Minesweeper (CS161)

Kiet, Nguyen Hoang The (23125023)
University of Science, VNU-HCM

December 2023

Contents

Page

List of Figures

1 User Manual

1.1 Home Screen

At startup, the user would be introduced by the home screen with a blank minefield on the left and navigation buttons on the right.

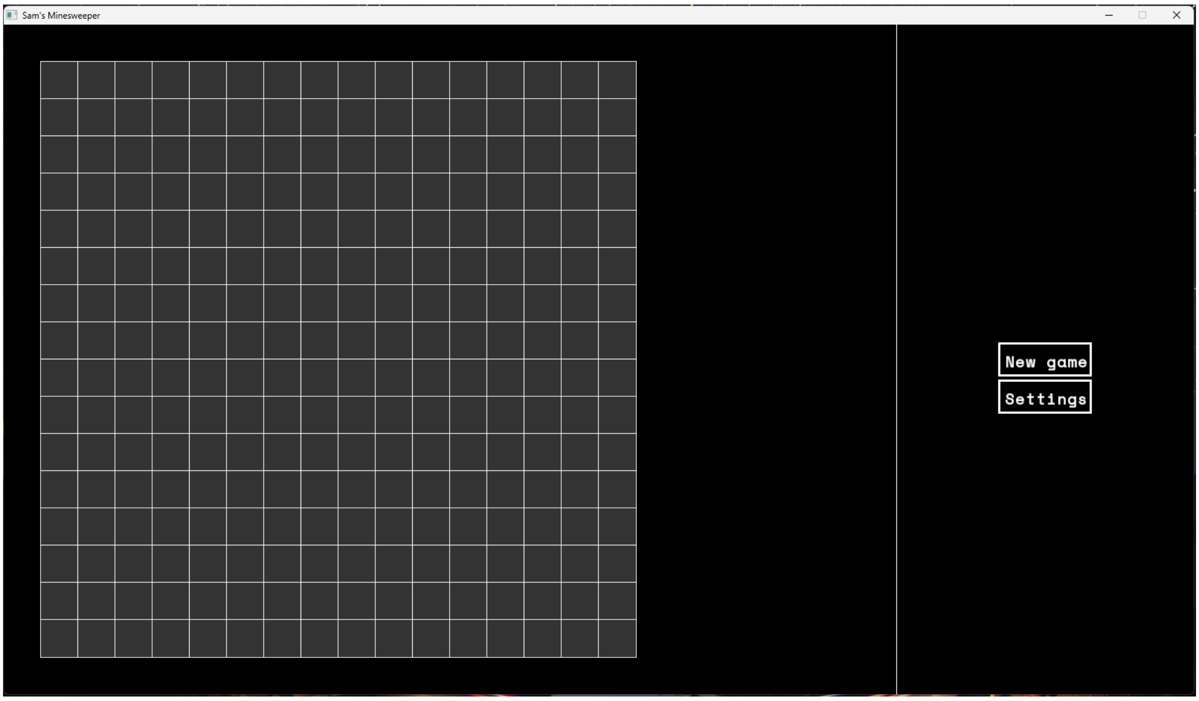


Figure 1.1: Home Screen

Choose *New game* to start a new game, or *Settings* to customise the dimensions and the difficulty of the board.

1.2 Gameplay

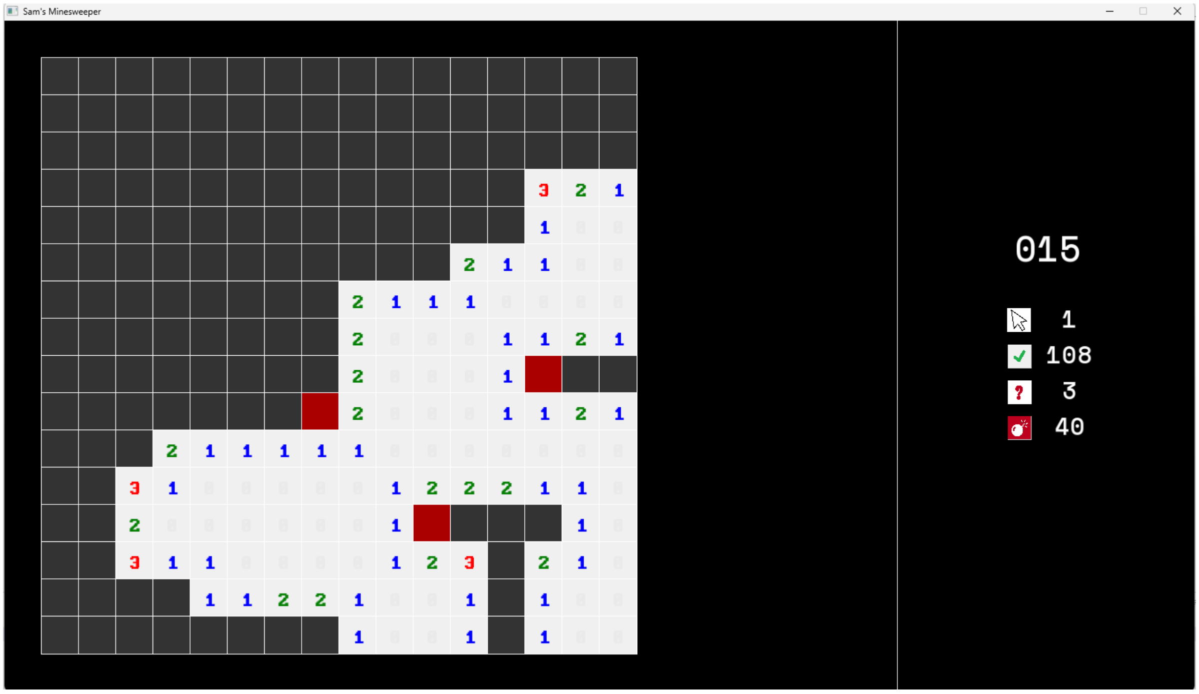


Figure 1.2: An example of a Minesweeper game in medium mode

1.2.1 Objective

- **Board Dimension**. The board consists of $R \times C$ cells arranged into a rectangle of R rows and C columns. By default, $R = 16, C = 16$.
- **Bombs and Bombed Cells**. A specified number (B) of bombs is hidden under some of the cells, with each bomb contained in no more than ONE cell. Those cells are said to be '**bombed**', whilst the other cells are called '**unbombed**'. By default, $B = 40$.
- **States of a Cell**. A cell could be in ONE of the following states:

