



Algorithm & Data Structure
Project Report Semester 2 2022-2023:

Minesweeper

Team Members:

ITITWE17029 Nguyễn Đình Minh Phúc (Saturday morning lab)

TABLE OF CONTENTS

I. INTRODUCTION	3
II. IMPLEMENTATION	5
III. FUNCTIONS	5
IV. PERSONAL CONTRIBUTION	14
V. REFERENCES	14

I/ Introduction

Minesweeper is a popular board game shipped with many operating systems by default. The goal of the game is to clear the board without detonating any mine on the board. If the player clicks on the block which contains a mine, the mine detonates and the game is over.

A block can contain a number or it can be blank. If a block is blank (have zero mines around it), it will continue to open more blocks surrounding it until it hits block with numbers (a block which has 1 or more mines around it).The number indicates how many mines are adjacent to this particular block.

We can also set flags on blocks by right clicking on it, to keep track the blocks with mine.

II/ Implementation

In this Minesweeper project, I choose to work with Angular because it is one of my few stronger framework, using HTML and CSS to represent user-interfaces and TypeScript, which is a superset of JavaScript, to handle the interactions.



Angular is a platform and framework for building single-page client applications using HTML, CSS and TypeScript. Angular is written in TypeScript. It implements core and optional functionality as a set of TypeScript libraries that you import into your applications.



TypeScript is developed and maintained by Microsoft, an object-oriented, open-source programming language. It's a superset of JavaScript that supports both dynamic and static typing. It provides classes, visibility scopes, namespaces, inheritance, unions, interfaces, and many other features. Also, it offers comments, variables, statements, expressions, modules, and functions.. Also, it compiles to plain JavaScript.



and has a rich

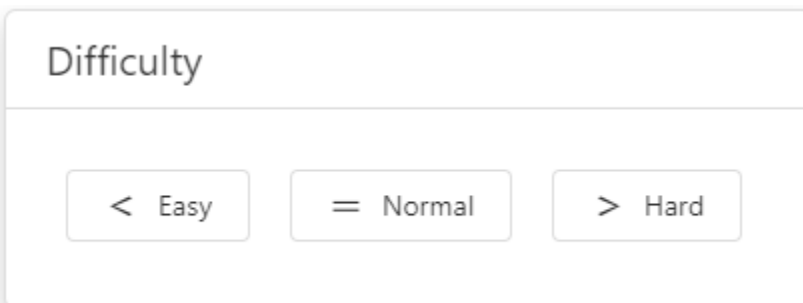
Visual Studio Code is a lightweight but powerful source code editor made by Microsoft and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js ecosystem of extensions for other languages and runtimes

III/ Functions

1. `selectDifficulty(data:short)`: Choose difficulty and get assets to draw board

```
selectDifficulty(data: any){
  if(data == this.Difficulty.Easy){ // 10x10 board
    this.width = 700;
    this.height = 700;
    this.shapedimension = 70;
    this.flagCount = 10;
    this.mine = 10;
    this.empty = 90;
  }
  else if(data == this.Difficulty.Normal){ // 16 x 16 board
    ///////////////
  }
  else if(data == this.Difficulty.Hard){ // 24 x 24 board
    ///////////////
  }
  if(this.currentDifficulty === data){
    this.drawBoard();
  }
  this.currentDifficulty = data;
  this.difficultyPopup = false;
}
```

After entering web page, there will be a popup with 3 difficulties. User must choose one of the difficulties to proceed to the board.



Difficulty

< Easy = Normal > Hard

2.drawBoard(): get assets from selectDifficulty() and draw the board

```
const cw = Math.fround(this.width );
const ch = Math.fround(this.height );
this.canvasRef.nativeElement.width = cw;
this.canvasRef.nativeElement.height = ch;
const ctx: CanvasRenderingContext2D =
this.canvasRef.nativeElement.getContext('2d');
ctx.clearRect(0,0,cw,ch);
ctx.strokeStyle = 'rgb(185, 173, 173)'; // grid borderline
```

