# Project Minesweeper

Tai Pham Nguyen Anh
Student ID: 23125016
Ho Chi Minh University of Science
pnatai23@apcs.fitus.edu.vn

## Main Features

I have included all the features specified in the project's description:

1. Drawing the board (including cells, mines, flags).
2. Changing the state of each cell when it is selected.
3. Timer.
4. Function:
   - New game.
   - Change board dimensions.
5. Randomize positions of the mines.
6. Receiving new position of the pointer (to select a cell) from screen.
7. Saving the current state of the board and loading in the next playing session: the game will automatically save the current state of the game when you click the main menu button when playing.
8. Saving high scores: The game will automatically update the highest score when you win a game.

## Other features:

- Render images, textures, game grid, … using raylib.
- Improved score calculating.
- Using `struct`.

## Structure of source code

### Summary:

For simplicity, I've divide major components of this game to 3 files: minesweeper.cpp, field.h, cell.h

- minesweeper.cpp: The main file, this handles graphics, draw textues and handle all the interactions of the game.
- field.h: This defines the structure and functions of the game field.
- cell.h: This defines the structure and functions of the cell that game field is going to use.

### Detailed description:

**minesweeper.cpp:**

1. The first three lines is to include necessary files.

2. Constants:
   - App state
   - Game resolution

3. Variables:
   - Textures (logo, buttons, …)
   - Other supporting variables for printing and scaling.

4. Functions:
   - `void mousezoomButton(const Vector2 &mousePoint, const Rectangle &buttonRec, float &scale_obj)`: Zoom the button texture when the mouse is hovering to a specific button.
   - `int getNum(int num, int ind)`: Get the digit at position *ind* of *min(num, 60)* for timer.
   - `void LoadTextures()`: Load all the necessary textures for rendering UI of the game.
   - `void UnloadAll()`: Unload all the textures that it have loaded in `LoadTexture()`.
   - `void backgroundFunc()`: Horizontally scrolling the background.
   - `int getDigit(int x, int ind)`: get the $ind^{th}$ digit from right of *x*.
   - `bool isActionAt(const Rectangle &block)`: Checks if mouse button is inside block and is pressed.
   - `Rectangle getRecCenter(const Rectangle &block, float scale_obj)`: Helper function to get `Rectangle` object of the scaled `block`.
5. `int main()` fucntion:
   - First lines are to prepare the game rendering.
   - Vectors *prop, mnVal, mxVal* are to limit the maximum possible values of the GameGrid.
   - When the Escape key is not pressed or other closing signals are not encountered:
     1. Handle the events, then updates variables and states from interaction of user and the game.
     2. Render the coresponding UI after interaction.

**field.h:**

1. Constants:
   - CHORD_BONUS: used to calculate the score when chording.
   - GAMESTATE: game states.
2. `struct Field`: This is the structure of grid of the game.
   1. Variables: including grid managing variables and necessary texture variables to render the grid.
   2. Functions:
      - `bool isInside(int row, int col)`: check if (row, col) position is inside the grid.
      - `bool canOpen(int row, int col)`: check if the cell can be opened when opening an empty cell.
      - `void startSweep(int row, int col, int pointBonus)`: open (row, col) cell and surrounding cells when (row, col) is empty using BFS, pointBonus is 2 if it is from chording, 1 if it is not.
      - `void toggleFlag(int row, int col)`: flag or unflag a cell.
      - `void generateField(int row, int col)`: randomly generate positions for mines, and initialize the states of all cells.
      - `void openSurroundings(int row, int col)`: handle chording at (row, col).
      - `void showAllMines()`: show all mines when losing.
      - `void takeAction()`: handle actions from user.
      - `void printCell(int row, int col)`: render the cell at (row, col).
      - `void printGame()`: render the grid.
      - `void SaveGame()`: save the game.
      - `bool LoadGame()`: load the game from *save.txt*, return `false` when *save.txt* is not exist.
      - `void init(int num_row, int num_col, int num_bomb)`: initialize the variables, textures.
      - `int calcFinalScore()`: calculate the current score using the following fomula:

$$\text{current score} = \text{initial score} + \text{chord score} + \text{opened cell score} + \text{risk score} - \text{penalty}$$