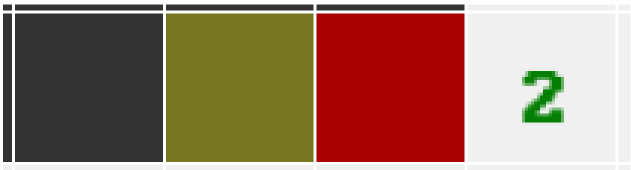


Figure 1.3: Possible cell states: `CLOSED`, `CLOSED` (hovered by mouse cursor), `FLAGGED`, `REVEALED` (unbombed cells, 2 bombs neighbouring)

  - `CLOSED`: the cell is yet to be revealed
  - `FLAGGED`: the cell is suspected to be bombed
  - `REVEALED`: the cell is revealed to be bombed or not; if it is not, then an additional number is also revealed, indicating the number of bombs neighbouring the revealed cell

- **Objective**. The object of the game is to **open as many unbombed cells as possible** without revealing a bombed cell.

1.2.2 Rules

- **Types of Move**. At each stage of the game, the player could either
  - *Flag a closed cell*, if the player believes that cell is bombed; or

  - *Reveal a closed cell*, if the player believes that cell is unbombed.

- **Flagging (Right-click)**. If the player thinks a cell is bombed, they can **flag** the cell by right-clicking on that cell. The cell will turn **red** when the player do so. All bombed cells need to be flagged in order for the game to be won.
- **Revealing (Left-click)**. If the player thinks a cell is unbombed, they can **reveal** the cell by left-clicking on that cell. Either one of the following can happen next:
  - *The chosen cell is bombed,* in which case, the game ends immediately, resulting in a **LOSS** for the player.
  - *The chosen cell is unbombed, but has at least ONE surrounding bomb,* in which case, the cell becomes revealed, with a number representing the count of neighbouring bombs also shown on that cell.
  - *The chosen cell is unbombed, and has NO surrounding bombs,* in which case, the chosen cell is revealed, and the process of revealing a cell is invoked recursively to all neighbouring cells.

- **Revealing Count**. The game counts the number of reveals (left-clicks) in a game. The less reveals, the better the gameplay.
- **Auto Revealing (Left-click on a revealed cell)**. This applies when the player is confident that they have flagged all of the bombed cells surrounding a revealed one, in which case, they can click on the revealed cell to automatically reveal all unflagged neighbouring cells. This move is just a shortcut, and does not save
- **Loss**. The player loses whenever they reveal a bomb.
- **Win**. The player wins when all bombed cells are flagged, and all unbombed cells are revealed.