To draw the board, I use canvas to draw grid on the board.

```
this.shapes = new Array(Math.fround(this.width/this.shapedimension))
for (let i = 0; i < this.shapes.length; i++) { this.shapes[i] = new
Array(Math.fround(this.height/this.shapedimension)) };
  for (let i = 0; i < this.shapes.length; i++) {
    for (let j = 0; j < this.shapes[i].length; j++) {
      let type = 0; // 0 = normal block, 1 = mine block
      let x = i * this.shapedimension;
      let y = j * this.shapedimension;
      let isOpened = false;
      let isFlagged = false;
      ctx.fillStyle = "#696969";
      ctx.fillRect(x, y, this.shapedimension, this.shapedimension);

      ctx.strokeStyle = "#0000ff";
      ctx.lineWidth = 2;
      ctx.strokeRect(x, y, this.shapedimension, this.shapedimension);
      this.shapes[i][j] = { x, y, type, isOpened , isFlagged};
    }
  }
```

Next, 2D array is initialized, and each block is set with assets from selectDifficulty().

```
let mineCount = 0;
while(mineCount < this.mine){
  let tempI = Math.floor(Math.random()* this.shapes.length);
  let tempJ = Math.floor(Math.random()* this.shapes[0].length);
  if(this.shapes[tempI][tempJ].type === 1){
    continue;
  }
  else{
    this.shapes[tempI][tempJ].type = 1;
    mineCount++;
  }}
}
```

After initializing the board, all that left is to plant mine on the board. In fact, just change the certain number of block's type to 1 (Mine).

3.rightClick(e: any): Place or unplace a flag on a unopened block

```
for(let i = 0; i < this.shapes.length; i++){
  for(let j = 0; j < this.shapes[0].length; j++){
    if(e.offsetX > this.shapes[i][j].x && e.offsetX <= this.shapes[i][j].x +
this.shapedimension
    && e.offsetY > this.shapes[i][j].y && e.offsetY <= this.shapes[i][j].y
+ this.shapedimension){
      if(!this.shapes[i][j].isOpened){
        const ctx: CanvasRenderingContext2D =
this.canvasRef.nativeElement.getContext('2d');
        if(!this.shapes[i][j].isFlagged){
          if(this.flagCount > 0){
            this.shapes[i][j].isFlagged = true;
            let flag = new Image();
            flag.src = "assets/flag.png";
            flag.onload = () => {
              // draw flag section
              //
              this.flagCount--;
              return;
            }
          }
          else{
            return;
          }
        }
        else{
          this.shapes[i][j].isFlagged = false;
          // redraw block section
          //
          this.flagCount++;
        }
      }
    }
  }
}
```

```
}
```

In Minesweeper, there are limited number of flags available. User can flag or unflag on a block whenever they want. When a block is flagged, aside from unflagging, user cannot perform any action on that block until they are unflagged.

When user performs a mouse click on UI, \$event includes offsetX and offsetY of the position that user clicked. Using these assets to pinpoint the block that fits the description. From there, perform placing flag action.

4. leftClick(\$event): sweep a block

```
for(let i = 0; i < this.shapes.length; i++){
  for(let j = 0; j < this.shapes[0].length; j++){
    if(e.offsetX > this.shapes[i][j].x && e.offsetX <= this.shapes[i][j].x +
this.shapedimension
    && e.offsetY > this.shapes[i][j].y && e.offsetY <= this.shapes[i][j].y
+ this.shapedimension){
      const ctx: CanvasRenderingContext2D =
this.canvasRef.nativeElement.getContext('2d');
      if(!this.shapes[i][j].isFlagged){
        if(this.shapes[i][j].type === this.BlockType.Mine) // if block has
mine
        {
          this.gameOver();
        }
        else{
          this.openBlock(i,j);
          if(this.empty === 0){
            this.winPopup = true;
          }
        }
      }
    }
  }
}
```

