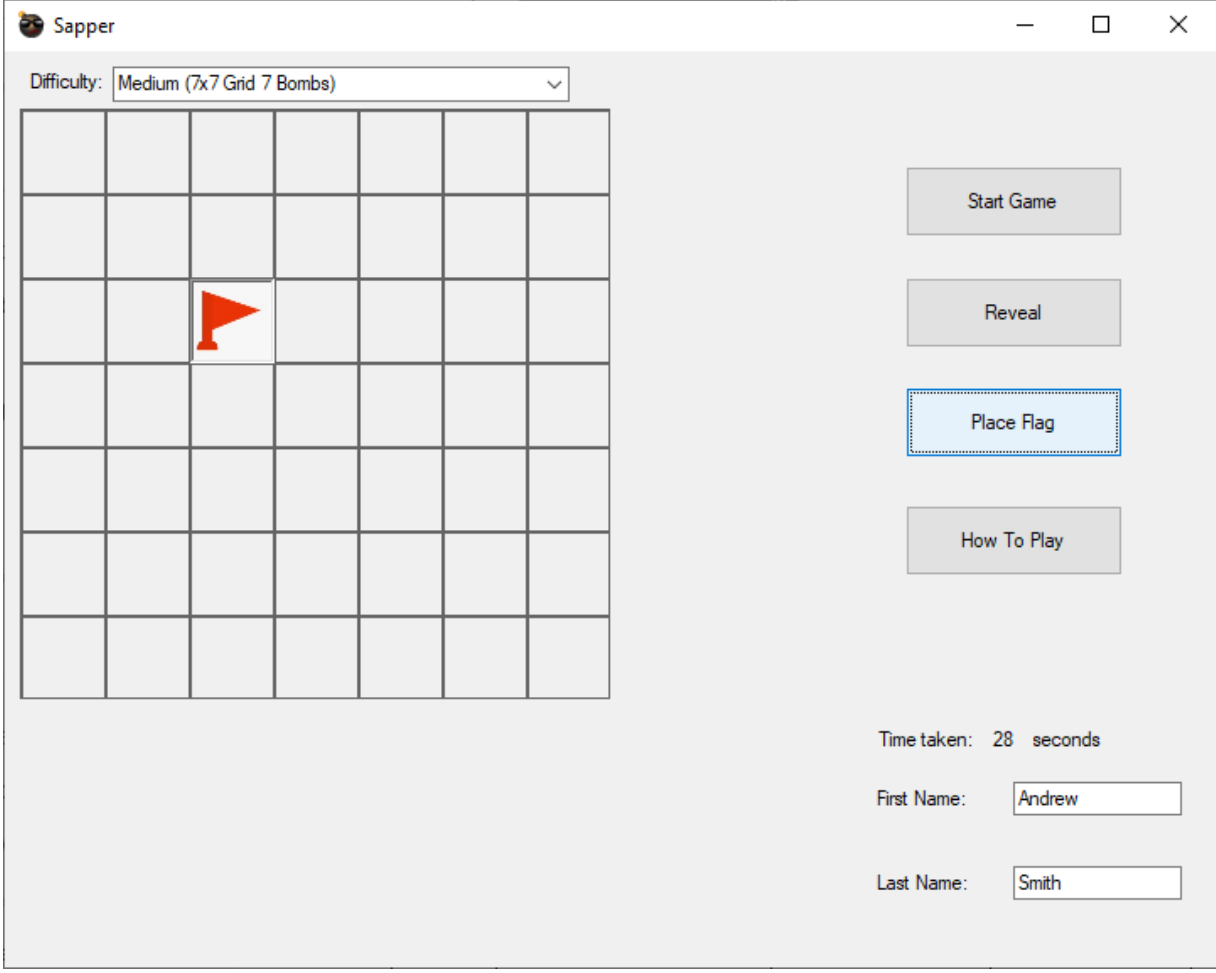
First Name:

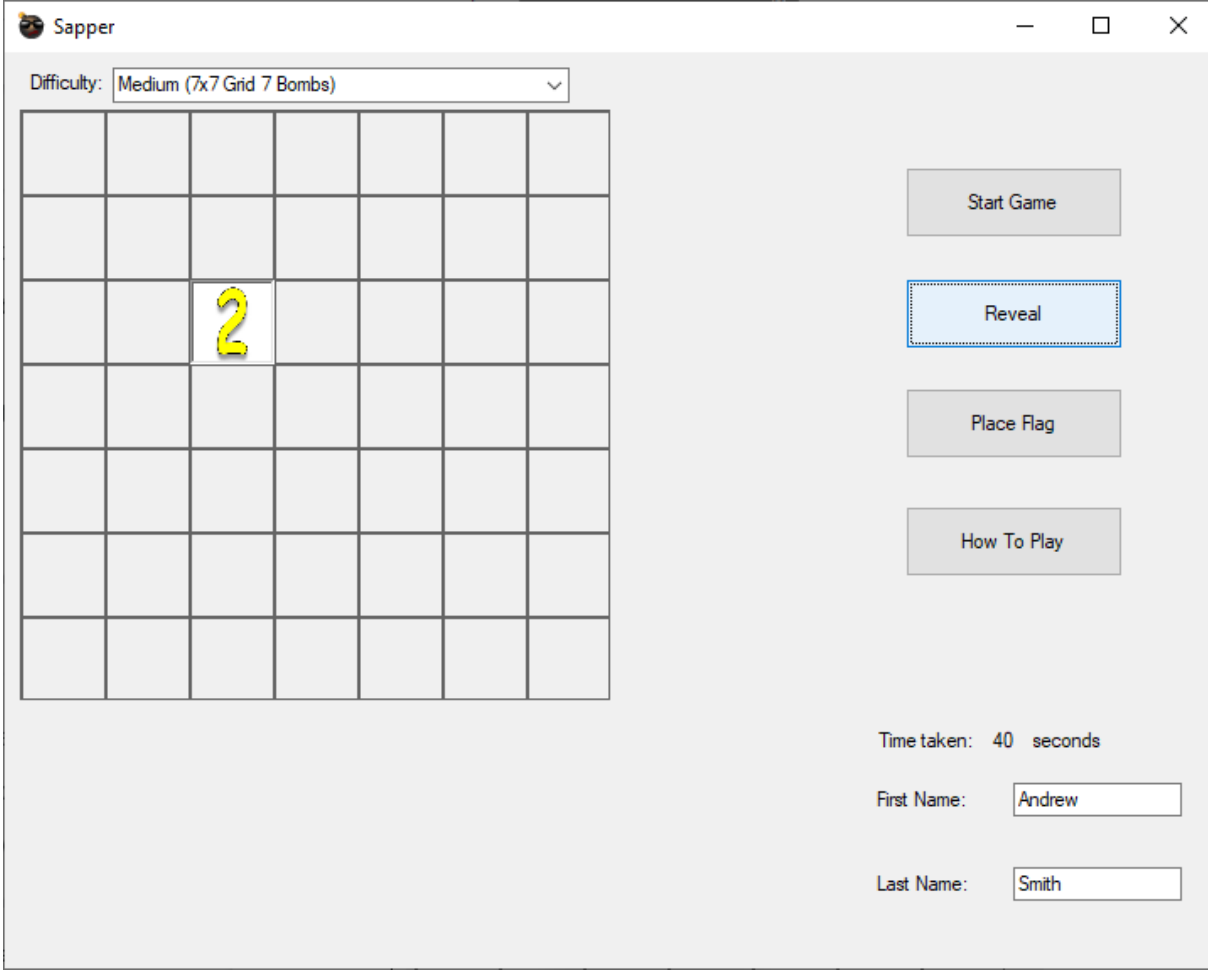
Last Name:

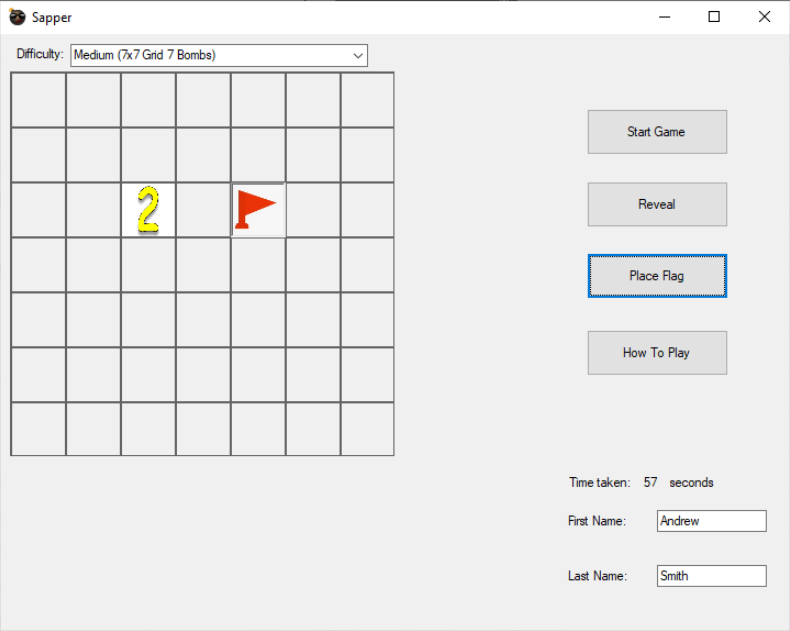
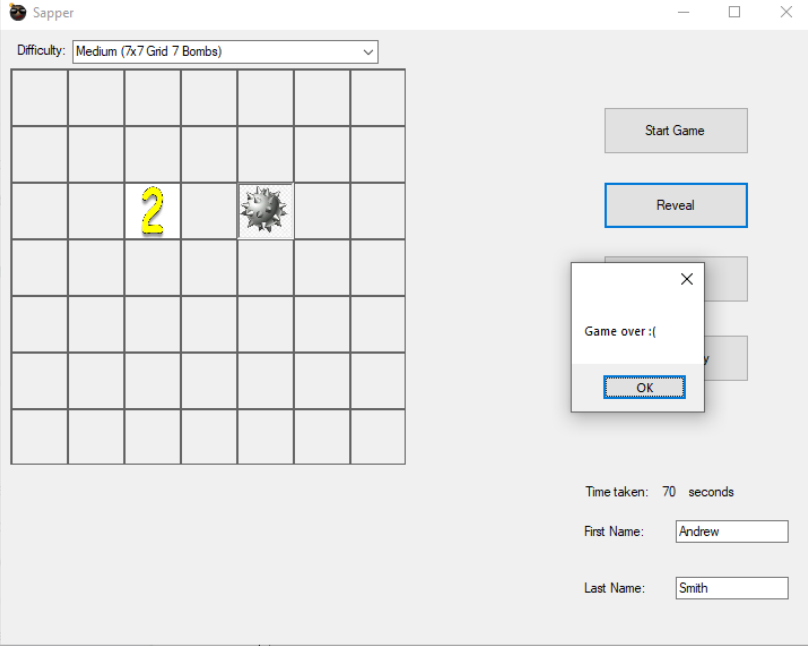
To beat Sapper, you must place a flag on all mine locations. The number that follows when clicked on a tile corresponds to how many mines are adjacent to that tile.