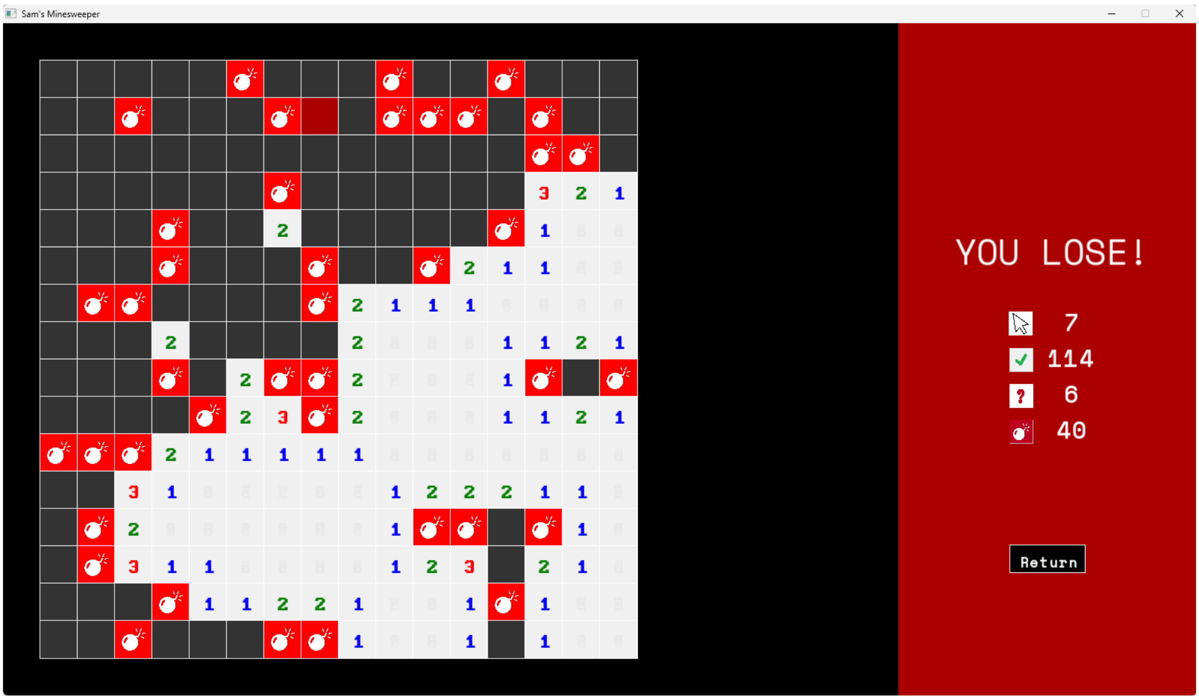
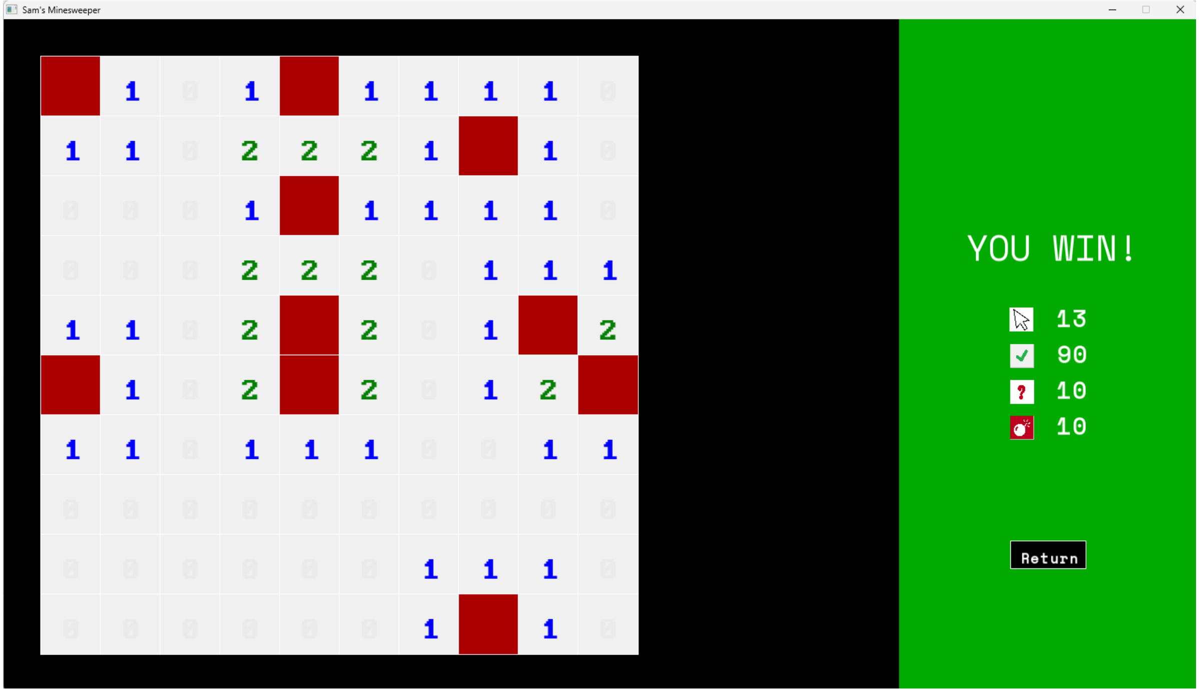


Figure 1.4: Losing screen


Sam's Minesweeper

🟥	1	0	1	🟥	1	1	1	1	0
1	1	0	2	2	2	1	🟥	1	0
0	0	0	1	🟥	1	1	1	1	0
0	0	0	2	2	2	0	1	1	1
1	1	0	2	🟥	2	0	1	🟥	2
🟥	1	0	2	🟥	2	0	1	2	🟥
1	1	0	1	1	1	0	0	1	1
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	1	0
0	0	0	0	0	0	1	🟥	1	0

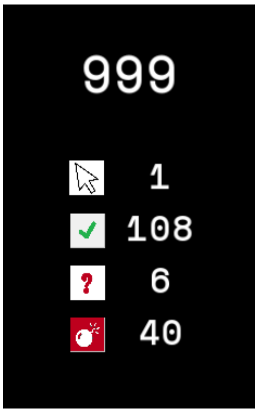
YOU WIN!

↖	13
✓	90
?	10
💣	10

Figure 1.5: Winning screen

1.2.3 Statistics

- **Sidepanel of Stats.** A small panel on the right contains most of the information about the game, including:
  - *Timer*. The number of seconds used in the game. Due to historical convention, the timer stops counting after 999 seconds.
  - *Reveals*. The number of reveals (left-clicks) made.
  - *Opened cells*. The number of cells opened.
  - *Flagged cells*. The number of flagged cells.
  - *Bombs*. The number of bombs.


999

↖	1
✓	108
?	6
💣	40

Figure 1.6: Sidepanel of Stats, from top to bottom: *Timer, Reveals, Opened cells, Flagged cells, Bombs*

1.2.4 Scoring

- **3BV (Bechtel's Board Benchmark Value)** [1]. 3BV is a rough indicator of the difficulty of the board. It computes the optimal number of left-clicks (or reveals) to clear the board. The higher the 3BV, the harder the game. This value is unknown to the player in live play.
- **The Purpose of 3BV**. Due to the high customisability and randomness of the game (dimensional changes, bombs configuration, etc.), there is need for a universal index to measure the difficulty of every board. Many other similar indices[2] also exists, such as 3BV/s, RQP (Rapport Qualité Prix), IOE (Index of Efficiency), etc.