      - Where:

    - initial score: calculated proportional to the $\frac{\text{number of mines}}{\text{number of cells}}$
    - chord score $= 2 \cdot$ cells opended by chording
    - opened cell score $= 1 \cdot$ cells opended not by chording
    - risk score $= 2 \cdot$ number of mines surrounding opened cells
    - penalty $= 5 \cdot$ time lapsed in seconds
  - `int getElapsed()`: get the elapsed time in seconds.
  - `~Field()`: destructor of Field structure, unload the textures in GPU.
3. Flow:
   1. Initialize using `void init(int num_row, int num_col, int num_bomb)`.
   2. Use `takeAction()` to handles interactions from user.
      - Invoke `generateField(int row, int col)` first time clicked the cell, guarantee that the first clicked cell is neither mine nor adjacent to mine.

**cell.h:**

1. Constants:
   - CELL_STATE: This represent the state of the cell
     - NOTHING: Neither a mine nor adjacent to mine.
     - MINE: mine.
     - ADJ_MINE: adjacent to mine.
2. `struct Cell`:
   1. Variables:
      - `int num_adj`: stores the number of mines adjacent to this cell.
      - `CELL_STATE state`: the state of the cell.
      - `bool hidden`: `true` if this cell is hidden, `false` when it is not.
      - `bool flagged`: `true` if this cell is flagged, `false` when it is not.
      - `Cell()`: constructor to initialize the default states of the cell.

## How to play

1. Open the `minesweeper.exe`

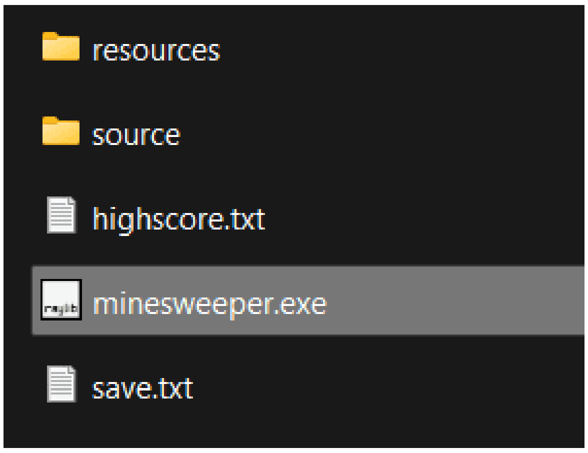
resources
source
highscore.txt
minesweeper.exe
save.txt

