Minesweeper

The target of this programming exercise was to recreate the fames Minesweeper game traditionally included in builds of Windows 3.1x and higher. Although since Windows 8, it is no longer included by default, it is still available on Microsoft Store.

Algorithms

Algorithmically, this solution is not as complicated, but it does a lot to acquaint the coder with general C# language structure and syntax.

The only noteworthy algorithm choice is that the tile numbering is generated in conjunction to placing the bombs themselves (as opposed to generating it after the bomb placement is done).

Program structure

A new game is initialized by specifying the desired difficulty in the StartForm, which in turn sets the bomb generation rate for each individual Game (there can be many running in parallel).

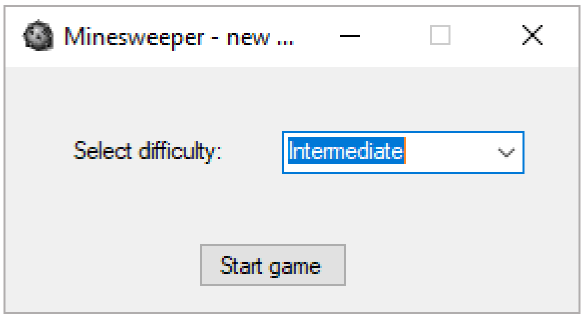


Figure 1: StartForm with Intermediate difficulty selected.

Quite evidently, every game, started from the StartForm is an independent object. It contains the main gamestate logic and instantiates the game Field, which contains individual tile data and the GUI / GameForm, which handles rendering and receives click events, which are passed through the Game object and evaluated by the Field object.

The GUIs are facilitated by using Windows Forms.

The GameForm is instantiated by the game class and is sent only the parent Game object for click callbacks. It also accesses the Field objects to get the individual tilestates and render them, one by one. It does so by using the embedded assets of tiles 16x16 px and reconstructs the images from EmbeddedResourceStreams.

The tiles are drawn one by one, at each GameForm invalidation. This highlights the first potential future feature, where only the changed tiles would be repainted, without triggering a full game repaint on every PaintEvent.

The GameForm also shows a StatusBar, containing info about flags placed / total bombs, which are properties of the underlying Field object.

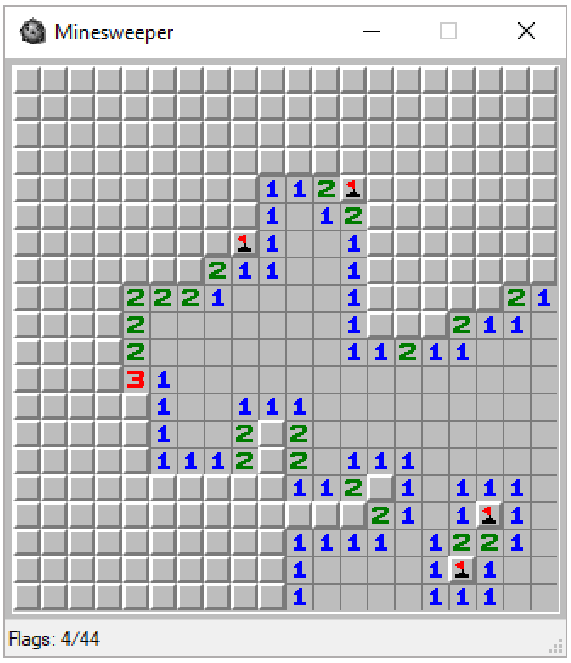


Figure 2: GameForm with a started game.

The Field object represent a rectangular array of tiles (with configurable dimensions) containing the individual tilestate, encapsulated in three main arrays: Tile State, where 0 represents a hidden tile, 1 a flagged tile and 2 an uncovered tile, next the TileNum array, which stores the indidual numbers displayed on the tiles and TileBomb, which is a boolean array, where true represents a bomb.

This class is initialized by the Game object. It also contains the final ClickHandlers, which determine the action taken, when clicking on a tile.

When an end state is reached, either by clicking on a tile with a bomb, or by correctly flagging all the bombs.

You can either restart the Game with the same settings, or close this instance of the Game.

A User Guide

The game starts with the Figure 1 Select difficulty dialog, where you select the desired difficulty. If you choose not the explicitly specify the game difficulty, it defaults to "Beginner".