[1] http://www.stephan-bechtel.de/3bv.htm

[2] https://minesweepergame.com/statistics.php

- **Scoring**. This project uses a custom scoring method, determined by the following function:

$$\text{Score} = \begin{cases} \dfrac{1}{\log(1+C)} \cdot \dfrac{1}{1+(\frac{1}{r}-1)^2} & \text{if } r < 1 \\[2ex] \dfrac{1}{\log(1+C)} \cdot \Big(\dfrac{0.75}{1-M}(r-1)+1\Big) & \text{if } r \geq 1 \end{cases}$$

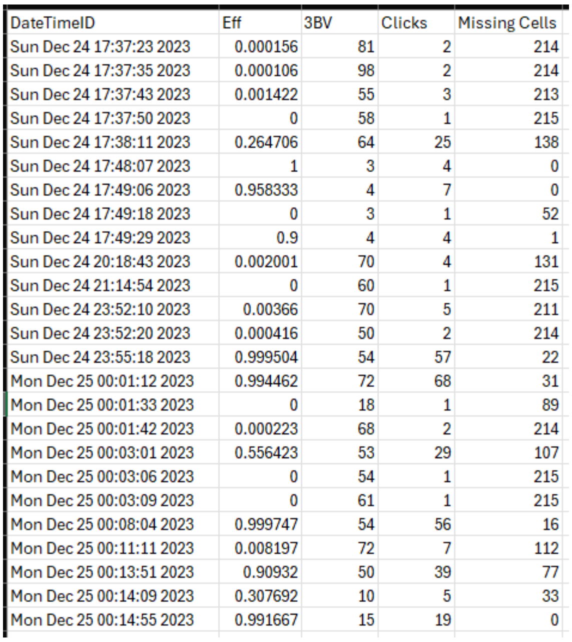
$$\text{where } r = \frac{\text{number of reveals} - 1}{3\text{BV}};$$

$$M = \text{the number of maximum reveals to clear the board;}$$

$$C = \text{the number of unopened cells.}$$

- **Scoring Methodology**. This scoring method is to:
  - *Encourage perfect or near-perfect games*: less unopened cells means more points; and
  - *Encourage optimal games*: more points are awarded if the player plays as optimal as the 3BV method

- **Leaderboard**. The history of played games can be viewed in `data/leaderboard.csv`.



DateTimeID	Eff	3BV	Clicks	Missing Cells
Sun Dec 24 17:37:23 2023	0.000156	81	2	214
Sun Dec 24 17:37:35 2023	0.000106	98	2	214
Sun Dec 24 17:37:43 2023	0.001422	55	3	213
Sun Dec 24 17:37:50 2023	0	58	1	215
Sun Dec 24 17:38:11 2023	0.264706	64	25	138
Sun Dec 24 17:48:07 2023	1	3	4	0
Sun Dec 24 17:49:06 2023	0.958333	4	7	0
Sun Dec 24 17:49:18 2023	0	3	1	52
Sun Dec 24 17:49:29 2023	0.9	4	4	1
Sun Dec 24 20:18:43 2023	0.002001	70	4	131
Sun Dec 24 21:14:54 2023	0	60	1	215
Sun Dec 24 23:52:10 2023	0.00366	70	5	211
Sun Dec 24 23:52:20 2023	0.000416	50	2	214
Sun Dec 24 23:55:18 2023	0.999504	54	57	22
Mon Dec 25 00:01:12 2023	0.994462	72	68	31
Mon Dec 25 00:01:33 2023	0	18	1	89
Mon Dec 25 00:01:42 2023	0.000223	68	2	214
Mon Dec 25 00:03:01 2023	0.556423	53	29	107
Mon Dec 25 00:03:06 2023	0	54	1	215
Mon Dec 25 00:03:09 2023	0	61	1	215
Mon Dec 25 00:08:04 2023	0.999747	54	56	16
Mon Dec 25 00:11:11 2023	0.008197	72	7	112
Mon Dec 25 00:13:51 2023	0.90932	50	39	77
Mon Dec 25 00:14:09 2023	0.307692	10	5	33
Mon Dec 25 00:14:55 2023	0.991667	15	19	0

Figure 1.7: An example of `leaderboard.csv`

1.3 Settings

1.3.1 In-app Settings

- **Presets**. Three predetermined configurations are given for the user to choose from:

Mode	Board size	# of bombs
Easy	10×10	10
Medium	16×16	40
Hard	40×30	200

- **Custom Dimension Change**. The user can click on `R+` or `R-` to increase or decrease the number of rows, respectively. Similarly, the user can click on `C+` or `C-` to increase or decrease the number of columns, respectively.
  Note: in case if the board does not fit the window (for example, the board is too wide), it may resize itself by continuously reducing the number of columns until fits.)
- **Custom Bomb Count**. The user can increase or decrease the number of bombs on the board by clicking on, respectively, `B+` or `B-`. A preview of the bombs distribution is shown on the left.
- **Temporariness of In-app Changes**. In-app settings changes are temporary. They are not stored externally and will be reset after application relaunch.