Figure 1: Run the minesweeper.exe

2. In the main menu:

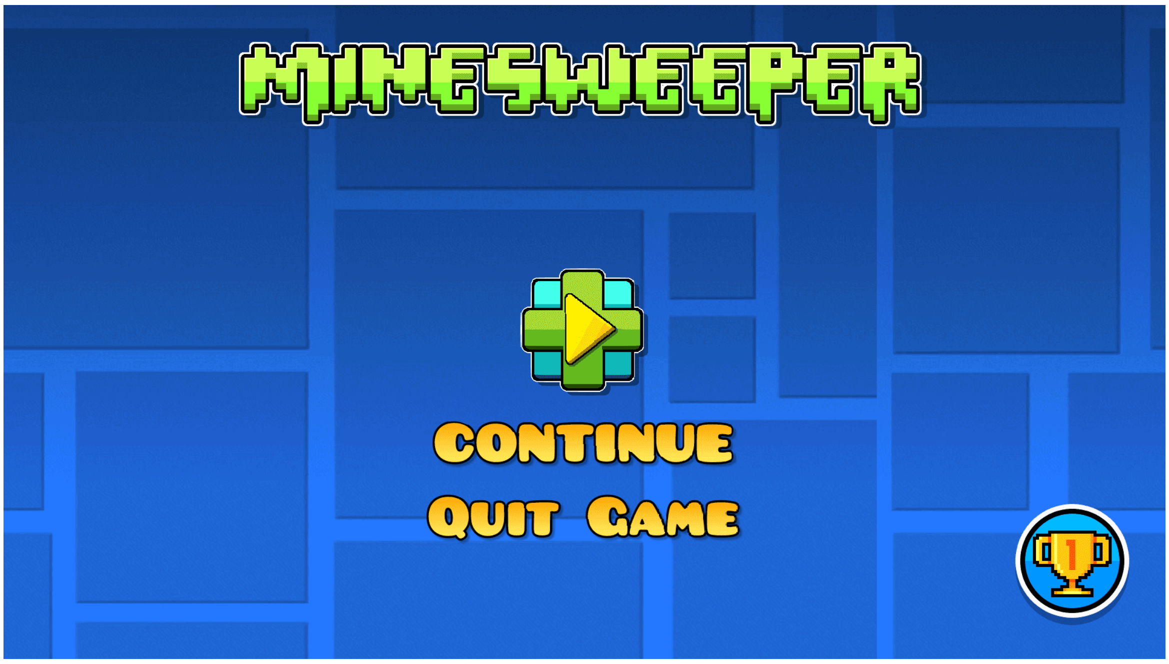

Figure 2: Main Menu.

- Click the *play button* in the center to play.
- Click the *continue* button to continue the last played game.
- Click the cup-icon in the bottom right corner to see your highest score.
- Click the *quit game* button the quit the game.

3. Highest score:

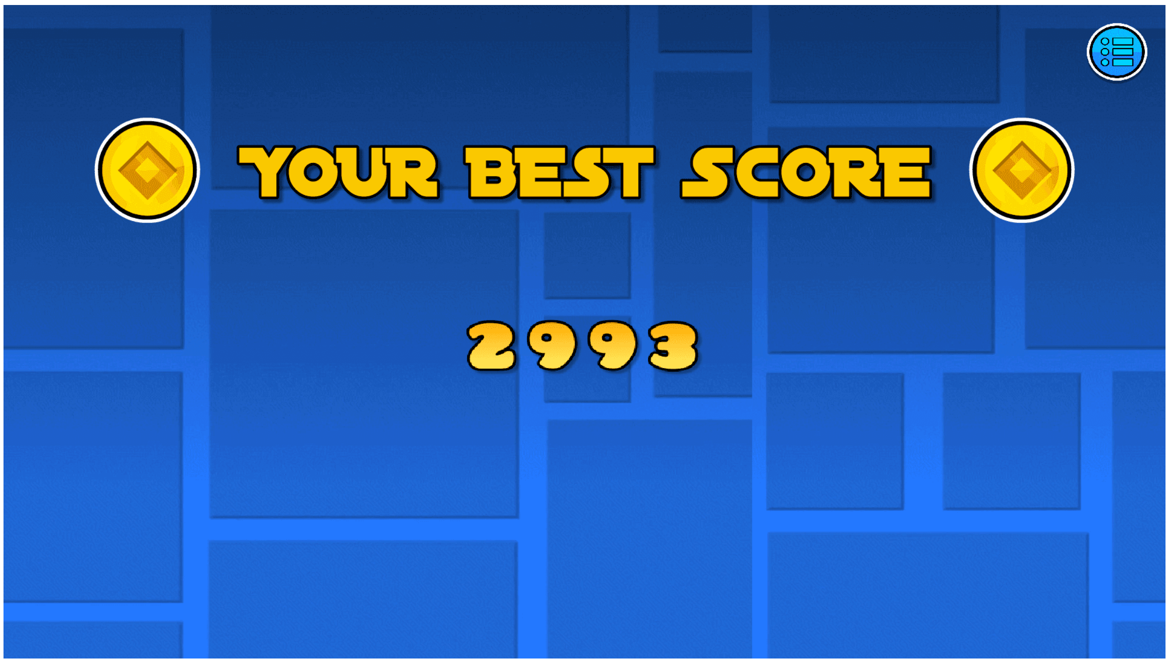

Figure 3: Highest Score Ever.