OK

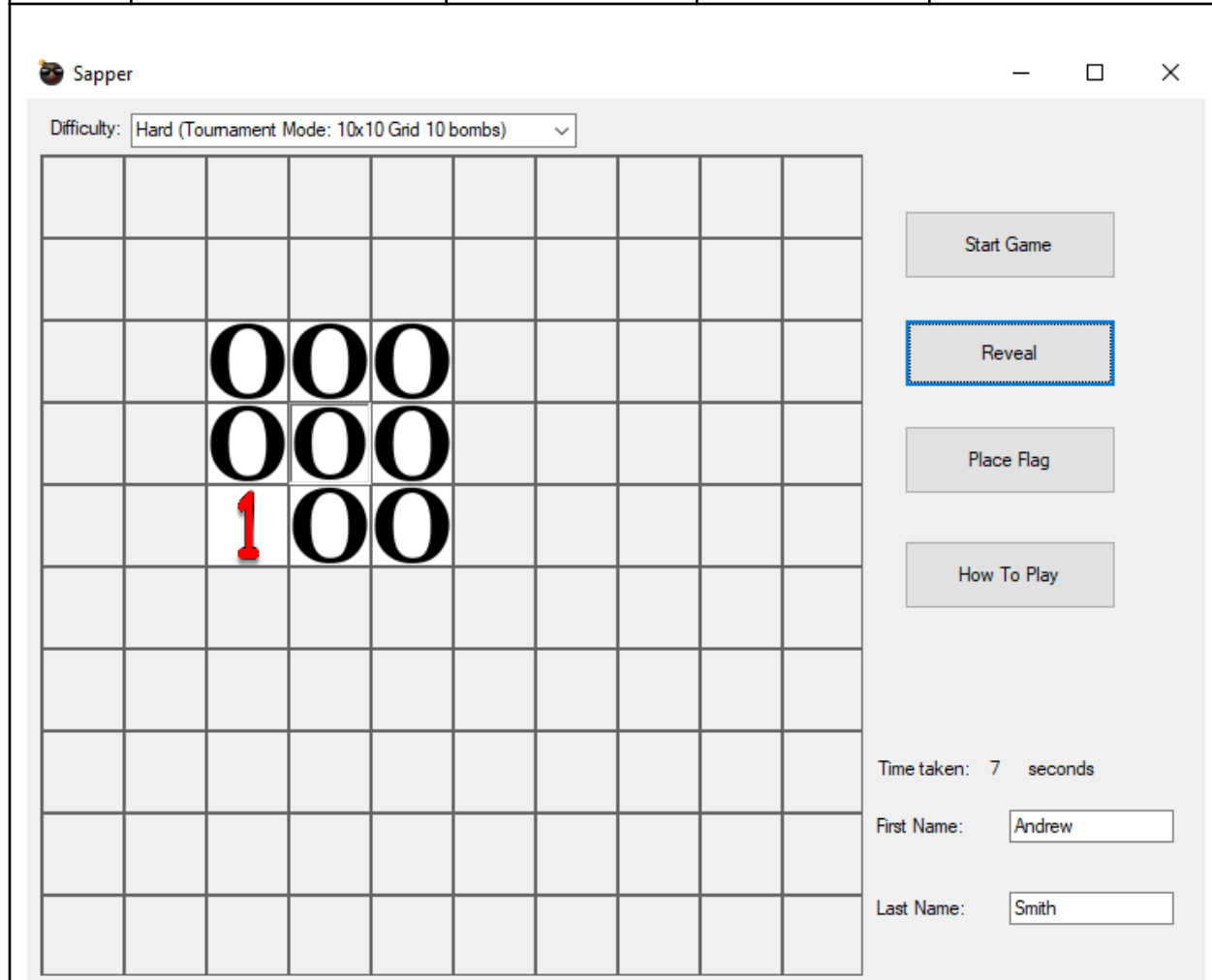
13	Test that the reveal button does not work when there is no field selected on the medium difficulty	Start a medium difficulty game and attempt to press the reveal button without selecting a field	The button will not work	The button did not work Test passed
14	Test that the place flag button does not work when there is no field selected on the medium difficulty	Start a medium difficulty game and attempt to press the place flag button when there is no field selected	The place flag button will not work	The place flag button did not work Test passed

15	Test that the place flag button works when there is a field selected on the medium difficulty	Start a medium difficulty game and attempt to place a flag on a selected field	The field will have a flag placed on it	There was a flag placed on the field Test passed
				
16	Test that the reveal button works when there is a field selected on the medium difficulty	Start a medium difficulty game and attempt to press the reveal button	The field selected will be revealed	The field selected was revealed Test passed

				
17	Test that the reveal button works when you reveal the field that has a flag on it already on the medium difficulty	Start a medium difficulty game and attempt to reveal a field that already has a flag on it.	The field will reveal, even if there is a flag on the field	The field that had a flag on it and was selected was revealed Test passed

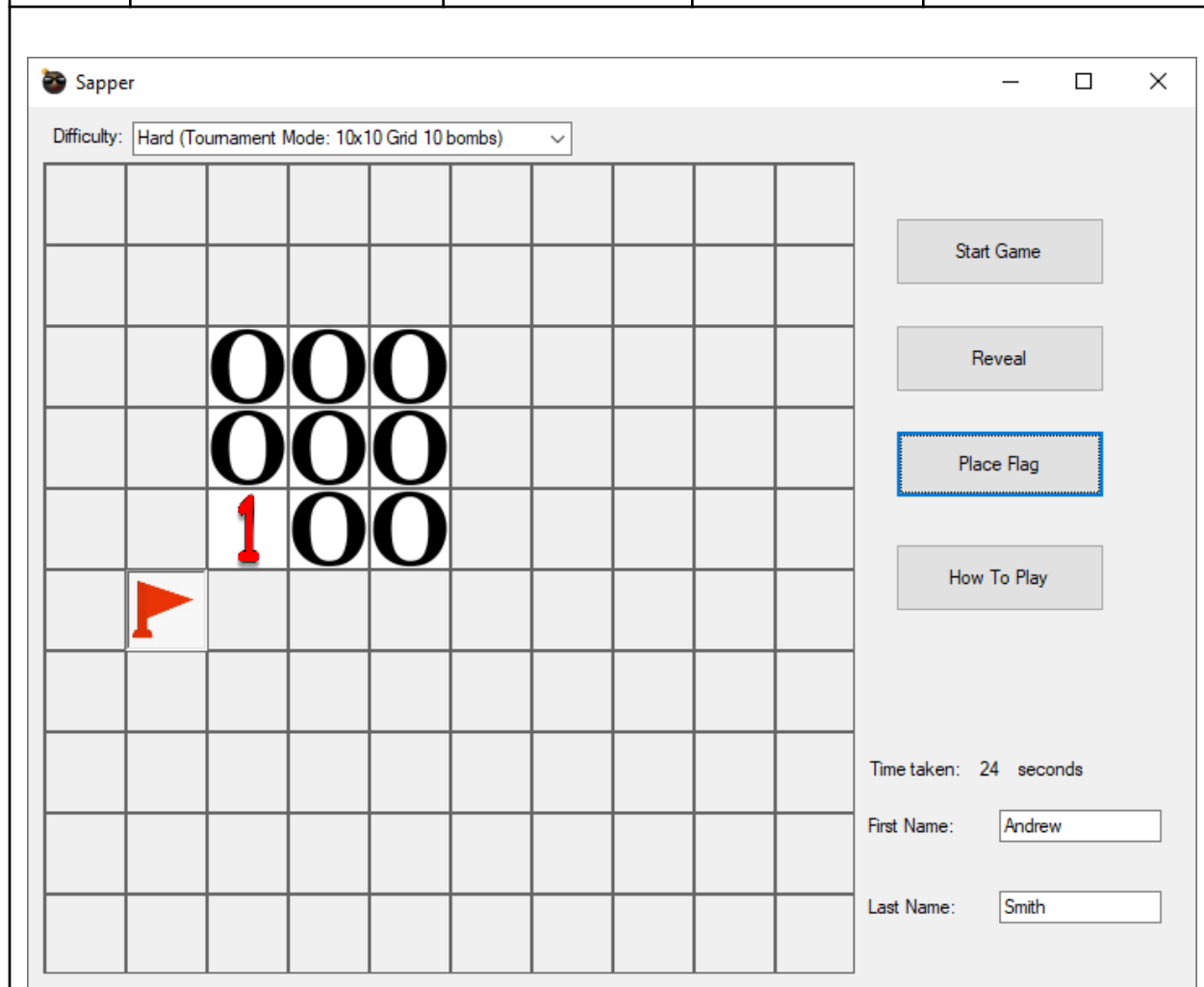
				
				
18	Test that the how to play button works when the hard difficulty is selected	Start a had difficulty game and then attempt to press the how to play button	The button will not work	The button did not work Test passed