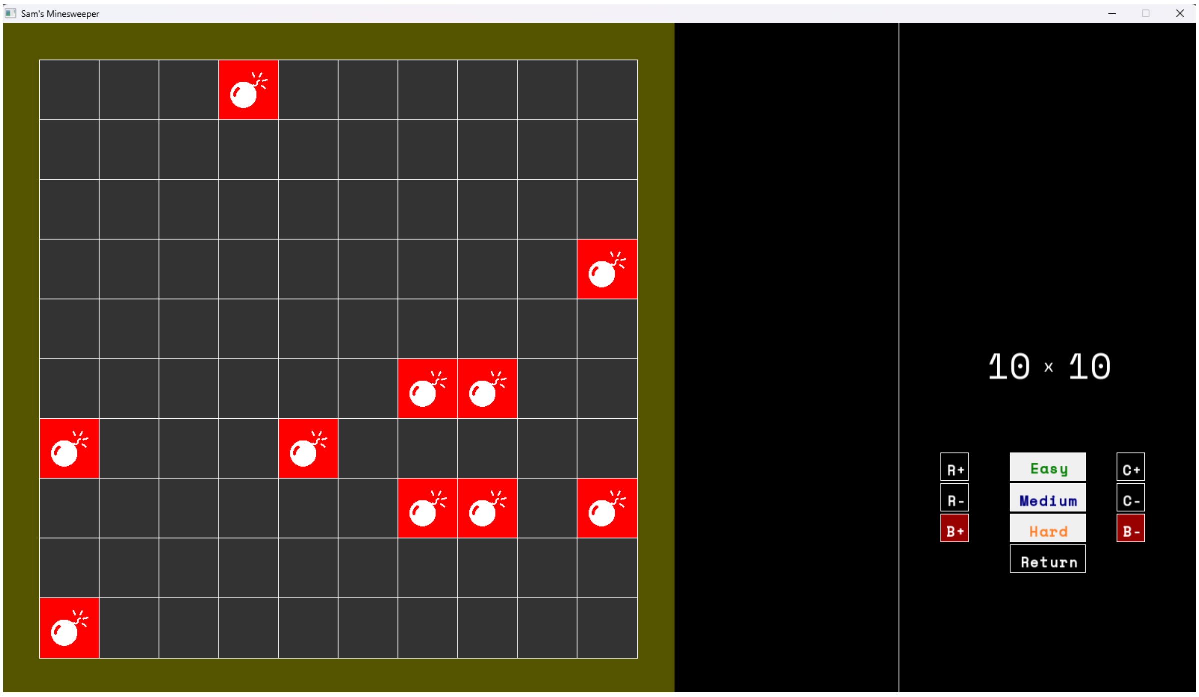


Figure 1.8: Easy board (10 × 10), 10 bombs

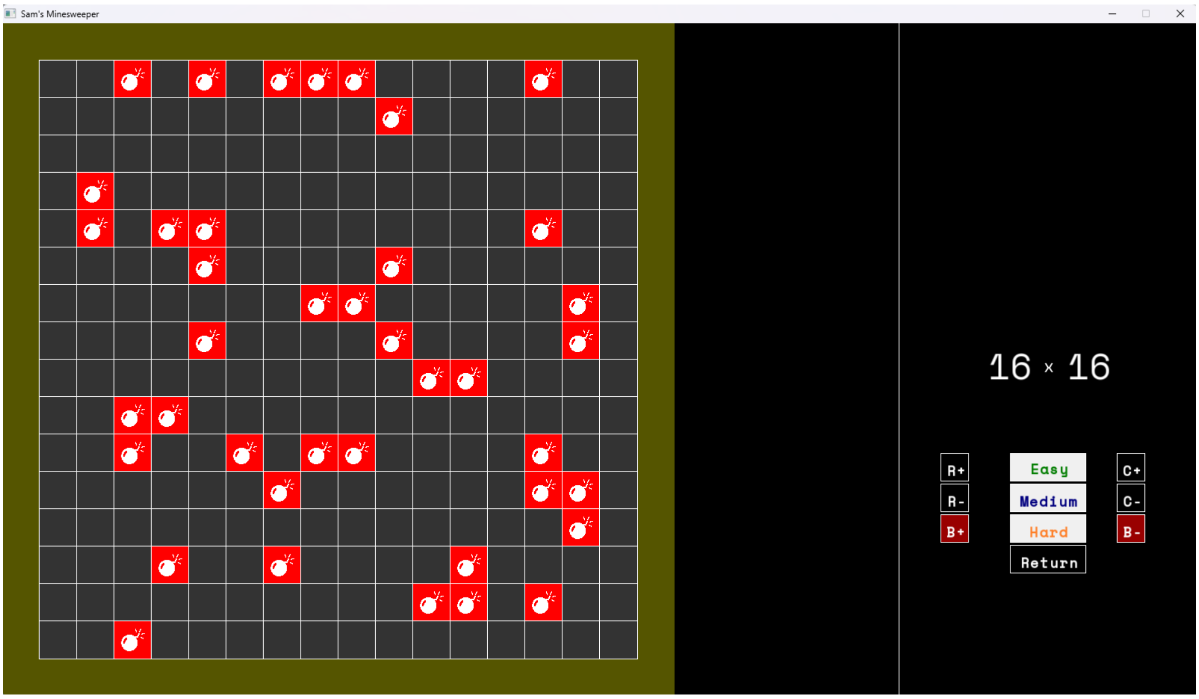


Figure 1.9: Medium board (16 × 16), 40 bombs

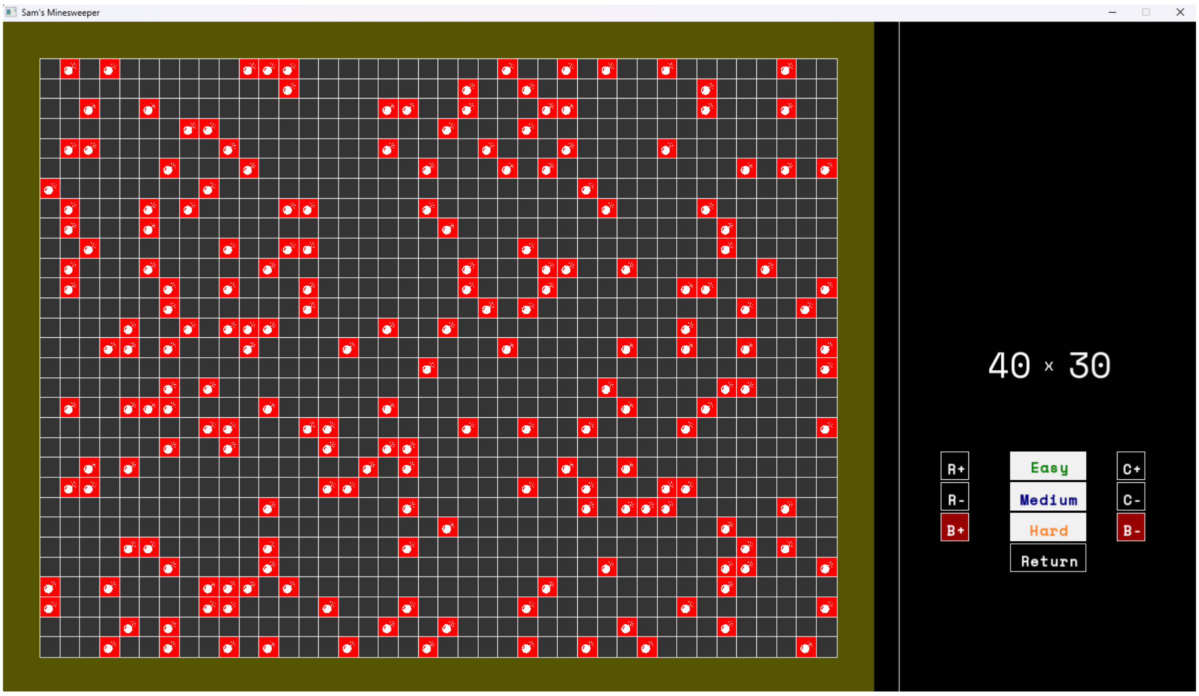


Figure 1.10: Hard board (40 × 30), 200 bombs

1.3.2 Custom Settings

- **Config File (`data/config.dat`)**. The application will read a single config file before launch. This config file includes all of the application settings.
- **Modifying the Config File**. The user can modify the config file to their likings, with notes that
  - *Syntax*: The config file is read line-by-line, each line must follow the `<VARIABLE> = <LITERAL>` syntax. Spaces/Tabs are omitted.
  - *Literal types*: A literal can be either a) an integer, consists of a string of digit-characters, b) a string of character, enclosed in quotes. or c) a hex code, consists of one hash character and 6 or 8 hex characters, usually to describe a colour code.
  - *Variable constraints*: Any variables in the config file should not be removed from the original file. Moreover, the type of any variables should not be altered. The program may crash otherwise.
  - *Comments*: Any line starting with a hash `'#'` character is considered a comment (similar to Python), and is omitted from reading.

```
BOARD_BOMBS       = 40
COLOUR_CELLS[1]   = #0100FB
FONT_TEXT_PATH    = "assets/fonts/SpaceMono-Regular.ttf"
```

Examples from the default config file.