4. When clicked the *play button* you will get to the Simple Configure Menu:

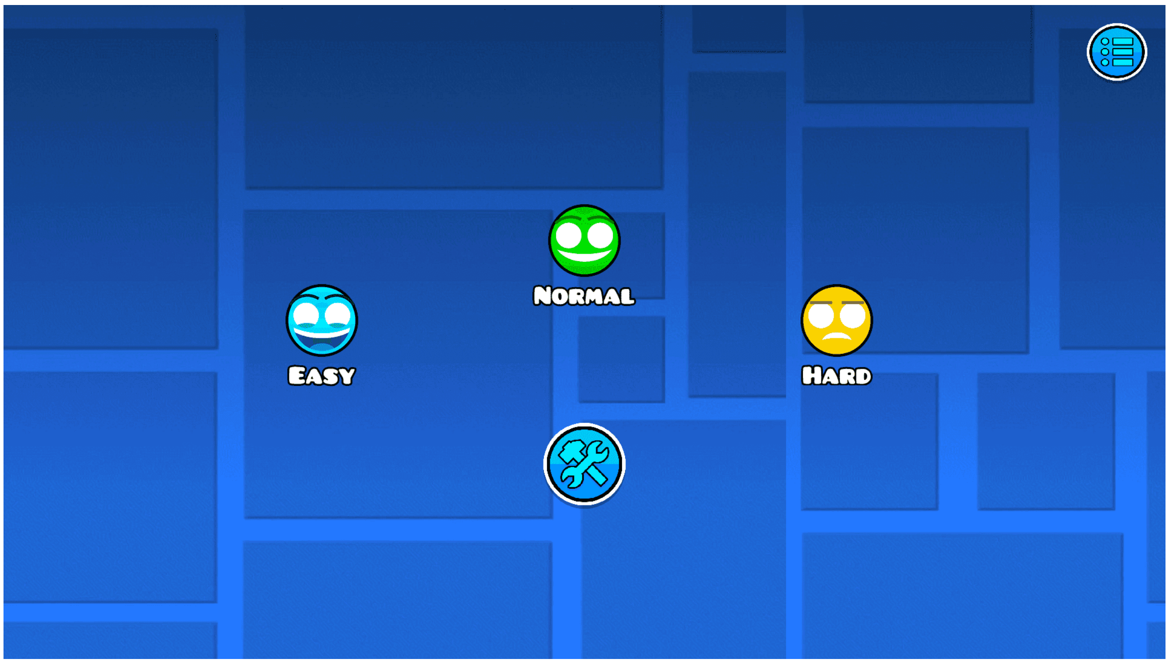

Figure 4: Simple Configure Menu.

- *easy mode*: 9 x 9 grid, 10 mines.
- *normal mode*: 14 x 18 grid, 40 mines.
- *hard mode*: 24 x 24 grid, 99 mines.
- *custom mode*: you can freely adjust grid parameters, and number of mines.

