CS161 Project: Mongusweeper

Author: 22125113 - Nguyen Quang Truong

Repository: <https://github.com/jalsol/cs161-minesweeper>

A “sussy-wussy” Among-Us-themed Minesweeper clone.

# Requirements

* C++20 (GNU GCC 11.3.0)
* CMake 3.14 or above
* [Raylib](https://github.com/raysan5/raylib)
* [Raygui](https://github.com/raysan5/raygui) (already included in the repository)
* [yaml-cpp](https://github.com/jbeder/yaml-cpp)
* (optional, preferred) A Linux environment

# Building

* Clone this repository
* Install [Raylib on your platform](https://github.com/raysan5/raylib#build-and-installation)
* Install [yaml-cpp on your platform](https://github.com/jbeder/yaml-cpp/blob/master/install.txt)
* Run ”cmake -S . -B ./build“
* Run “make -C build”
* The executable “mongusweeper” will appear

# Features

* “Creative and enjoyable” meme theme (including looks and sounds!)
* Customizable board dimensions and number of bombs
* Saved previous game
* Saved high score
* …And more waiting for you to discover (if my lecturers and teaching assistants are reading this, I swear I have fulfilled every important requirement)!

# License

This project is licensed under Apache License 2.0. Dependencies are licensed under their respective licenses.

Due to the usage of the “Among Us IP”, this project also follows the [Fan Creation Policy](https://www.innersloth.com/fan-creation-policy/) by Innersloth LLC. This policy is preferred over the project’s license and the dependencies’ licenses, in case any conflict occurs.

# Usage



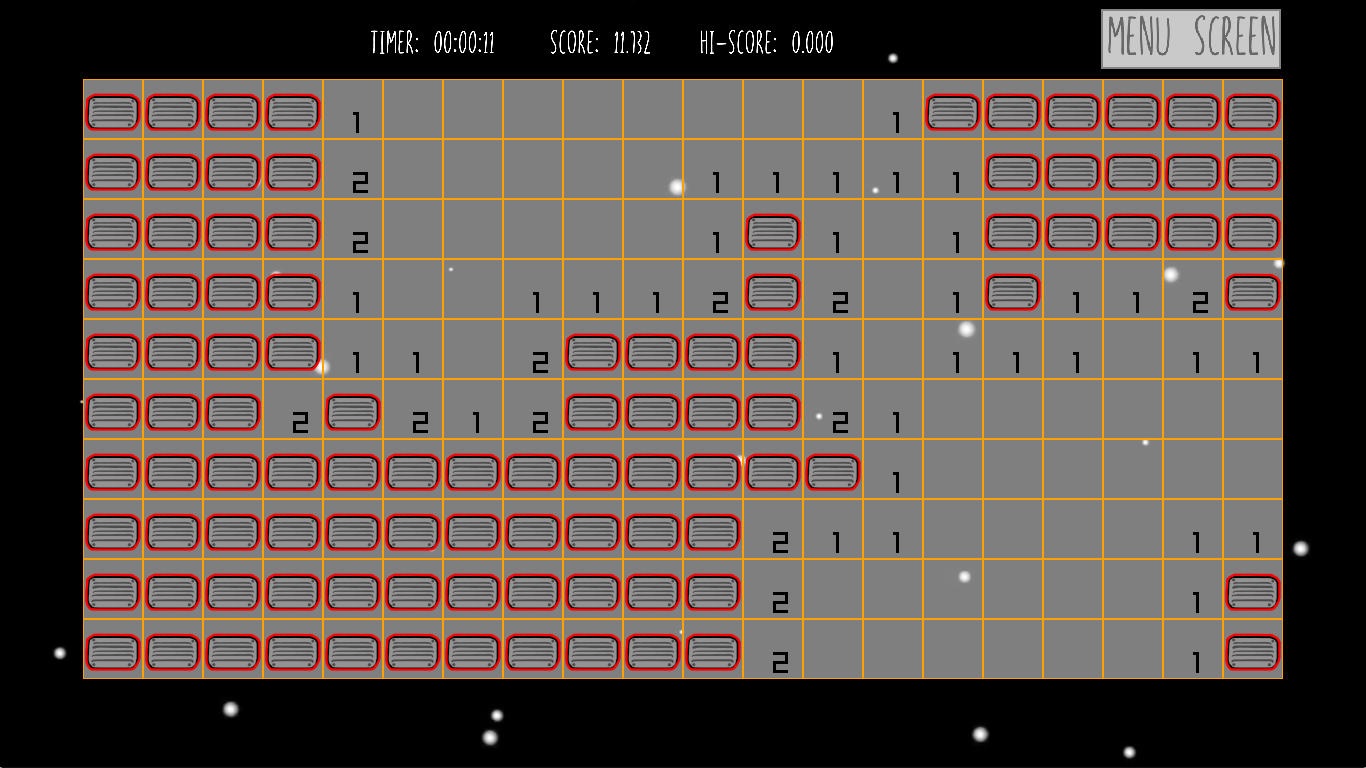
In the menu screen, there are 3 options:

* New Game: start a new game
* Continue: continue from a saved game (if existed)
* Settings: configure settings for the game



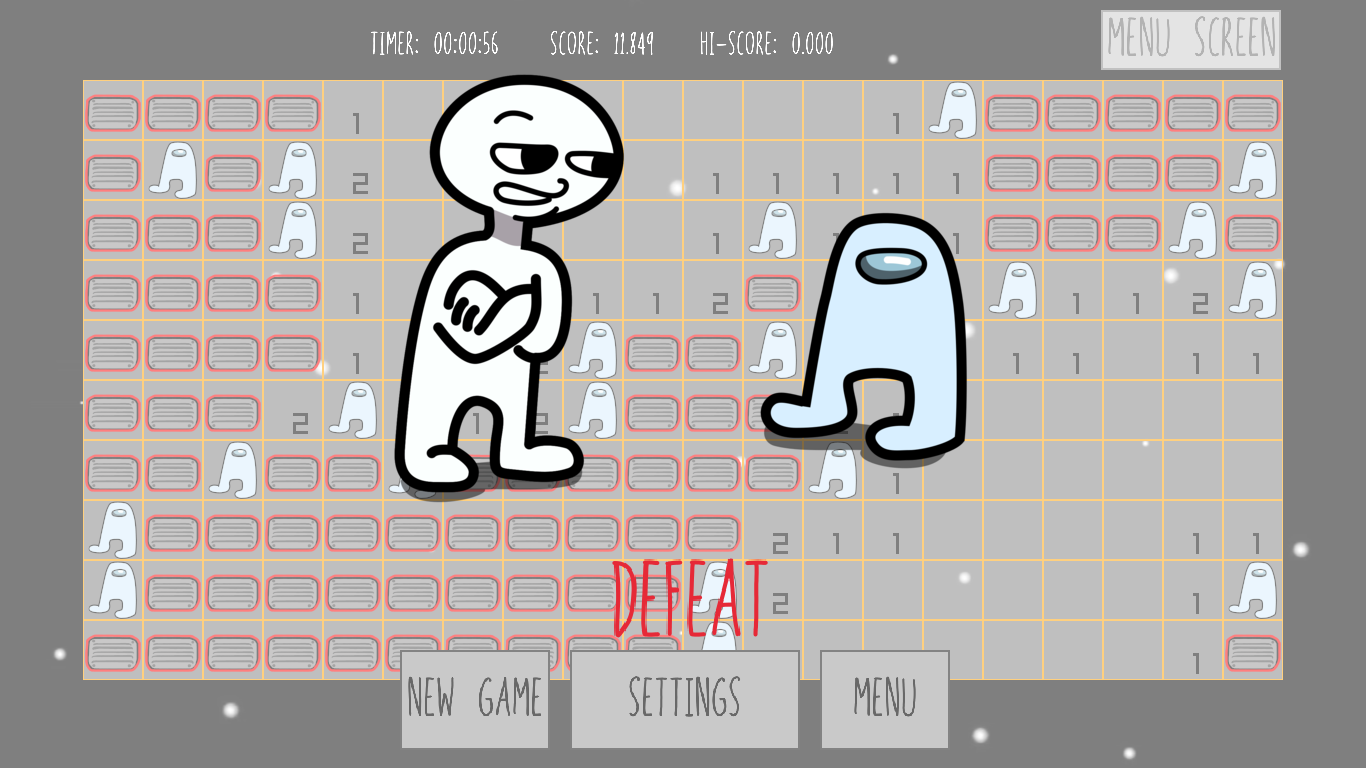
In the settings screen, there are 3 modifiable values:

* Width: the width of the game table
* Height: the height of the game table
* Bombs: the number of bombs on the game table



In the gameplay screen, it’s your sussy minesweeper gameplay:

* The game table
* Timer
* Score and High score
* A button to save the current game and go back to the menu screen



The victory screen is on the left. The defeat screen is on the right.

# Video demonstration

<https://www.youtube.com/watch?v=W1Rf-jkQfd0>

# 

# Documentation

## enum class CellState

(defined in cell.h)

| **enum** **class** **CellState** {  Closed,  Opened,  Flagged, }; |
| --- |

The value of enum class CellState tells the current state of a cell (closed, open, flagged).

## struct Cell

(defined in cell.h)

Represents a cell in the table of the game.

| Variable | Default value | Description |
| --- | --- | --- |
| static constexpr int  bomb\_cell\_value | -1 | The value of the cell if it’s a bomb |
| static constexpr int  cell\_size | 60 | The size of the rendered cell (in pixels) |
| static constexpr int  font\_size | 30 | The size of the text inside the cell (in pt) |
| int  m\_value | 0 | The value of the cell, either bomb\_cell\_value (if it’s a bomb) or the number of bomb cells next to it |
| double  m\_screen\_pos\_x | 0 | The x-coordinate of the top-left pixel of the rendered cell |
| double  m\_screen\_pos\_y | 0 | The y-coordinate of the top-left pixel of the rendered cell |
| CellState  m\_cell\_state | Closed | The current state of the cell |

| Function | Description |
| --- | --- |
| Cell() | Constructs a new Cell |
| void  setValue(int value) | Assigns value to m\_value |
| int  getValue() const | Returns the value of m\_value |
| CellState  getCellState() const | Returns the value of m\_cell\_state |
| void  setCellScreenPos(int x, int y) | Assigns x to m\_screen\_pos\_x and  assigns y to m\_screen\_pos\_y |
| void  drawCell() | Draws the cell on the screen |
| bool  reveal() | - Reveals the cell  - Returns true if the cell is a bomb, otherwise false |
| void  toggleFlag() | - Toggle m\_cell\_state between Closed and Flagged  - Does nothing if m\_cell\_state equals Opened. |

## struct Table

(defined in table.h)

Represents the table of the game that the player has to solve.

| Variable | Default value | Description |
| --- | --- | --- |
| static constexpr int  corner\_x | 83 | The x-coordinate of the top-left pixel of the rendered table |
| static constexpr int  corner\_y | 80 | The y-coordinate of the top-left pixel of the rendered table |
| static constexpr int  center\_x | 683 | The x-coordinate of the center of the rendered table |
| static constexpr int  center\_y | 380 | The y-coordinate of the center of the rendered table |
| int  m\_width | 0 | The width of the table  (in number of Cells) |
| int  m\_height | 0 | The height of the table  (in number of Cells) |
| std::vector<Cell>  m\_board |  | The cells of the table |
| std::vector<  std::pair<int, int>>  m\_bomb\_cell\_coords |  | Pairs of coordinates of bomb cells |
| int  m\_cells\_revealed | 0 | The number of cells revealed |