2 Project Structure

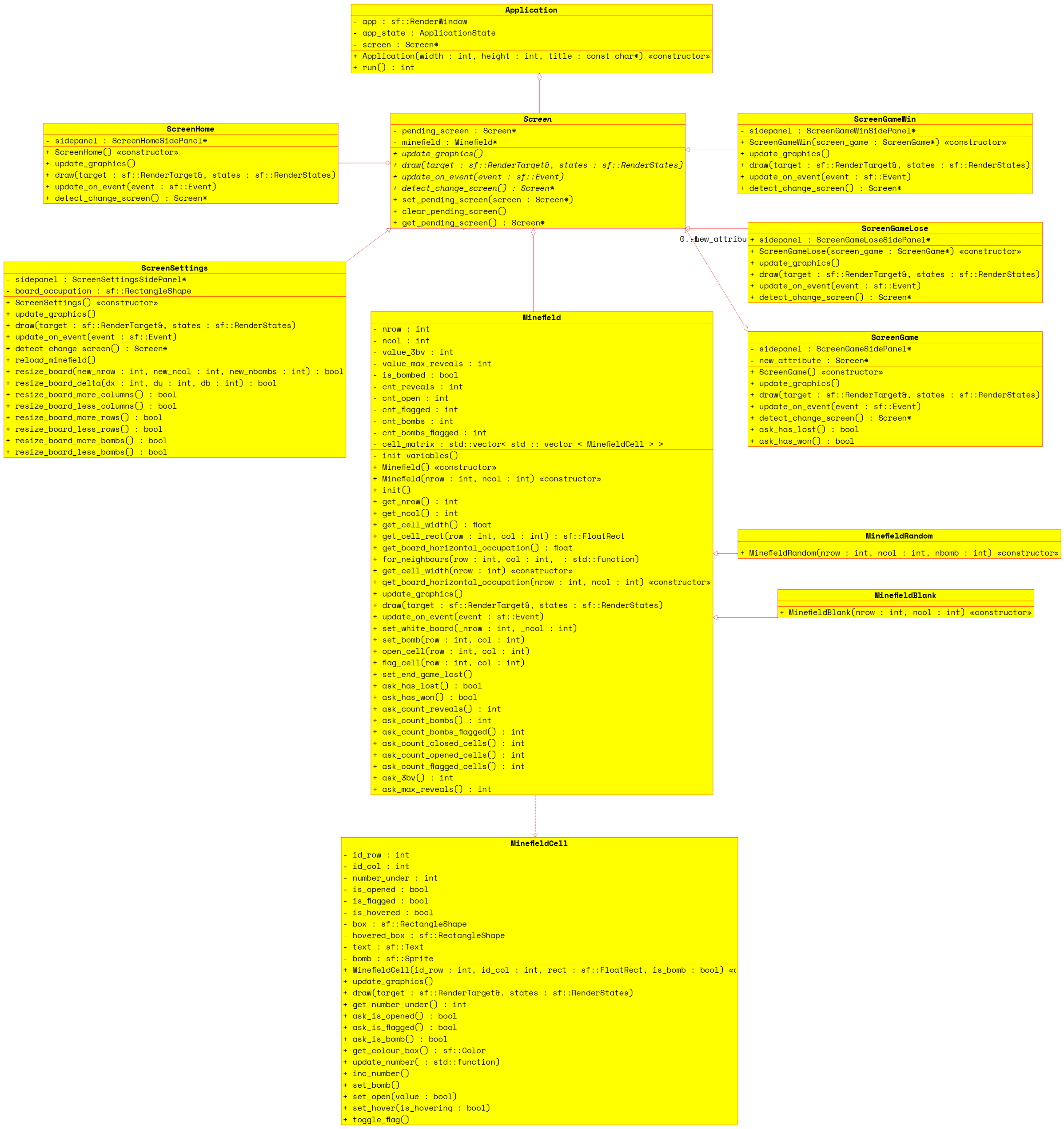

Application
- app : sf::RenderWindow
- app_state : ApplicationState
- screen : Screen*
+ Application(width : int, height : int, title : const char*) «constructor»
+ run() : int
ScreenHome
- sidepanel : ScreenHomeSidePanel*
+ ScreenHome() «constructor»
+ update_graphics()
+ draw(target : sf::RenderTarget&, states : sf::RenderStates)
+ update_on_event(event : sf::Event)
+ detect_change_screen() : Screen*
Screen
- pending_screen : Screen*
- minefield : Minefield*
+ update_graphics()
+ draw(target : sf::RenderTarget&, states : sf::RenderStates)
+ update_on_event(event : sf::Event)
+ detect_change_screen() : Screen*
+ set_pending_screen(screen : Screen*)
+ clear_pending_screen()
+ get_pending_screen() : Screen*
ScreenGameWin
- sidepanel : ScreenGameWinSidePanel*
+ ScreenGameWin(screen_game : ScreenGame*) «constructor»
+ update_graphics()
+ draw(target : sf::RenderTarget&, states : sf::RenderStates)
+ update_on_event(event : sf::Event)
+ detect_change_screen() : Screen*
0..1
new_attribu
ScreenGameLose
+ sidepanel : ScreenGameLoseSidePanel*
+ ScreenGameLose(screen_game : ScreenGame*) «constructor»
+ update_graphics()
+ draw(target : sf::RenderTarget&, states : sf::RenderStates)
+ update_on_event(event : sf::Event)
+ detect_change_screen() : Screen*
ScreenSettings
- sidepanel : ScreenSettingsSidePanel*
- board_occupation : sf::RectangleShape
+ ScreenSettings() «constructor»
+ update_graphics()
+ draw(target : sf::RenderTarget&, states : sf::RenderStates)
+ update_on_event(event : sf::Event)
+ detect_change_screen() : Screen*
+ reload_minefield()
+ resize_board(new_nrow : int, new_ncol : int, new_nbombs : int) : bool
+ resize_board_delta(dx : int, dy : int, db : int) : bool
+ resize_board_more_columns() : bool
+ resize_board_less_columns() : bool
+ resize_board_more_rows() : bool
+ resize_board_less_rows() : bool
+ resize_board_more_bombs() : bool
+ resize_board_less_bombs() : bool
Minefield
- nrow : int
- ncol : int
- value_3bv : int
- value_max_reveals : int
- is_bombed : bool
- cnt_reveals : int
- cnt_open : int
- cnt_flagged : int
- cnt_bombs : int
- cnt_bombs_flagged : int
- cell_matrix : std::vector< std :: vector < MinefieldCell > >
- init_variables()
+ Minefield() «constructor»
+ Minefield(nrow : int, ncol : int) «constructor»
+ init()
+ get_nrow() : int
+ get_ncol() : int
+ get_cell_width() : float
+ get_cell_rect(row : int, col : int) : sf::FloatRect
+ get_board_horizontal_occupation() : float
+ for_neighbours(row : int, col : int, : std::function)
+ get_cell_width(nrow : int) «constructor»
+ get_board_horizontal_occupation(nrow : int, ncol : int) «constructor»
+ update_graphics()
+ draw(target : sf::RenderTarget&, states : sf::RenderStates)
+ update_on_event(event : sf::Event)
+ set_white_board(_nrow : int, _ncol : int)
+ set_bomb(row : int, col : int)
+ open_cell(row : int, col : int)
+ flag_cell(row : int, col : int)
+ set_end_game_lost()
+ ask_has_lost() : bool
+ ask_has_won() : bool
+ ask_count_reveals() : int
+ ask_count_bombs() : int
+ ask_count_bombs_flagged() : int
+ ask_count_closed_cells() : int
+ ask_count_opened_cells() : int
+ ask_count_flagged_cells() : int
+ ask_3bv() : int
+ ask_max_reveals() : int
ScreenGame
- sidepanel : ScreenGameSidePanel*
- new_attribute : Screen*
+ ScreenGame() «constructor»
+ update_graphics()
+ draw(target : sf::RenderTarget&, states : sf::RenderStates)
+ update_on_event(event : sf::Event)
+ detect_change_screen() : Screen*
+ ask_has_lost() : bool
+ ask_has_won() : bool
MinefieldRandom
+ MinefieldRandom(nrow : int, ncol : int, nbomb : int) «constructor»
MinefieldBlank
+ MinefieldBlank(nrow : int, ncol : int) «constructor»
MinefieldCell
- id_row : int
- id_col : int
- number_under : int
- is_opened : bool
- is_flagged : bool
- is_hovered : bool
- box : sf::RectangleShape
- hovered_box : sf::RectangleShape
- text : sf::Text
- bomb : sf::Sprite
+ MinefieldCell(id_row : int, id_col : int, rect : sf::FloatRect, is_bomb : bool) «c
+ update_graphics()
+ draw(target : sf::RenderTarget&, states : sf::RenderStates)
+ get_number_under() : int
+ ask_is_opened() : bool
+ ask_is_flagged() : bool
+ ask_is_bomb() : bool
+ get_colour_box() : sf::Color
+ update_number(: std::function)
+ inc_number()
+ set_bomb()
+ set_open(value : bool)
+ set_hover(is_hovering : bool)
+ toggle_flag()

Figure 2.1: Main code structure

This project is structured in an Object Oriented Programming (OOP) style.