5. When clicked the *custom mode* button you will get to the Custom Menu:

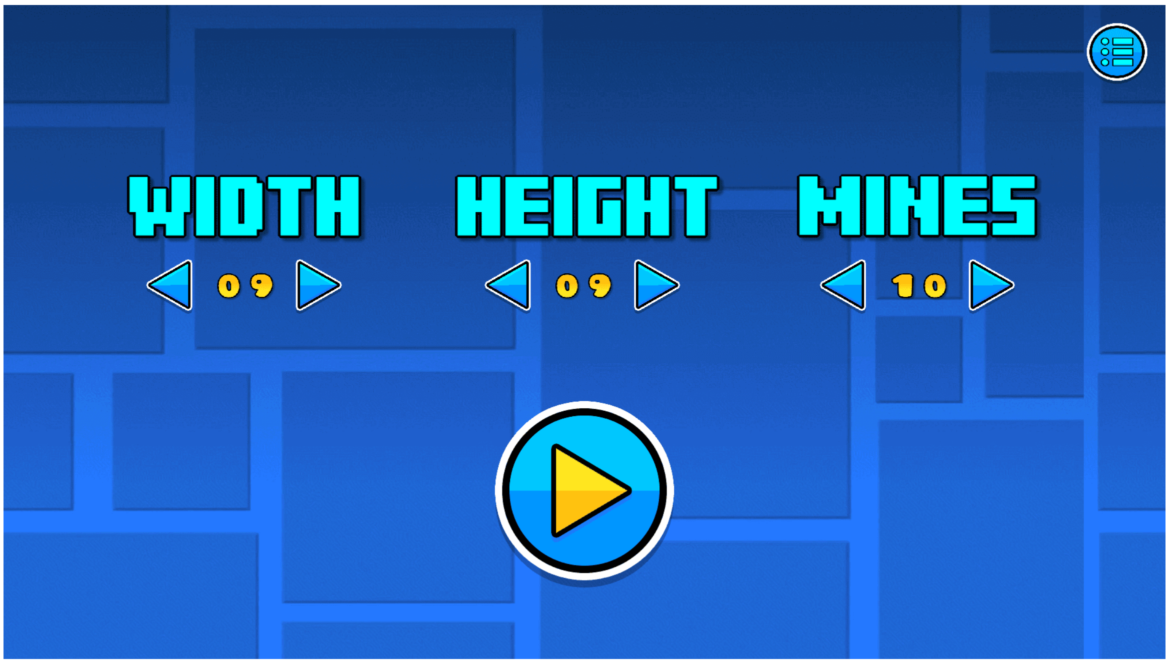

Figure 5: Custom Menu.

- You can press the increase and decrease buttons to change the width, height, and number of mines.
- After that you can press the play button to play.