19	Test that the reveal button does not work when there is no field selected on the hard difficulty	Start a hard difficulty game and attempt to press the reveal button without selecting a field	The button will not work	The button did not work Test passed
20	Test that the reveal button does work when there is a field selected on the hard difficulty	Start a hard difficulty game and attempt to press the reveal button when a field is selected	The field selected will be revealed	The field selected was able to be revealed Test passed



21	Test that the place flag button does not work when there is no field	Start a hard difficulty game and attempt to	The place flag button will not work	The button did not work
----	--	---	-------------------------------------	-------------------------

	selected on the hard difficulty	press the place flag button without		Test passed
22	Test that the place flag button does work when there is a field selected on the hard difficulty	Start a hard difficulty game and attempt to press the press the place flag button with a field selected	A flag will be placed on the field selected	A flag was placed on the field selected Test passed



23	Test that the reveal button works on a field that already has a flag on it on the hard difficulty	Start a hard difficulty game, reveal a field and then attempt to	The field will reveal the value underneath the flag	The field revealed the value under the flag when the reveal button was pressed
----	---	--	---	--

	to be revealed on the easy difficulty	and then reveal the top left field		Test passed
<div><div>Sapper</div><div>Difficulty: Easy (5x5 Grid 5 Bombs)</div><div><div>1</div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div>Start Game</div><div>Reveal</div><div>Place Flag</div><div>How To Play</div></div><div><div>Time taken: 2 seconds</div><div>First Name: Andrew</div><div>Last Name: Smith</div></div></div>				
25	Test that the most top right index is able to be revealed correctly on the easy difficulty	Start an easy difficulty game and then reveal the top right field	The top right field will be revealed	<div>The top right field was revealed</div> <div>Test passed</div>

<div><div>Sapper</div><div>Difficulty: Easy (5x5 Grid 5 Bombs) ▾</div><div><div><div><div>1</div></div></div></div><div><div>Start Game</div><div>Reveal</div><div>Place Flag</div><div>How To Play</div></div><div>Time taken: 3 seconds</div><div>First Name: <input type="text" value="Andrew"/></div><div>Last Name: <input type="text" value="Smith"/></div></div> <td>26</td> <td>Test that the most bottom left index is able to be revealed on the easy difficulty</td> <td>Start an easy game and then reveal the bottom left field</td> <td>The bottom left field will be revealed</td> <td>The bottom left field was revealed Test passed</td>	26	Test that the most bottom left index is able to be revealed on the easy difficulty	Start an easy game and then reveal the bottom left field	The bottom left field will be revealed	The bottom left field was revealed Test passed
---	----	--	--	--	---

<div><div>Sapper</div><div>Difficulty: Easy (5x5 Grid 5 Bombs) ▾</div><div><div><div>2</div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div></div></div><div><div>Start Game</div><div>Reveal</div><div>Place Flag</div><div>How To Play</div></div><div><div>Time taken: 3 seconds</div><div>First Name: Andrew</div><div>Last Name: Smith</div></div></div>				
27	Test that the most bottom right index is able to be revealed on the easy difficulty	Start an easy game and then reveal the bottom right field	The bottom right field will be revealed	The bottom right field was revealed Test passed

<div><div>Sapper</div><div>Difficulty: Easy (5x5 Grid 5 Bombs) ▾</div><div><div><div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div>2</div></div></div></div><div><div>Start Game</div><div>Reveal</div><div>Place Flag</div><div>How To Play</div></div><div><div>Time taken: 2 seconds</div><div>First Name: <input type="text" value="Andrew"/></div><div>Last Name: <input type="text" value="Smith"/></div></div></div>				
28	Test that the most top left index is able to be revealed on the medium difficulty	Start a medium game and then reveal the top left field	The top left field will be revealed	The top left field was revealed Test passed