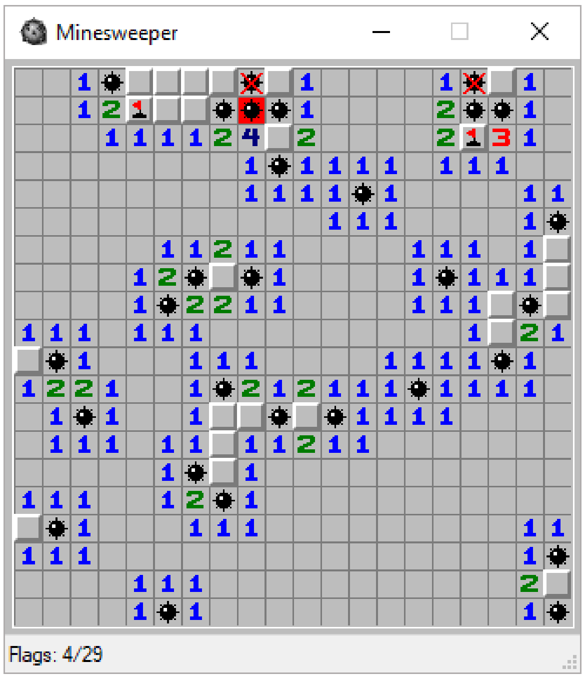
Then you can click the Start Game button, which opens a new window with the game itself.

Here, the controls are is follows – the game plane is divided into tiles, which can be either left-clicked to reveal the tile beneath, or right-clicked to mark a tile containing a bomb (or alternatively dismiss the flag from an already marked tile).

Note that flagged tiles cannot be clicked to prevent accidental bomb triggerings.

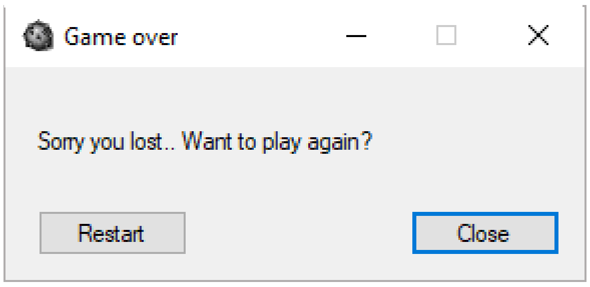
Each click can also be cancelled by moving from the clicked tile while still holding the pressed mouse button and then releasing it outside the bounds of the clicked square.

The bomb placement is to be determined through the number displayed on tiles, which is the sum of number of bombs in the surrounding 8 squares.



Flags: 4/29

Figure 4: End state with a triggered bomb.



Sorry you lost.. Want to play again?

Figure 3: Lost game – Game over dialog.

Once all of the bombs (whose total count can be seen on the status bar after the slash) have been correctly identified. The congratulates you and offers you to restart the current game or alternatively close it.

Note, that the same (without the congratulations) occurs when you trigger a bomb by clicking on it. This results in the bombs being uncovered, where the flags left behind denote correctly marked bombs, the crossed-out bombs symbolize a misplaced flag and the triggered bomb is highlighted in red.

Programming Proceedings

To be frank, this project took far more time than I initially expected. The main game logic was quite simple to write. The actual game window took far more time, as I had to learn the Windows Forms object model.

Potential future features

A customizable Field size would be quite simple to implement, as all the backend stuff has been made for it. The StartForm would have to be modified and and the user guide would have to be expanded.

A "fair" minesweeper was also suggested, where if you were to trigger a bomb, it would not go off if it was not certain it would be there. This would essentially require generating a new minefield containing bombs to suit such arrangements.

It would also be possible to implement custom scaling on the game, instead of using the actual tile bitmap size.

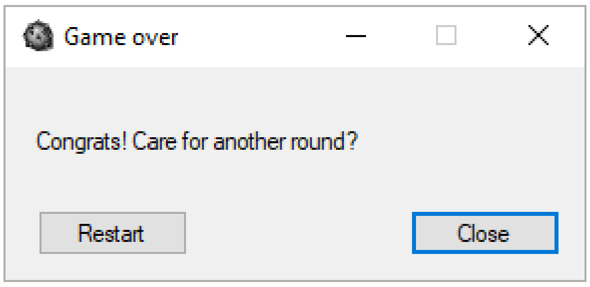
This game could also be implemented in a universal fashion, refraining from Windows Forms and the OS lock that comes with them.

A competitive mode could be introduced, where each player strives to flag as many bombs as possible and would receive minus points for triggering a bomb.

The StatusBar could be replaced with the original header.

In Conclusion

This programming exercise was mainly a good primer for the C# course that follows and in OOP as well. It was quite fun to be honest, to recreate such an iconic game.


Game over

Congrats! Care for another round?

Figure 5: Game won – Game over dialog..