6. Hover mouse to a cell and press the left-mouse button to open it:

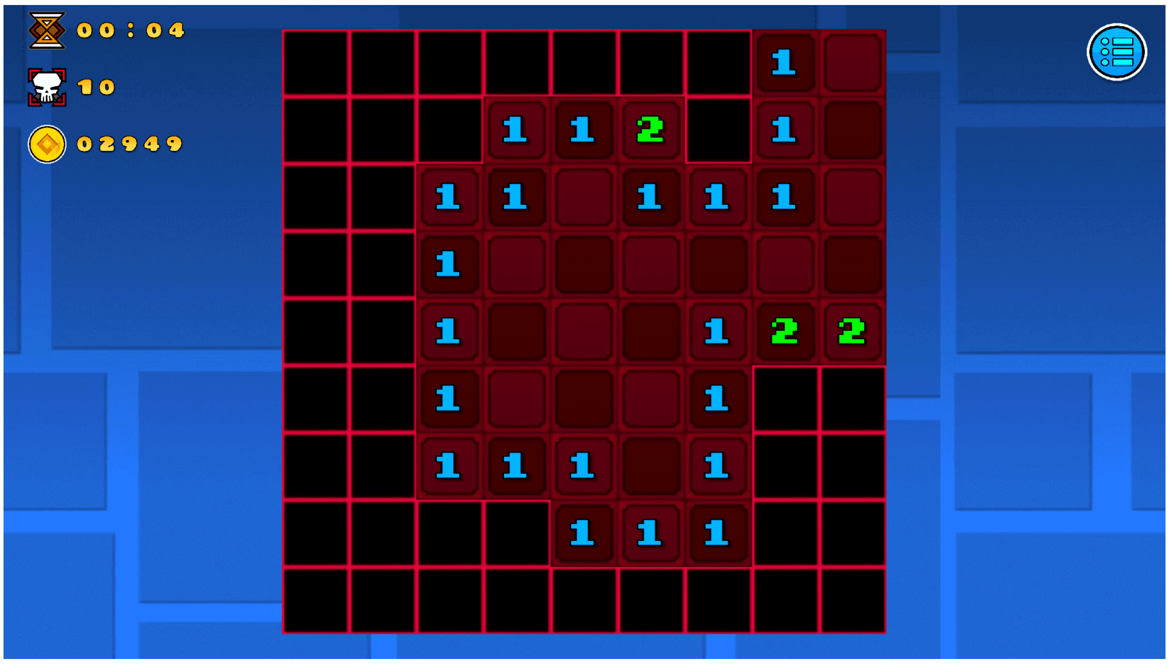

Figure 6: Hover mouse to a cell and press the left-mouse button to open it.

7. Right-click a cell to flag it:

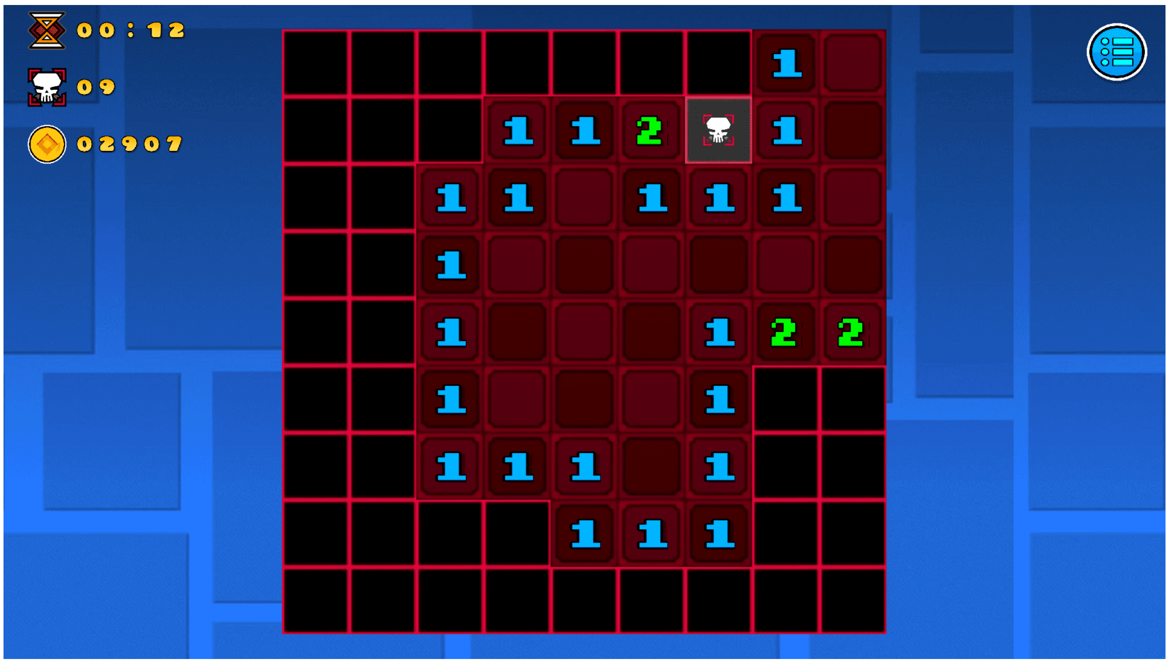

Figure 7: Right-click a cell to flag it.