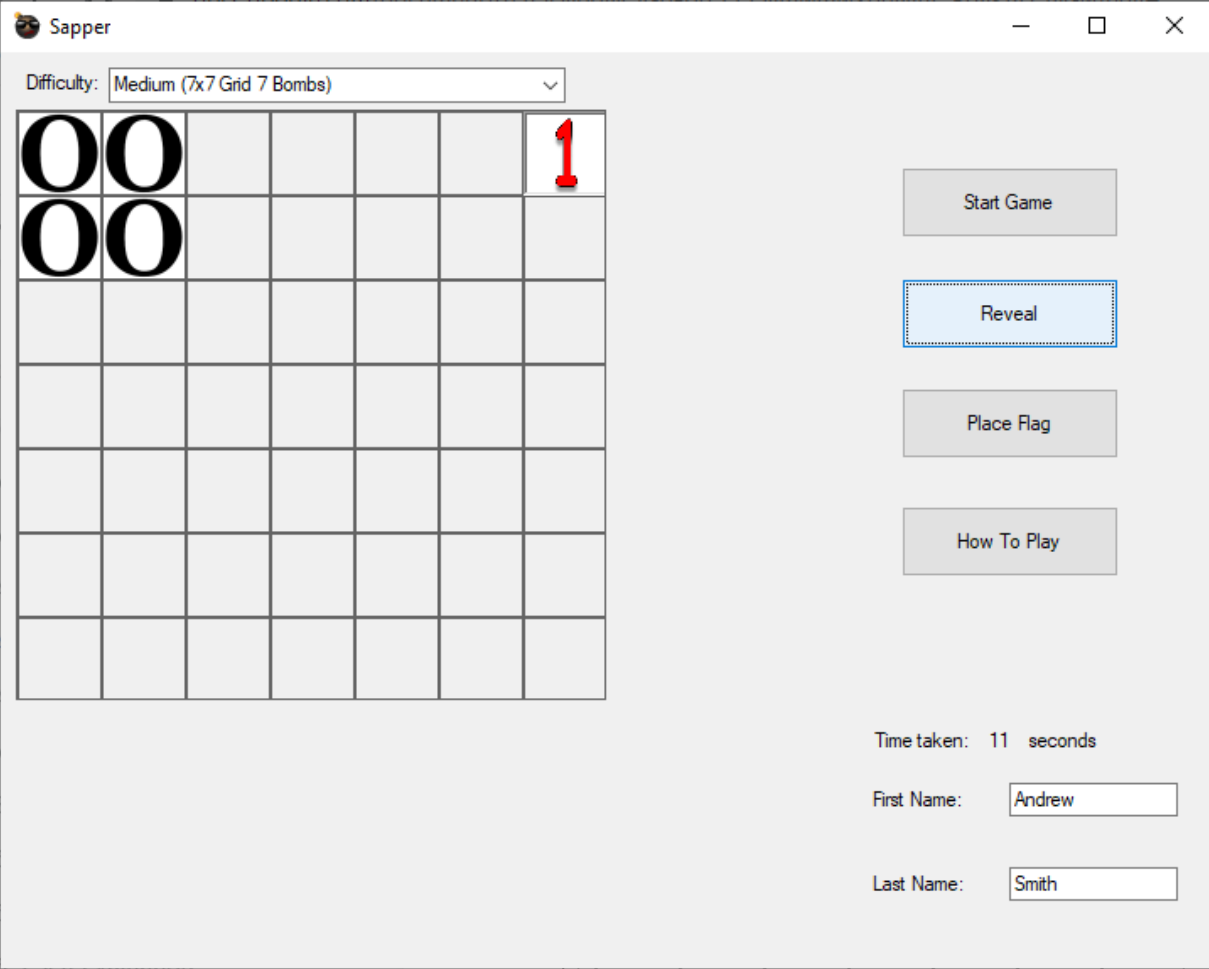
29

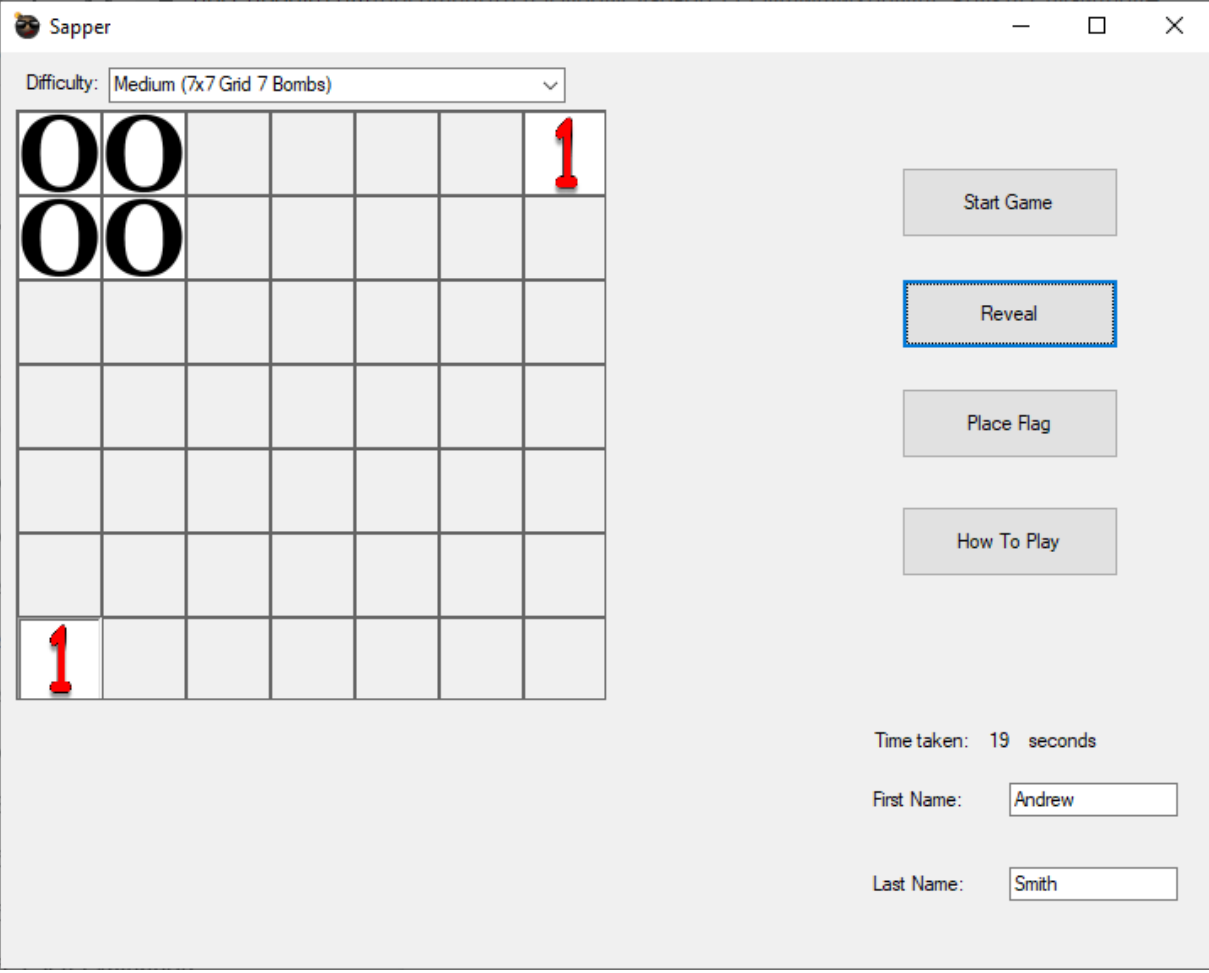
Test that the most top right index is able to be revealed on the medium difficulty

Start a medium game and then reveal the top right index

The top right field will be revealed

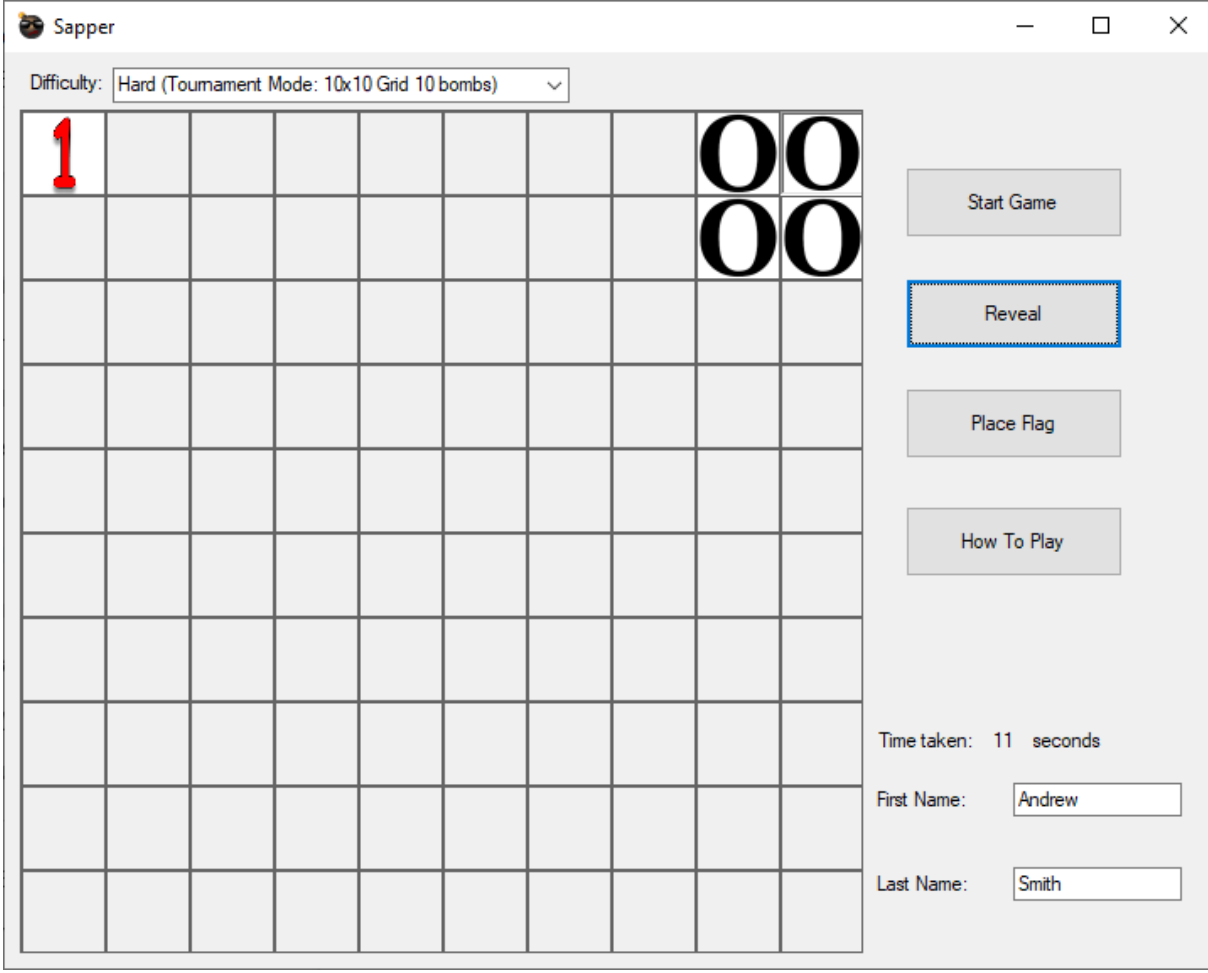
The top right field was revealed
Test passed

					
30	Test that the most bottom left index is able to be revealed on the medium difficulty	Start a medium difficulty game and reveal the bottom left field	The bottom left field will be revealed	The top right field was revealed	Test passed

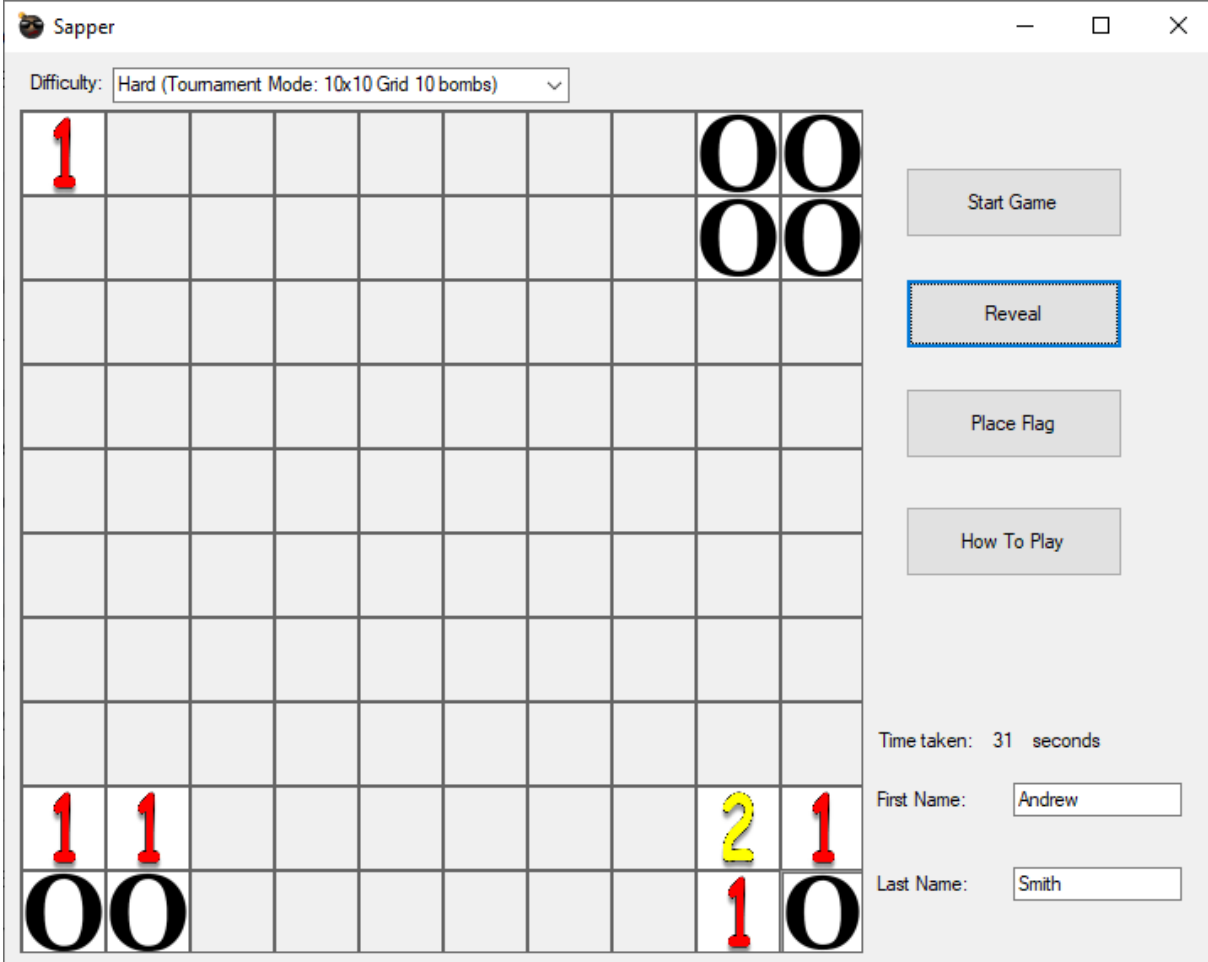
					31	Test that the most bottom right index is able to be revealed on the medium difficulty	Start a medium difficulty game and reveal the bottom right field	The bottom right field will be revealed	The bottom right field was revealed Test passed
---	--	--	--	--	----	---	--	---	--