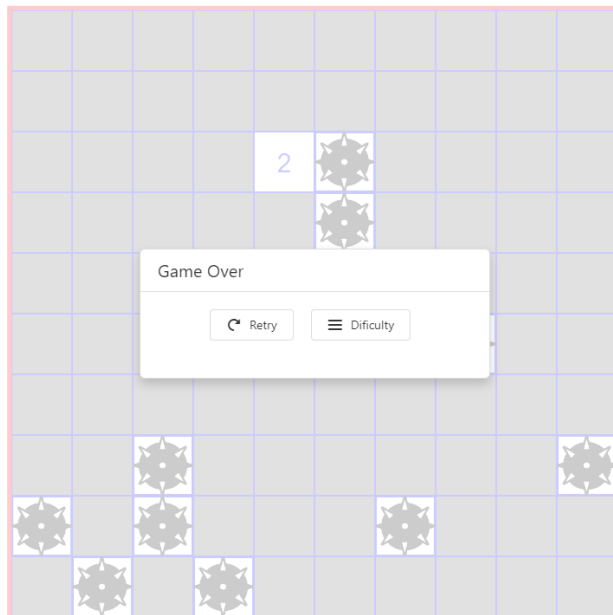
Same procedure as rightClick() function, but different logic. As mention before, if a block is flagged, leftClick() will return and do nothing to the program.

If it is not flagged, proceed to check whether a block has mine or not. If there is a mine under that block, go straight to game over popup. Else perform recursive function, openBlock().

5. gameOver():

```
const ctx: CanvasRenderingContext2D =
this.canvasRef.nativeElement.getContext('2d');
  for(let i = 0; i< this.shapes.length; i++){
    for(let j = 0; j< this.shapes[0].length; j++){
      if(this.shapes[i][j].type === this.BlockType.Mine){
        this.shapes[i][j].isOpen = true;
        let flag = new Image();
        flag.src = "assets/mine.png";
        flag.onload = () => {
          // draw mine section
          //
        }
      }
    }
  }
  this.gameoverPopup = true;
```

When user clicks on a mine, the board now redraws every blocks with mine under it, and popup a gameover popup by changing Boolean gameoverPopup = true.



6. openBlock(i,j: int)

```

var neighborMineCount = 0;
    for(let x = i-1; x <= i+1; x++) {
        for(let y = j-1; y<= j+1;y++){
            if((x >= 0 && x < this.shapes.length) && (y >=0 && y<
this.shapes[0].length)){
                if(this.shapes[x][y].type === this.BlockType.Mine){
                    neighborMineCount++;
                }
            }
        }
    }
}

```

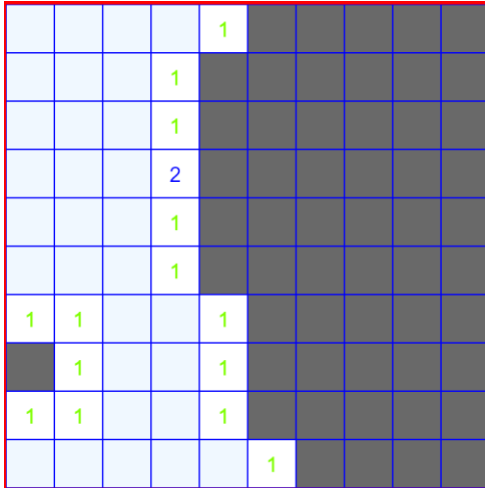
00	01	02	03	04	05	06	...
16	17	18	19	20	21	22	...
32	33	34	35	36	37	38	...
48	49	50	51	52	53	54	...
64	65	66	67	68	69	70	...

It is already established that the block being processed here is empty, but first, it is essential to know whether there is mine around the block

```

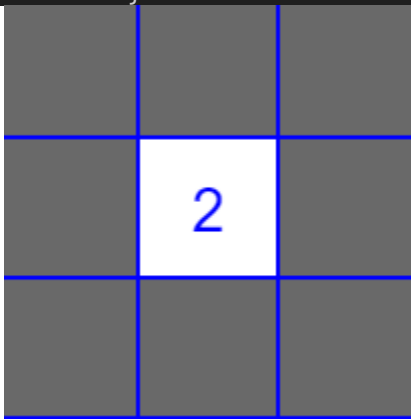
if(neighborMineCount === 0){
    // draw empty block section
    //
    if(i - 1 >= 0){
        this.openBlock(i - 1, j);
    }
    if(j - 1 >= 0){
        this.openBlock(i, j -1);
    }
    if(i + 1 < this.shapes.length){
        this.openBlock(i + 1, j);
    }
    if(j + 1 < this.shapes[0].length){
        this.openBlock(i, j + 1);
    }
}
}



```











If the block has no mine around, redraw the block with empty block and continue to open empty blocks by doing this function recursively to the block's adjacent blocks.