2.1 Choice of graphics library

This project utilises the SFML library[3] for graphics. One of its biggest advantages is that SFML is designed with OOP in mind, thus it is easier to apply to our project, and to also extend the library to our usage.

[3] .https://www.sfml-dev.org/

2.2 Application class

```
class Application {
private:
    sf::RenderWindow    app;
    Screen*             screen;

public:
    Application(int width, int height, const char* title);
    int run();
};
```

The Application class consists of

- sf::RenderWindow app: graphics window
- Screen* screen: pointer to Screen object that handles all graphics and events in a specific state.

The universal game flow is defined in Application::run():

```
int Application::run() {
    while (app.isOpen()) {
        // Event polling
        sf::Event event;
        while (app.pollEvent(event)) {
            if (event.type == sf::Event::Closed) {
                app.close();
                return 0;
            }
            screen->update_on_event(event);
        }

        // State switching
        app.clear();
        Screen* nxt_screen = screen->detect_change_screen();
        screen = nxt_screen;

        // Graphics drawing
        screen->update_graphics();
        app.draw(*screen);
        app.display();
    }
}
```

2.3 Screen classes

```
class Screen : public sf::Drawable, public sf::Transformable, {
private:
    Screen      *pending_screen = nullptr;
    Minefield   *minefield;

public:
    virtual void    update_graphics() = 0;
    virtual void    draw(sf::RenderTarget &target,
                             sf::RenderStates states) const = 0;
    virtual void    update_on_event(sf::Event event) = 0;
    virtual Screen* detect_change_screen() = 0;

    void        set_pending_screen(Screen *screen);
    void        clear_pending_screen();
    Screen*     get_pending_screen();
};

    friend class ScreenHome;
    friend class ScreenSettings;
    friend class ScreenGame;
    friend class ScreenGameLose;
    friend class ScreenGameWin;
}
```

The Screen class is a base class that contains

- Minefield* minefield: pointer to the Minefield object.
- Screen* pending_screen: pointer to the subsequent Screen object. If pending_screen is not nullptr, the application should switch to a different state in the next frame.

The functionality of this base class is to handle state switching within the application. If there is state switching, the application first register the newly-created screen into pending_screen, then at the end of the game loop, Application::screen will be updated accordingly.

ScreenHome, ScreenSettings, ScreenGame, ScreenGameLose ScreenGameWin are all child classes of Screen, therefore automatically inherit the state-switching functionality. They are also friends of Screen so that they can get direct access to minefield.

Each class also has a corresponding SidePanel class that handles common user control like button clicks. The parent Screen class would ask for information from the SidePanel class.

All Screens classes are defined in screen.hpp, while all SidePanels classes are defined in sidepanel.hpp

2.4 Minefield class

This class is to store all information about the game board, and to handle events from the user.

```
    class Minefield : public sf::Drawable, public sf::Transformable
    {
    private:
        int         nrow, ncol;
        int         value_3bv, value_max_reveals;
        bool        is_bombed;

        int         cnt_reveals;
        int         cnt_open;
        int         cnt_flagged;
        int         cnt_bombs;
        int         cnt_bombs_flagged;

        std::vector<std::vector<MinefieldCell>> cell_matrix;

        // ...
    }
```

The `Minefield` class contains

- `cell_matrix`: Matrix of `MinefieldCell`.
- Other variables represent different states of the board. They are computed/updated immediately after each user event that triggers a change of the board, thus keeping querying information about the board efficient.

The `MinefieldCell` class contains

- Data corresponding to one single cell of the board: `id_row`, `id_col`, `number_under` ($= -1$ if bombed, $\in [0, 8]$ otherwise) and other states.
- SFML graphics objects. These are initiated once throughout the entirety of one game.

```
class MinefieldCell : public sf::Drawable, public sf::Transformable
{
private:
    int     id_row, id_col;
    int     number_under;
    bool    is_opened, is_flagged, is_hovered;

    sf::RectangleShape  box;
    sf::RectangleShape  hovered_box;
    sf::Text            text;
    sf::Sprite          bomb;

    // ...
}
```

Both classes are defined in `minefield.hpp`

2.5 `Context` and `ContextReader` class

`Context` class stores all global variables in a single global entity: `Context context`. Other objects can call by simply access the variables in the `context` object.

`ContextReader` class reads the `data/config.dat` file, then interprets it into maps of correlations.

```
class ContextReader {
private:
    std::map<std::string, int> mp_int;
    std::map<std::string, unsigned int> mp_hex;
    std::map<std::string, std::string> mp_str;

public:
    ContextReader();
    ContextReader(const std::string& filename);

    void report_error();
    void load_file(const std::string& filename);
    void load_file(const char* filename);

    int get_int(const std::string& token);
    unsigned int get_hex(const std::string& token);
    std::string get_str(const std::string& token);
};
```

3 Future Improvements

We list here some possible features that can be implemented into our Minesweeper project.

3.1 Gameplay

- **First Move Guarantee**. The user should not lose on the opening move. One approach is if the user hit a bomb on the opening move, that bomb is transferred to a different cell. This approach is also used on later versions of Windows Minesweeper.
- **Load and Save a Session**. The user would be able to save a game for future play. It could also be implemented automatically, so that when the user closed the program or it crashed in the middle of a game, the user would be able to recover the game that they were playing.

3.2 Scoring

- **In-app Viewable Leaderboard**. The user would be able to open the leaderboard within the app. Currently the user has to use an external program to open `data/leaderboard.csv` to view the leaderboard.

3.3 Settings

- **JSON-style Config File**. The user would be able to change the config file within the app. For that, the program should be able to read a JSON-style config file (`config.json`). This could be done via JSON-reader libraries written in C++. An instance would be JSON for Modern C++ by nlohmann[4].

[4] JSON for Modern C++ by nlohmann: https://github.com/nlohmann/json