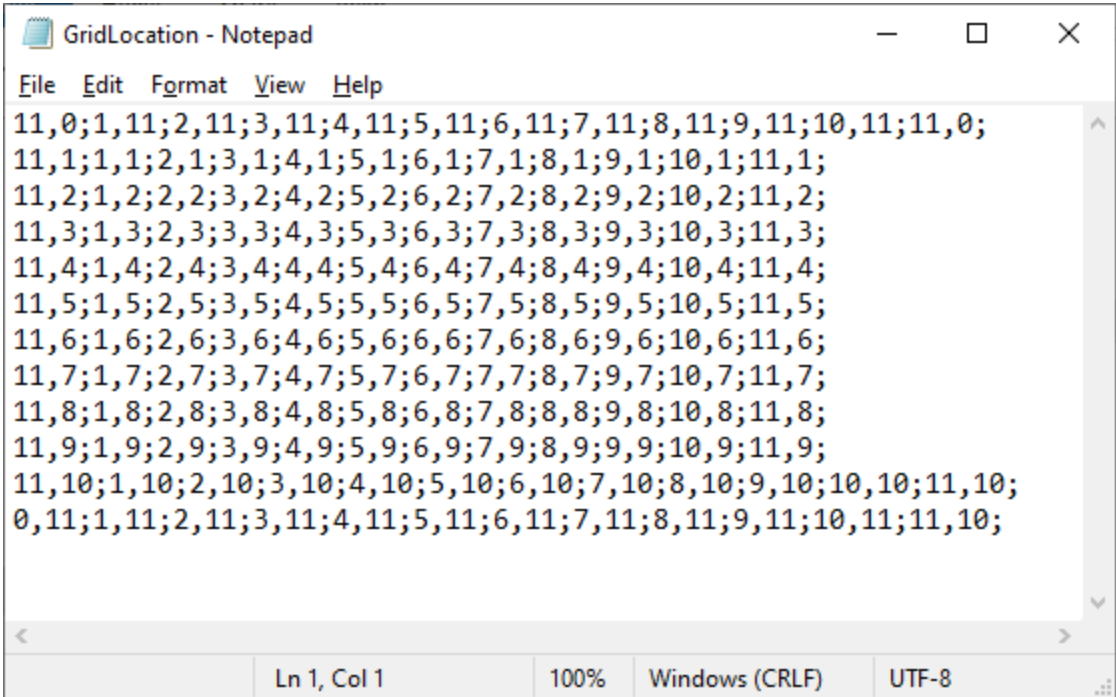
<div><div>Sapper</div><div>Difficulty: Medium (7x7 Grid 7 Bombs)</div><div><div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div></div><div><div>Start Game</div><div>Reveal</div><div>Place Flag</div><div>How To Play</div></div><div><div>Game over :(</div><div>OK</div></div><div><div>Time taken: 8 seconds</div><div>First Name: Andrew</div><div>Last Name: Smith</div></div></div></div>				
32	Test that the most top left index is able to be revealed on the hard difficulty	Start a hard difficulty game and reveal the top left field	The top left field will be revealed	<div>The top left field was revealed</div> <div>Test passed</div>

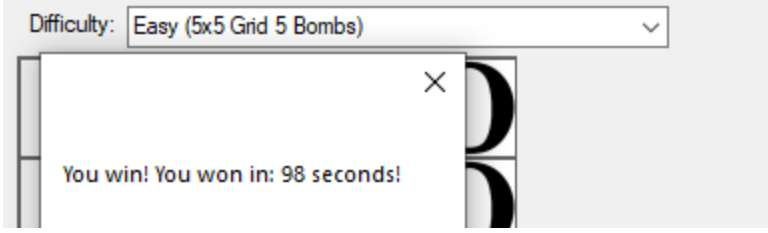
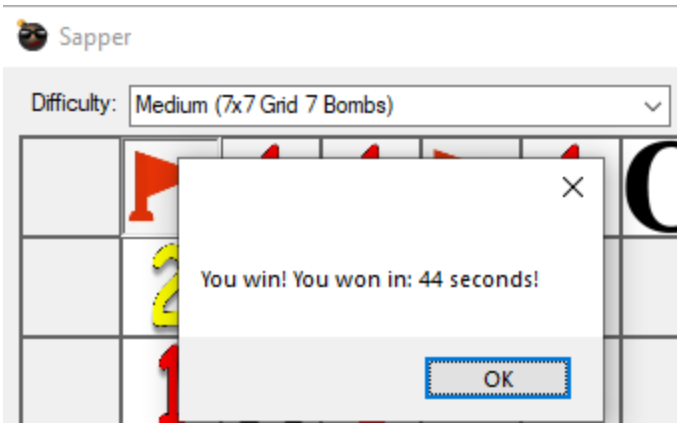
<div><div>Sapper</div><div>Difficulty: Hard (Tournament Mode: 10x10 Grid 10 bombs) ▾</div><div><div><div>1</div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div><div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div></div><div><div>Start Game</div><div>Reveal</div><div>Place Flag</div><div>How To Play</div><div>Time taken: 5 seconds</div><div>First Name: <input type="text" value="Andrew"/></div><div>Last Name: <input type="text" value="Smith"/></div></div></div>				
33	Test that the most top right index is able to be revealed on the hard difficulty	Start a hard difficulty game and reveal the top right index	The top right field will be revealed	The top right field was revealed Test passed

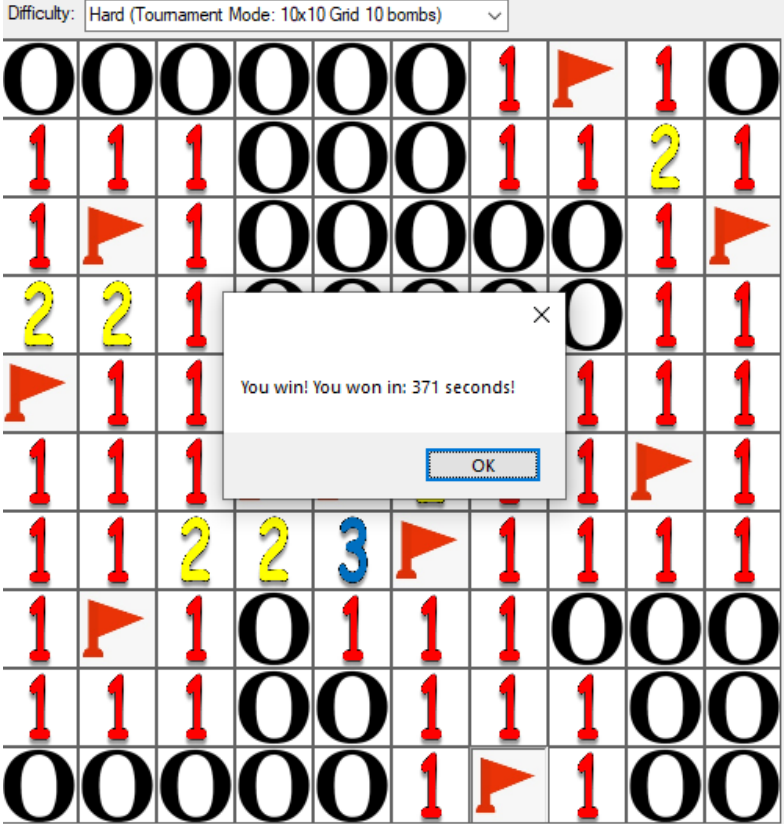
									
34	Test that the most bottom left index is able to be revealed on the hard difficulty	Start a hard difficulty game and reveal the bottom left field	The bottom left field will be revealed	The bottom left field was revealed	Test passed				

<div> <div>Sapper</div> <div>Difficulty: Hard (Tournament Mode: 10x10 Grid 10 bombs) ▾</div> <div> <div> <div>1</div> <div>00</div> <div>00</div> </div> <div> <div>1</div> <div>1</div> <div>00</div> </div> </div> <div> <div>Start Game</div> <div>Reveal</div> <div>Place Flag</div> <div>How To Play</div> <div>Time taken: 24 seconds</div> <div>First Name: <input type="text" value="Andrew"/></div> <div>Last Name: <input type="text" value="Smith"/></div> </div> </div>									
35	Test that the most bottom right index is able to be revealed on the hard difficulty	Start a hard difficulty	The bottom right field will be revealed	The bottom right field was revealed					