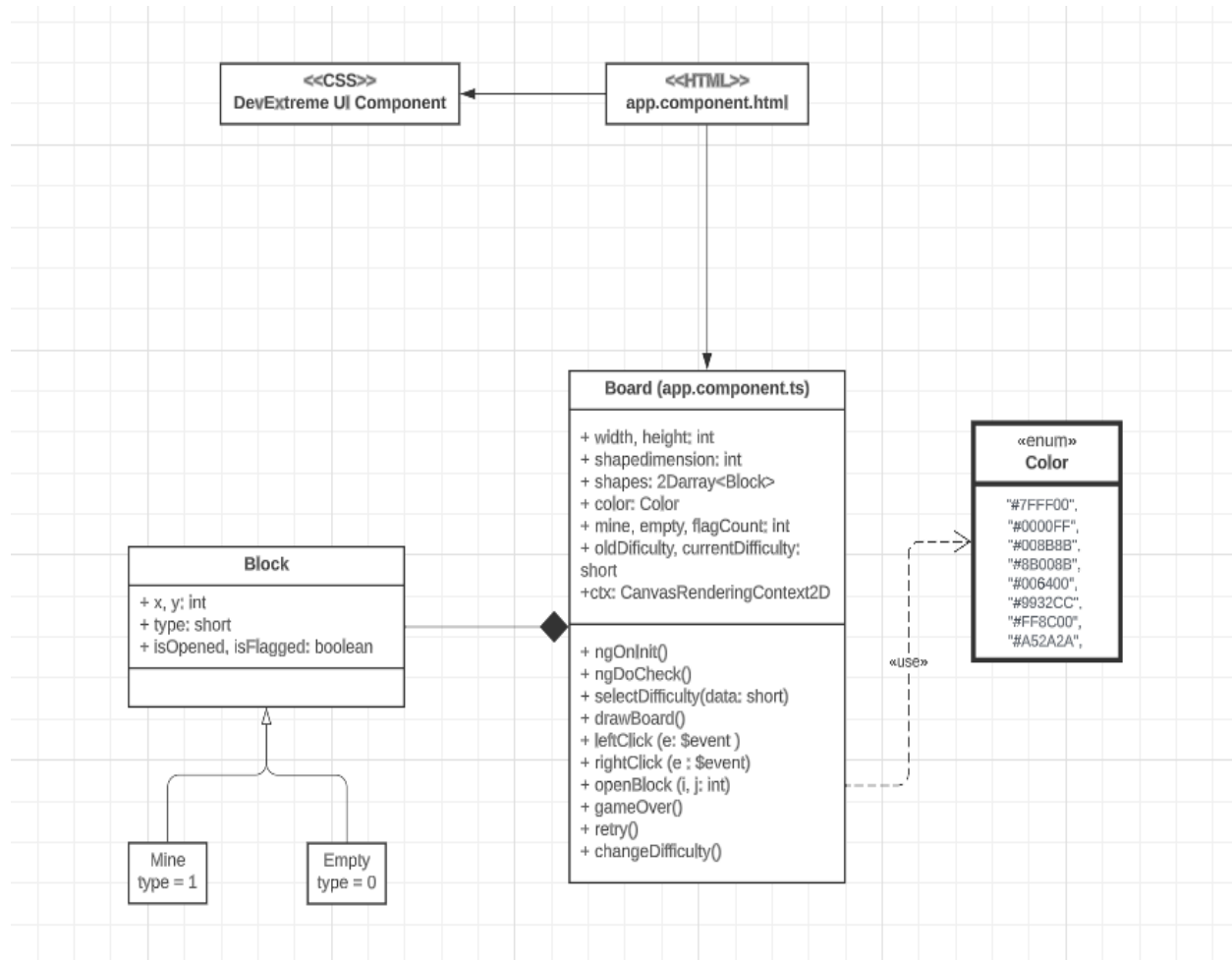
```
else{
    ctx.clearRect(this.shapes[i][j].x, this.shapes[i][j].y,
this.shapedimension, this.shapedimension );
    ctx.fillStyle = this.color[neighborMineCount-1];
    ctx.textAlign = "center";
    ctx.font = "30px Arial";
    ctx.fillText(`${neighborMineCount}`,this.shapes[i][j].x +
this.shapedimension/2,this.shapes[i][j].y + this.shapedimension/1.5);
    ctx.strokeStyle = "#0000ff";
    ctx.lineWidth = 2;
    ctx.strokeRect(this.shapes[i][j].x, this.shapes[i][j].y,
this.shapedimension, this.shapedimension);
    this.shapes[i][j].isOpen = true;
    this.empty --;
}
```