| Function | Description |
| --- | --- |
| Table() | Constructs a new Table |
| Table(int width, int height) | Constructs a new Table,  assigns width to m\_width,  assigns height to m\_height and  initialize m\_board with the size equal to m\_width \* m\_size |
| Cell&  getCell(int x, int y) | Returns a reference to the cell with the table coordinates (x, y) |
| int  getWidth() const | Returns m\_width |
| int  getHeight() const | Returns m\_height |
| void  drawTable() | Draws the table on the screen |
| void  fillTable() | Fills the table randomly |
| void  initCellsScreenPos() | Initializes the screen positions of each cell |
| void  clearNearbyCells(  int coord\_x,  int coord\_y) | Reveal nearby cells until revealing numbered cells (cells with value greater than 0) |
| std::pair<int, int>  getCoordsFromPos(  double pos\_x,  double pos\_y) | Returns the pair of table coordinates of the cell with the mouse cursor at position (pos\_x, pos\_y) |
| std::pair<double, double>  getPosFromCoords(  int coord\_x,  int coord\_y) | Returns the pair of screen positions of the topleft pixel of the cell with the table coordinates (coord\_x, coord\_y) |
| int  revealCell(  int coord\_x,  int coord\_y) | - Reveals the cell at table coordinates (coord\_x, coord\_y)  - Returns -1 if clicked on a bomb, 1 if all non-bomb cells are revealed, otherwise 0 |
| bool  coordsInRange(  int coord\_x,  int coord\_y) | Returns true if the coordinates (coord\_x, coord\_y) are within the table’s dimensions, otherwise false |
| void  loadFromSaveData(  const std::string& table,  const std::string& state) | Loads saved data from the data file (table representing the values of the cells, state representing the cell states) |
| int  getNumberOfRevealedCells() const | Returns m\_cells\_revealed |

## struct Config

(defined in config.h)

A singleton that holds the game’s configurations, using [Scott Meyers’ singleton design](https://laristra.github.io/flecsi/src/developer-guide/patterns/meyers_singleton.html) (which utilizes local static variables).

Note: Since the game is single-threaded, problems involving *“pontential thread safety issues with data members of this singleton type”* does not appear to be an issue.

| **struct** **Config** { **public**:  **static** Config& **getConfigInstance**();  **private**:  Config();  ~Config();   Config(**const** Config&) = **delete**;  Config(Config&&) = **delete**;  Config& **operator**=(**const** Config&) = **delete**;  Config& **operator**=(Config&&) = **delete**; }; |
| --- |

| Variable | Default value | Description |
| --- | --- | --- |
| int  table\_width | 20 | The width of the table  (in number of Cells) |
| int  table\_height | 10 | The height of the table  (in number of Cells) |
| int  number\_of\_bombs | 20 | The number of bombs |
| YAML::Node  config |  | The YAML Node representing the config file |

| Function | Description |
| --- | --- |
| Config() | - Constructs a new Config  - Creates the configuration file if not existed  Invokes readConfig() |
| ~Config() | - Invokes writeConfig()  - Destructs the Config object |
| static Config&  getConfigInstance() | Returns a reference to the Config instance |
| void  readConfig() | - Reads from the configuration file  - Invokes writeConfig() (in case the read values are invalid and require re-initialization) |
| void  writeConfig() | Writes to the configuration file |

## enum class GameState

(defined in gameplay\_screen.h)

| **enum** **class** **GameState** {  Playing,  Won,  Lost, }; |
| --- |

The value of enum class GameState tells the current state of a game (playing, won, lost).