				
36	Test that the gridcontents file displays correctly.	Start a game and then load the grid contents file	Should display a 13x13 grid with the values of the game. The “outer” coordinates should all equal zero as they are hardcoded placeholders which allow the removeZeroesFromNeighbour subroutine to execute robustly.	<p>The file did display the correct values in correspondence to the game</p> <p>Test passed</p>

37	Test that the gridlocation file displays correctly.	Start a game then load the grid location file	Should display a list of each index corresponding to "coordinates" with the outer coordinates being placeholder hard coded values	The file did display each location in correspondence to their coordinates Test passed
				
38	Test that the winning condition works on the easy difficulty	Start an easy difficulty game and win	Well done message will pop up	The well done message box popped up Test passed

				
39	Test that the winning condition works on the medium difficulty	Start a medium difficulty game and win	Well done message will pop up	The well done message box popped up Test passed
				
40	test that the winning condition works on the hard difficulty	Test a hard difficulty game and win	Well done message will pop up	The well done message box popped up Test passed

				
41	Test that the losing condition works on the easy difficulty	Start an easy difficulty game and lose	The frmYouLost will load	The frmYouLost was loaded and displayed Test passed

<div> <div>Difficulty: Easy (5x5 Grid 5 Bombs) 256</div> <div> <div>! KABOOM!!</div>  </div> </div>				
42	Test that the losing condition works on the medium difficulty	Start a medium difficulty game and lose	The frmYouLost will load	The frmYouLost was loaded and displayed Test passed

 <p>The screenshot shows a Minesweeper game window. At the top, a dropdown menu indicates the difficulty is 'Medium (7x7 Grid 7 Bombs)'. Below the menu, a red circle with an exclamation mark is followed by the text 'KABOOM!!'. The main area of the window is filled with a large, bright orange and yellow nuclear explosion graphic against a dark background.</p>				
43	Test that the losing condition works on the hard difficulty	Start a hard difficulty game and lose	The frmYouLost will load	The frmYouLost was loaded and displayed Test passed

<div> <div>Difficulty: Hard (Tournament Mode: 10x10 Grid 10 bombs) ▾</div> <div> <div>KABOOM!!</div> <div>  </div> </div> </div>				
44	Test that the program runs and starts up without crashing	Execute my program	The program will start up correctly	As expected Test passed

Sapper

Difficulty: Easy (5x5 Grid 5 Bombs) ▼

To be able to start the game please enter your first name and last name

Start Game

Reveal

Place Flag

How To Play

Time taken: 0 seconds

First Name:

Last Name:

45	Test that the first name text box allows for a string to be inputted	Input the string "test" in the first name box	The string "test" should display in the first name text box	As expected. Test passed
46	Test that the Last Name textbox allows for a string to be inputted	Input the string "test" in the last name box	The string "test" will display in the last name box	As expected Test passed

<div> <div>First Name: <input type="text"/></div> <div>Last Name: <input type="text" value="Test"/></div> </div>				
47	Test that the remove zeroes from neighbours algorithm is executed when a zero is clicked	Start a hard difficulty game and remove a 'zero'	A 3x3 grid will display separately, with the field selected in the centre being '0'	The 3x3 grid was displayed with the centre selected field being '0'

Sapper

Difficulty: Hard (Tournament Mode: 10x10 Grid 10 bombs)

					2				
		0	0	1					
		0	0	1					
		0	1	1					

Start Game

Reveal

Place Flag

How To Play

Time taken: 11 seconds

First Name:

Last Name:

4.3 Requirement Testing

Objective	Which tests cover this requirement?
1	9,14,15,21,22
2	44
3	38,39,40
4	In video
5	In video
6	36
7	16, 17, 20, 23
8	45,46
9	1, 2, 3, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 38, 39, 40, 41, 42, 43
10	47

Section 5: Evaluation

Summary

Personally I feel that the development of my project has been somewhat successful in the fact that I have a near-perfect solution. By this, I mean that I have been able to meet most of my client requirements and the requirements that haven't exactly been met would only cause problems if my client was to redistribute my project to other businesses or societies.

5.1 Evaluation of the objectives

Objective 1: The system must allow players to mark specific fields as flags (bomb spots)

This client requirement has indeed been met. I initially thought about having a right click event to trigger the flag placing, however after further research and discussion with my client, a button was created that would take an image from the bin/debug folder and display said image on the field that is selected. Whilst this may increase the activity and volume of user input, this solution was much simpler and took less time to implement than it would have been to research how to include an exclusive right mouse click.

Objective 2: The system must be playable on an ordinary computer device (laptop, desktop) at home.

This requirement has been met. I have made this program using the C#.net framework, meaning that this program comes with an executable file that can be executed on any computer device. This objective works perfectly for all ordinary computer devices, as every personal machine has the ability to execute C# code.

Objective 3: The system must allow the user to complete the game (meet the winning condition).

This requirement has been met. The program has a winning condition that executes once all bomb locations have had flags placed on them. This is extremely important to my program, as a game cannot be 'whole' unless there is a winning condition. This was programmed onto a click event of a button as per request from my client. This then ultimately fulfils the client objective as it enables the game to be completed by reaching the winning condition of having all bombs have flags placed on top of them.

Objective 4: The system must use minimal memory

This requirement has been met. This program only uses memory when loading images, with each image only taking up 10kb of storage; so worst case scenario, a full game on the tournament difficulty would use 1mb of memory. This means that my program can run efficiently and have no issues with memory leaks. This meets the requirement because there isn't more I can do to reduce the memory usage in terms of displaying pictures, in a tangible amount of time.

Objective 5: The system must be able to run multiple times within the same instance

This requirement has been met. I have programmed this project with the intention having the game being replayable multiple times, so I basically coded the frameworks of the game on the click event of the "start game" button (btnStartGame), so each time the button is pressed, a new game is initiated, with whole new bomb locations. This essentially works as a sort of 'restart' button if you aren't happy with the time you are completing the game in. However, if you are to lose the game by clicking on a bomb, then you are forced to shut the program down and start again, as a sort of added measure of stress which would make the user think harder before randomly guessing. This meets the requirement as the way it is programmed ensures that the game can be replayed as many times as the user would like.

Objective 6: The system must have the ability to save the state of the game

This requirement has been met. The program saves the state of the game as soon as the 'Start Game Button' is pressed. This is saved in a file called "GridContents.txt", because it contains the values of each grid space. I have programmed this algorithm to go through each index in the GridArr array using a nested for loop, and writes the value of that index to the file, and is then separated using a hard coded "," to make the file look more appealing to the eye. This then meets the requirement, as each time a game is started, the values of each grid field are saved to a file, for access at any given time.

Objective 7: The system must allow the player to reveal a field

This requirement has been met. This is very essential to the program, as you can't actually play the game without revealing a field and being extremely lucky to blindly identify bomb spots. This was programmed behind a button labelled "reveal". This button takes the value of the field selected, and loads a picture with the associated number. Another way to do this would be just to have it on a click event, however my client clearly requested that this feature was to be applied to a button; so this meets the requirement, as I have taken into consideration the request of my client.

Objective 8: The system must allow the user to save their name

This requirement has been met. This requirement was programmed using the system.IO library, so that I could use a file handling algorithm to take the input from a textbox and write said input to a .txt file along with the score (time taken) and the difficulty that the game was played on. This is fine for my client, as they need the scores to be saved for their tournament ONLY, however

this requirement would possibly fall down if it was to be published to the public, due to the simplicity of using a .txt file rather than a database. Regardless, this requirement has been met in regards to how my client wants this to be completed.

Objective 9: The system must allow the user to adjust difficulty

This requirement has been met. This program works in a way that changes the amount of values that are inserted into the array, as well as changing the amount of iterations that the algorithm takes to create the game. My client wanted this, because whilst their intention of the project may be that of a tournament software, my client would still like to display the game in their facility, so adding a system that allows the user to play the game on different difficulties allows for a more adhering nature to the program. This requirement has been completed for the best way for my client, as the program plays the exact same, just with different grid sizes with lower amounts of bombs.

Objective 10: The system must remove adjacent squares

This requirement has been met. This algorithm only ever executes when the field that has been revealed is a zero. If the field revealed is a zero, the subroutine to remove all adjacent squares is called, and then a nested for loop is used from -1 to 1 to check the 3x3 grid of squares that would be adjacent. This comes as an increase to the quality of life aspect of the game, as if you were to reveal a zero without this algorithm, then it would impact the time taken, removing / lessening the skill aspect of the game. This suited my client, as they feel that having this algorithm instead of another algorithm that would remove ALL zeros would be too easy for the user, removing the 'hard' difficulty of the tournament; so i decided that this was not necessary for the lower difficulties, as the smaller grids would make the remove adjacent zeroes almost irrelevant. Therefore, this requirement has been met to the satisfaction of my client, as the game has been coded deliberately to make the usual game of "minesweeper" harder.

5.2 Client Feedback & Review

Once Sapper had been developed to the standard I liked, I set up an interview with my client, which you can see in [Appendix B: Client Feedback](#).

Requirement Number	Satisfaction level
(1)	Highly satisfied
(2)	Highly satisfied
(3)	Highly satisfied
(4)	Highly satisfied
(5)	Highly satisfied
(6)	Satisfied
(7)	Highly satisfied
(8)	Satisfied
(9)	Highly satisfied
(10)	Highly satisfied

My client overall was very happy with the program that I had produced for him, stating that it “met all of my expectations”, to which he also noted that visually it looked like I have captured exactly what minesweeper looked like, stating it had a “retro” look. Adam Bailey also reciprocated the fact that Sapper looks to be challenging enough for the tournament that Sapper was requested for, meaning that the difficulty that I had to set myself has been successful. However, when I asked Adam about whether there should be more features added, he stated that I “can do it for my own personal gain”.