	1	1	1			
	1		2	1		
	1	2		1		
		1	1	1		

	1	2	3	2	1	
	2				2	
	3		8		3	
	2				2	
	1	2	3	2	1	

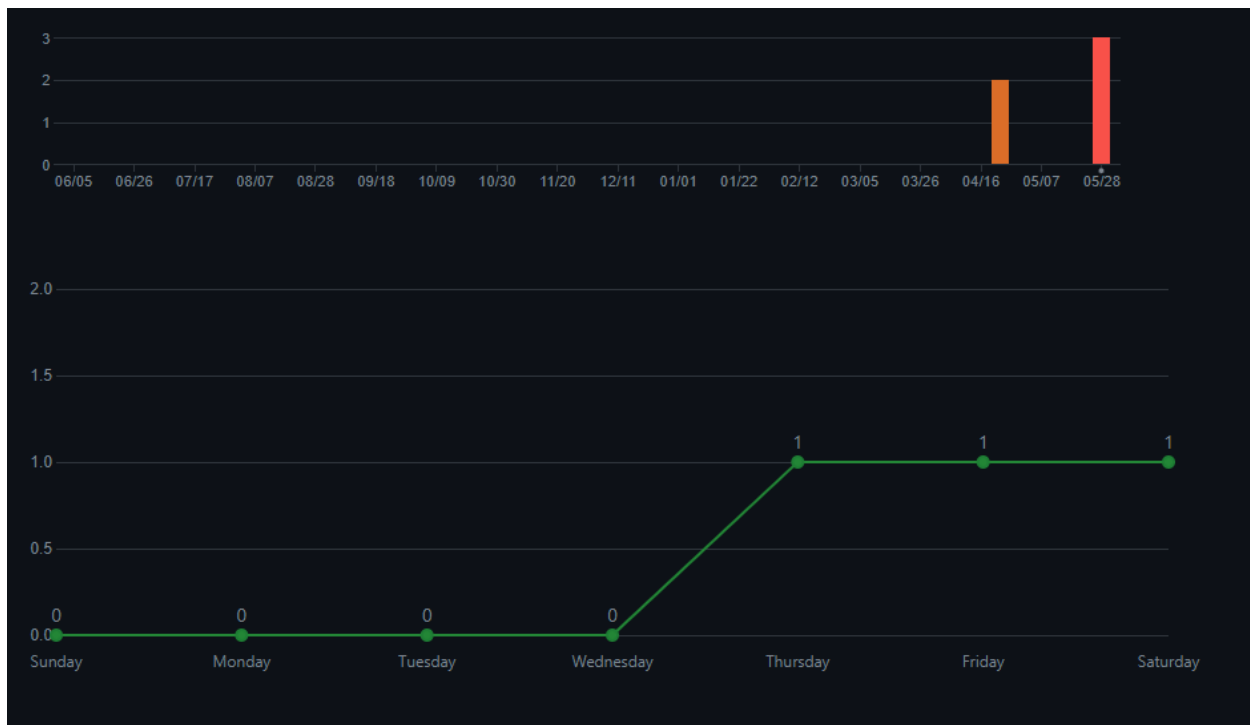
Else get CSS color from Color enum, open the block by redrawing the block, and add a number based on the number of mines around the block with colors from Color enum.



IV/ Personal Contribution

Project's GitHub: https://github.com/Megahutner/Minesweeper_DSA

Project's Timeline:



V/ References

<https://angular.io>

<https://js.devexpress.com>

<https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D>