## namespace gameplayScreen

(defined in gameplay\_screen.h)

The namespace containing functions for the gameplay.

| Function | Description |
| --- | --- |
| void  interact() | Allows interactions and handle events |
| void  draw() | Draws the screen |
| void  startNewGame() | - Initializes a new Table with dimensions from the Config instance  - Sets internal::game\_state to Playing |
| void  loadOldGame() | - Loads data from save file  - Modify the configurations to the old game  - Restore the game state from saved data |
| void  saveOldGame() | Saves current game state to save file |
| void  loadHighScore() | Loads high score |
| void  saveHighScore() | Saves current high score |

## namespace gameplayScreen::internal

(defined in gameplay\_screen.h)

Internal functions and variables for the gameplay. Not meant to be accessed outside namespace gameplayScreen.

| Variable | Default value | Description |
| --- | --- | --- |
| extern Table  table |  | The table of the game |
| extern GameState  game\_state | Playing | The state of the game |
| extern int  time\_elapsed | 0 | The duration elapsed of the game (in seconds) |
| extern int  frame\_counter | 0 | The number of frames since the last elapsed second |
| extern double  score | 0 | The current score of the game |
| extern double  high\_score | 0 | The highest score recorded of all games |

| Function | Description |
| --- | --- |
| void  updateFrameCount() | - Increases frame\_counter by 1  - If frame\_counter equals global::frames\_per\_second, resets frame\_counter to 0 and increases time\_elapsed by 1 |
| std::array<int, 3>  getCurrentTime() | Get an array representing the current time  The values of the array are {hours, minutes, second} |

## namespace menuScreen

(defined in menu\_screen.h)

The namespace containing functions and variables for the menu.

| Variable | Default value | Description |
| --- | --- | --- |
| constexpr int  play\_corner\_x | 95 | The x-coordinate of the top-left pixel of the New Game button |
| constexpr int  play\_corner\_y | 390 | The y-coordinate of the top-left pixel of the New Game button |
| constexpr int  continue\_corner\_x | 95 | The x-coordinate of the top-left pixel of the Continue button |
| constexpr int  continue\_corner\_y | 495 | The y-coordinate of the top-left pixel of the Continue button |
| constexpr int  settings\_corner\_x | 95 | The x-coordinate of the top-left pixel of the Settings button |
| constexpr int  settings\_corner\_y | 600 | The y-coordinate of the top-left pixel of the Settings button |
| constexpr int  button\_width | 230 | The width of the menu buttons |
| constexpr int  button\_height | 100 | The height of the menu buttons |

| Function | Description |
| --- | --- |
| void  interact() | Allows interactions and handle events |
| void  draw() | Draws the screen |

## namespace settingsScreen

(defined in settings\_screen.h)

The namespace containing functions for the settings screen.

| Function | Description |
| --- | --- |
| void  interact() | Allows interactions and handle events |
| void  draw() | Draws the screen |

## enum class ScreenType

(defined in screen.h)

| **enum** **class** **ScreenType** {  Menu,  Gameplay,  Settings, }; |
| --- |

The value of enum class ScreenType tells the current state of a game (menu, gameplay, settings).

## namespace global

(defined in screen.h)

The global namespace containing functions allowing screen switch.

| Function | Description |
| --- | --- |
| void  screenToMenu() | Switches to menu screen |
| void  screenToGameplay() | - Loads high score and starts a new game  - Switches to gameplay screen |
| void  screenToContinue() | - Loads high score and loads the saved game  - Switches to gameplay screen |
| void  screenToSettings() | Switches to settings screen |
| ScreenType  getScreenType() | Returns the current screen type |

## namespace global

(defined in globals.h)

The global namespace containing variables for the program.

| Variable | Default value | Description |
| --- | --- | --- |
| constexpr int  screen\_width | 1366 | The width of the screen |
| constexpr int  screen\_height | 768 | The height of the screen |
| constexpr int  frames\_per\_second | 60 | The number of frames per second |

## Miscellaneous

(defined in utils.h)

| Function | Description |
| --- | --- |
| void DrawTextSus(  const char\* text,  int pos\_x,  int pos\_y,  int font\_size,  Color color) | Draws text with the “sus” typeface  (“In your face, Joffrey”, used by the game “Among Us”) |