8. If the number of flags is equal to the number in the current cell, Left-click that numbered cell to chord (open surrounding cells) (Be sure to have flagged correct mines or you will lose):

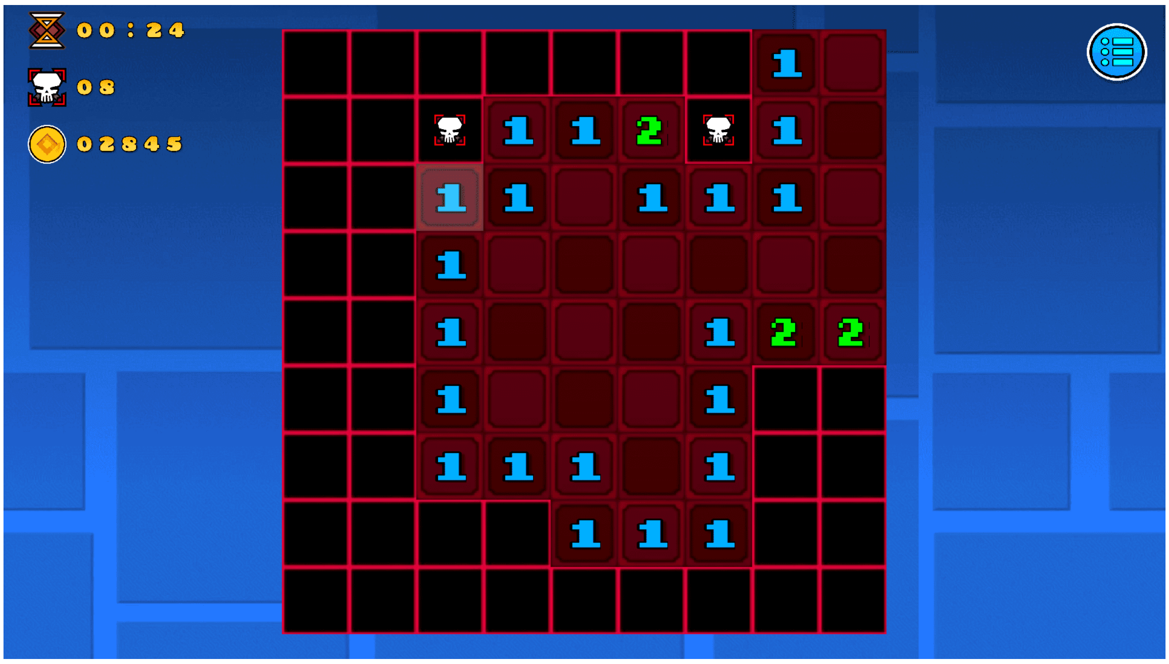

Figure 8: If the number of flags is equal to the number in the current cell, Left-click that numbered cell to chord.