One key aspect that my client noted was the fact that when a zero is revealed, **only** the adjacent fields are removed (as per his request). My client had asked for this in our initial meeting (see [Appendix A](#)) but was especially profound with how it turned out, stating that having all the zeros removed at once would make the game too easy and take away from the tournament aspect that my client really wanted to capture.

Another thing to note would be that my client is only somewhat satisfied with requirement 8, as they noted it would’ve been “cooler” to have the scores saved on a database, however due to the vagueness of the requirement and due to simplicity, a text file was used to save scores as it was a lot quicker in terms of coding, and reduced my work load significantly. Regardless, my

client did note that what I had developed did in fact do the job it was supposed to do, but they were expecting a slightly different method.

Furthermore, I asked my client how they felt about the feature of revealing a field and placing a flag on the field, and the overall consensus was that he was ecstatic with it, stating that it improved the “skill requirement”, because it required more human interaction than random luck.

5.3 Self Evaluation

I personally feel that the project that I have created has been sub par to what I feel I can really do. This is more down to myself and poor time management of the project, handled by myself. Regardless, it still meets the requirements that my client has asked for and the outcome does not truly represent the hours that I have put into this project. However, just as with any project, this project will be improved through many different additions or alterations, which will be noted in [5.4](#)

5.4 Future Improvements

If I was to attempt this project again, I would go about it in a completely different way. By this, I mean that I would give myself much stricter deadlines, and be proactive in completing the project sooner, giving myself more time for the write up section. Code-wise, I will implement more higher level algorithms in my second version, such as:

Recursion to remove ALL zeros that are linked - It is undeniable that my program could have made use of recursion in order to remove every zero that is linked to each other. This would be done by using each instance of ‘zone’ to contain an “isVisited” variable that tells the program whether this field needs to be checked for adjacent values, which would then allow me to use selection to decide whether the field is selected or not and then to remove all adjacent values, with the subroutine being defined in terms of itself.

Use a database to store the players details and score - The lack of a database definitely hinders this program, in that the usage of Query language limits the technicalities of the program. The database would only have one table called “Player”, that contains attributes First Name, Surname, Score and Difficulty.

Display the values of the grid field as a string rather than a picture - This improvement is only minor, as my program doesn’t use much memory, however it would significantly reduce the memory usage, as displaying a single char string would take up a lot less memory, due to the string only being 7 bits long, compared to the thousand pixel image of a number. This would be done using a linked list with randomised values within the list. I would be able to look at each index and output the value of each index accordingly.

Appendix A: Client Interview

Q: Why did you decide to use the game 'Minesweeper' for the upcoming tournament?

A: "Well, the answer is simple. Minesweeper is a very basic game that allows players to engage their brain in an abstract way to manipulate data given to them; and as gamers, we are all competitive and want to do better than each other, so this allows us to express our competitive nature whilst still enjoying what we do. Minesweeper is also a very minimal program, so there really can't be anything that goes wrong with it."

Q: Why did you choose to have the program only reveal the adjacent squares if a '0' is clicked on?

A: "Having all of the zeroes removed with a lucky hit makes the game just way too easy, so to add on to the tournament style, I want this game to be as challenging as it can be with a game of this standard."

Q: Why did you choose for the program to save the state of the game when initialised?

A: "Having the game save its state means that we can look back at what the game looked like and how potentially we could improve on our pattern spotting within minesweeper."

Q: Why do you plan to have the program contain multiple difficulties?

A: "Well, once the tournament is over, I would quite like to put up a stand where members can come and sit down and play the program, so having multiple difficulties would allow us to create a more 'fun' environment rather than the competitive tournament environment."

Q: Why have you decided for the program to be computer based?

A: "At the society, we currently have a couple of computers due to our surplus budget, but we will be hosting a tournament on these computers, and using these devices allow us to set up a sort of 'main stage'."

Q: How long have you been 'gaming'?

A: "As early as I can remember! My dad had been keen on getting some father and son time, so he would take us to the arcade and spend the afternoon playing arcade games like Pac-man, Space Invaders... You know, the usual arcade games from the 80's. Ever since then, I've always used video games as a way to connect with another reality, to hopefully forget the

struggles that run rampant through society. So those days with my dad will always be remembered when I turn on any console or machine, and brighten my day that bit more.”

Q: Would you say that society is a closed ‘family’?

A: “Err... Sort of? I don't know... I have heard from many people that have attempted to attend the society that they find it hard to join in with the social activities but that sort of stuff is sometimes needed to create that close relationship. In my opinion, if you come down to the society often enough then you are gradually involved in the ‘family’ culture. This is why I want this program, as it will invite others into our closer bubble.

Appendix B: Client Feedback

Question	Client Feedback
Is the User interface up to standard for the purpose of the program? If so, please state why?	Yes, I am MORE than happy with how the project looks aesthetically. It's a very simple and basic design, but that is exactly what minesweeper was... retro! Very challenging on the mind but very simple aesthetically. This user interface captures minesweeper exactly how I remember.
Would you regard the project as challenging enough for the users intended and the tournament that this project has been requested for?	This program is exactly what I needed for the tournament. The tournament difficulty means that I am able to hold this competition without any doubt of the players being given an 'easy' task. The fact that the players have to reveal each and every 3x3 space that has a zero still makes sure that the game doesn't reveal half the field with one lucky click, so this is a very nice balance in the difficulty.
Are you happy with the amount of features within the program, or would you rather there be more included next time?	Well, I feel that the game has captured exactly what I had hoped it would do, in that it's a near redevelopment of the hit game 'Minesweeper', so in regards to more features... Not really, obviously if that's something that you would like to add for your own personal gain then of course that could be arranged, but for the purpose that I need it for, it does all that I need it to do.
Are you pleased with the outcome of the zero field removal algorithm?	When we last spoke I really nailed it to you that having all the linking empty fields reveal themselves if you are to reveal one would make the game SO easy, and would remove the tournament aspect that I believe this game can capture, so the fact that this has been implemented brings me much joy because this removes any luck that can be involved, and brings in an increased skill factor as you don't know when the linked zeros will stop.
Are you satisfied with the saving of the players name into the scoreboard	Yes, although the scores could've been recorded on a database, which would've been quite cooler, I did ask for the scores to be recorded and saved, and didn't specify how or why, so if this made the project slightly easier for your workload, then I am okay with it, as it still fulfils what I have asked for.
Are you happy with the	Yes, I am quite impressed with this feature, because it allows the

saving of the state of the game?	players in the tournament to look back at games they would have just played and can spot patterns that may improve their skill to play the game and/or improve their mental stability, in that they are constantly using brain power to problem solve. The only downside that this feature has, is that the only way to access the saved state of the game is to open its internal folders, so I guess that would have to be put somewhere on the project interface, but that can be done whenever. Regardless, it has still been completed, so I can't complain too much.
Are you happy with how the player is expected to place flags on the board?	Yes, the fact that you have to click a button rather than use your mouse alone, increases the skill factor that this game requires if you want to complete it quickly, as you can't click on random spots and pray for the best, you have to move over to the button itself, and I believe that this will give the user a moment to second guess themselves, which is a bonus for cognitive engagement!
Following this, are you happy with the revealing field feature?	Again, this feature increases the skill requirement, as there is more human input required, so the more that a player has control over the game, the more there will be a skill requirement, so this captures exactly what is needed for the tournament.
Has the final product met all of your expectations?	Overall, this program has in fact met all of my expectations as all of the requirements that I had set out, and I fully believe that this tournament will bring the whole society together!