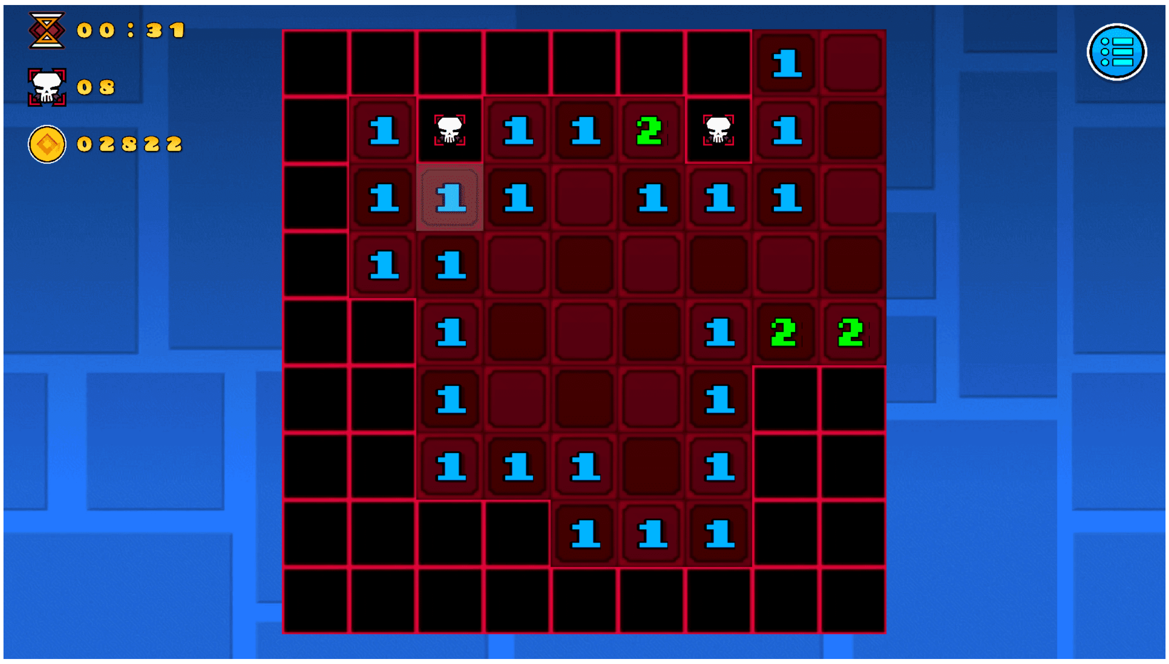

Figure 9: If the number of flags is equal to the number in the current cell, Left-click that numbered cell to chord.

9. Click the icon in the top-right corner to get to the Main Menu (the current game will be automatically saved).
10. You can view the time lapsed, number of mines remaining after flagged, and current score on the top left corner.

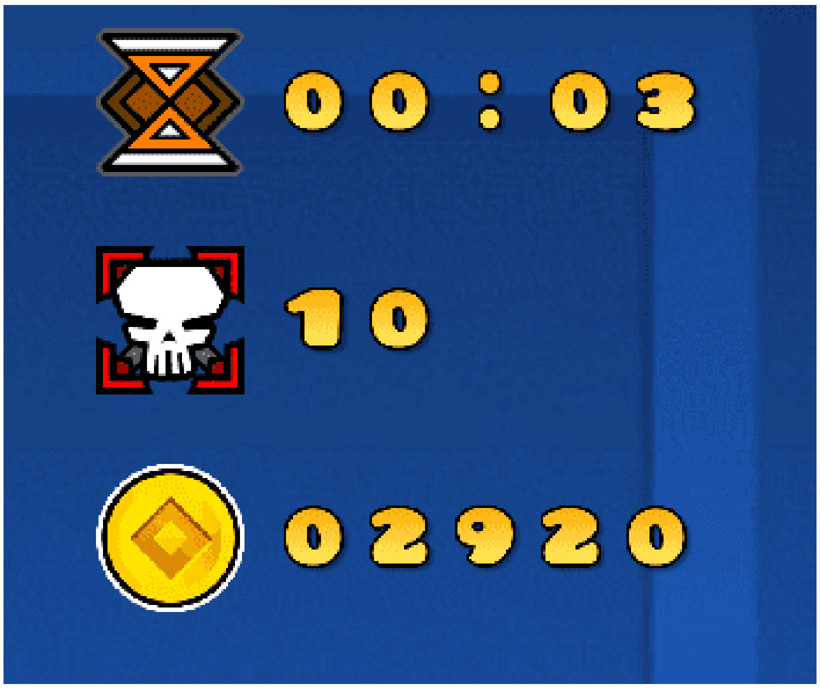

Figure 10: From the top: Time lapsed, number of mines remaining after flagged, current score respectively.

## How to compile the code

1. Download and install Raylib .
2. Download and install Visual Studio Code .
3. Copy /resources folder from /exe folder to /source.
4. Open Visual Studio Code in /source folder and open *minesweeper.exe*.
5. Press F5 (or Run -> Start Debugging) to compile and run the code.