Appendix C: Code Listing

frmMainform.cs:

```
using System;

using System.Drawing;

using System.Windows.Forms;

using System.IO;

using System.Collections.Generic;

using System.Data.SQLite;

namespace Sapper
```

```
{  
    public partial class frmMainForm : Form  
    {  
  
        string NumToDisplay;  
        public Zone _selectedSquare = null;  
        private Zone[,] FieldArr = new Zone[12, 12];  
        public string difficulty;  
        private Random rng = new Random();  
        public int[,] FlagArr = new int[11, 11];  
        int[,] GridArr = new int[13, 13];  
        int StartButtonCounter = 0;  
  
        int bombsleft = 10;  
        int GridY;  
        int GridX;  
        int BombAroundX;  
        int BombAroundY;  
        int timetaken;  
  
        public frmMainForm()  
        {  
            InitializeComponent();  
            btnStartGame.Enabled = false;  
        }  
  
        public void InitializeTimer()
```

```
{
    timetaken = 0;
    timer.Interval = 1000;
    timer.Tick += Timer_Tick;

    timer.Enabled = true;
}

private void Timer_Tick(object sender, EventArgs e)
{
    timetaken += 1;
    lblTimeTaken.Text = Convert.ToString(timetaken);
}

private void btnStartGame_Click(object sender, EventArgs e)
{
    int bombsNeeded;

    timetaken = 0;
    StartButtonCounter += 1;

    StreamWriter FileSave = new StreamWriter(new
FileStream("GridContents.txt", FileMode.OpenOrCreate));

    StreamWriter GridLocationFile = new StreamWriter(new
FileStream("GridLocation.txt", FileMode.OpenOrCreate));

    if (StartButtonCounter == 1)
    {
        InitializeTimer();
    }
}
```

```
}  
pnlGrid.Controls.Clear();  
if (cmbxDifficulty.SelectedIndex == 0)  
{  
    pnlGrid.Width = 250;  
    pnlGrid.Height = 250;  
    bombsNeeded = 5;  
    difficulty = "Easy";  
}  
else if (cmbxDifficulty.SelectedIndex == 1)  
{  
    pnlGrid.Width = 350;  
    pnlGrid.Height = 350;  
    bombsNeeded = 7;  
    difficulty = "Medium";  
}  
else  
{  
    pnlGrid.Width = 500;  
    pnlGrid.Height = 500;  
    bombsNeeded = 10;  
    difficulty = "Hard (Tournament)";  
}
```

```
btnRevealZone.Enabled = false;  
btnPlaceFlags.Enabled = false;
```

```
int MazeSquaresHorizontal = pnlGrid.Width / Zone.ZoneWidth;

int MazeSquaresVertical = pnlGrid.Height / Zone.ZoneHeight; //
Setting the size of each panel

for (int i = 0; i < MazeSquaresVertical; i++)
{
    for (int j = 0; j < MazeSquaresHorizontal; j++)
    {
        Zone newGridSpace = new Zone();
        int bombsAround = 0;

        while (bombsNeeded != 0)
        {
            GridX = rng.Next(10) + 1;
            GridY = rng.Next(10) + 1;
            if (GridArr[GridX, GridY] != 9)
            {
                GridArr[GridX, GridY] = 9; // 9 = bomb

                bombsNeeded -= 1;
            }
        } //This While loop places bombs.

        //if (GridArr[j + 1, i + 1] == 9)
        //{
            //    newGridSpace.Click += SelectField; // Used for
determining which gridspace should be selected/deselected
```

```

        //    newGridSpace.Location = new Point(i *
newGridSpace.Width, j * newGridSpace.Height);

        //    pnlGrid.Controls.Add(newGridSpace); // adds the
gridSpace to the panel

        //} // This places ONLY the bombs (Testing purpose only)


newGridSpace.GridY = i + 1;
newGridSpace.GridX = j + 1;


newGridSpace.Click += SelectField;// Used for clicking a
field

newGridSpace.Location = new Point(j * newGridSpace.Width,
i * newGridSpace.Height);
pnlGrid.Controls.Add(newGridSpace);// adds the gridSpace
to the panel


FieldArr[i + 1, j + 1] = newGridSpace;
for (int Xblank = 0; Xblank < 11; Xblank++)
{
    Zone Blank = new Zone();
    Blank.GridX = Xblank;
    Blank.GridY = 0;
    FieldArr[0, Xblank] = Blank;
    Blank.GridY = 11;
    FieldArr[11, Xblank] = Blank;
} //Fills in left side to be blank

```

```
for (int Z = 0; Z < 11; Z++)
{
    Zone Blank = new Zone();
    Blank.GridX = 0;
    Blank.GridY = Z;
    FieldArr[Z, 0] = Blank;
    Blank.GridX = 11;
    FieldArr[Z, 11] = Blank;
    FieldArr[11, 11] = Blank;
} //Fills in top side to be blank

FieldArr[i + 1, j + 1].GridX = newGridSpace.GridX;
FieldArr[i + 1, j + 1].GridY = newGridSpace.GridY;

BombAroundX = i + 1;
BombAroundY = j + 1;

for (int X = -1; X <= 1; X++)
{
    for (int Y = -1; Y <= 1; Y++)
    {
        if (GridArr[BombAroundX, BombAroundY] != 9)
        {
            if (GridArr[BombAroundX + X, BombAroundY + Y]
== 9)
            {
                bombsAround = bombsAround + 1;
            }
        }
    }
}
```

```

                                GridArr[BombAroundX, BombAroundY] =
bombsAround;

                                }

                                }

                                }

                                } //sets the numbers of bombs adjacent to the field
                                }

                                } //Setting up grid

for (int i = 0; i <= 11; i++)
{
    for (int j = 0; j <= 11; j++)
    {
        if (FieldArr[i, j] != null)
        {
            int lineToSaveX = GridArr[j, i];
            FileSave.Write(lineToSaveX + ","); //Saving contents
of the 2d-array (grid) to a fil

        }

    }

    FileSave.WriteLine();

} //writing Grid contents to a file

```

```
        FileSave.Close();

        for (int i = 0; i <= 11; i++)
        {
            for (int j = 0; j <= 11; j++)
            {
                if (FieldArr[i, j] != null)
                {
                    int lineToSaveX = FieldArr[i, j].GridX;
                    int lineToSaveY = FieldArr[i, j].GridY;
                    GridLocationFile.Write(lineToSaveX + ","); //Saving
contents of the 2d-array (grid) to a fil
                    GridLocationFile.Write(lineToSaveY + ";");

                }

            }

            GridLocationFile.WriteLine();

        } //writing Field Location to a file.
        GridLocationFile.Close();
    }

    private void SelectField(object sender, EventArgs e)
    {
        if (_selectedSquare != null)
        {
```

```
        _selectedSquare.BorderStyle = BorderStyle.FixedSingle;
    }

    btnRevealZone.Enabled = true;
    btnPlaceFlags.Enabled = true;

    _selectedSquare = ((Zone)sender);
    _selectedSquare.BorderStyle = BorderStyle.Fixed3D;
} // runs when a panel is clicked

private void RemoveZeroesFromNeighbours()
{

    Zone Square = new Zone();
    for (int i = -1; i <= 1; i++)
    {
        for (int j = -1; j <= 1; j++)
        {

            Square = FieldArr[_selectedSquare.GridY + i,
            _selectedSquare.GridX + j];

            Square.SizeMode = PictureBoxSizeMode.StretchImage;

            NumToDisplay = GridArr[Square.GridX,
            Square.GridY].ToString();

            //MessageBox.Show(GridArr[Square.GridX,
            Square.GridY].ToString());

            Square.Load("D:\\Sapperpog\\Sapper\\bin\\Debug\\number-"
            + NumToDisplay + ".jpg");
```

```

        }

        }//removes a 3x3 square with the field clicked in the middle

    } //

private void btnPlaceFlags_Click(object sender, EventArgs e)
{
    Zone Flag = new Zone();

    Flag = FieldArr[_selectedSquare.GridY, _selectedSquare.GridX];

    FlagArr[_selectedSquare.GridX, _selectedSquare.GridY] =
GridArr[_selectedSquare.GridX, _selectedSquare.GridY];

    Flag.SizeMode = PictureBoxSizeMode.StretchImage;

    Flag.Load("D:\\Sapperpog\\Sapper\\bin\\Debug\\flag.jpg");

    CheckWin();

} //Flags are the key to winning condition

private void btnRevealZone_Click(object sender, EventArgs e)
{
    _selectedSquare.BorderStyle = BorderStyle.Fixed3D;

    //MessageBox.Show(Convert.ToString(GridArr[_selectedSquare.GridX,
_selectedSquare.GridY])); // shows number behind grid testing purposes only!

    _selectedSquare.SizeMode = PictureBoxSizeMode.StretchImage;

    NumToDisplay = GridArr[_selectedSquare.GridX,
_selectedSquare.GridY].ToString();

    _selectedSquare.Load("D:\\Sapperpog\\Sapper\\bin\\Debug\\number-"
+ NumToDisplay + ".jpg");//Determines what image should be outputted in
reference to the number of bombs adjacent

```

```
        if (Convert.ToInt32(NumToDisplay) == 0)
        {
            RemoveZeroesFromNeighbours();
        } //Removes adjacent squars
        else if (Convert.ToInt32(NumToDisplay) == 9)
        {
            Lost();
        } //losing condition
    }

    private void CheckWin()
    {
        if (FlagArr[_selectedSquare.GridX, _selectedSquare.GridY] == 9)
        {
            bombsleft += -1;
        }
        if (bombsleft == 0)
        {
            MessageBox.Show("You win! You won in: " + timetaken + "
seconds!");
            StreamWriter playerDetails = new StreamWriter(new
FileStream("Highscores.txt", FileMode.OpenOrCreate));
            playerDetails.WriteLine(tbxFirstName.Text + " " +
tbxSurname.Text + " Difficulty: " + difficulty + " Time Taken: " +
timetaken);
            playerDetails.Close();
            timer.Stop();
        }
    }
}
```

```
        } //Checks whether 10 flags have been placed on 10 bombs

    }

    private void Lost()
    {
        frmYouLost_ f1 = new frmYouLost_();
        f1.ShowDialog();
        this.Close();
    }

    private void btnViewRules_Click(object sender, EventArgs e)
    {
        MessageBox.Show("To beat Sapper, you must place a flag on all
mine locations. The number that follows when clicked on a tile corresponds to
how many mines are adjacent to that tile.");
    }

    private void frmMainForm_Load(object sender, EventArgs e)
    {
        btnPlaceFlags.Enabled = false;
        btnRevealZone.Enabled = false;
        cmbxDifficulty.SelectedIndex = 0;
    }
```

```
        private void tbxSurname_TextChanged(object sender, EventArgs e)
        {
            btnStartGame.Enabled = true;
            label2.Text = "";
        }
    }
}
```

Class Zone:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

```
namespace Sapper
{
```

```
public class Zone : PictureBox
{
    public Zone()
    {
        this.Width = 50;
        this.Height = 50;
        this.BorderStyle = BorderStyle.FixedSingle;
    }
    public const int ZoneWidth = 50;
    public const int ZoneHeight = 50;
    public int GridX { get; set; }
    public int GridY { get; set; }
    public bool isVisited;
}
}
```

frm_YouLost!.cs

```
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;
```

```
namespace Sapper
{
    public partial class frmYouLost_ : Form
    {
        public frmYouLost_()
        {
            InitializeComponent();
        }

        private void frmYouLost__Load(object sender, EventArgs e)
        {
            MessageBox.Show("Game over :(");
        }
    }
}
```