CPS2004 — Object Oriented Programming

Assignment

Juan Scerri
B.Sc. (Hons)(Melit.) Computing Science and Mathematics (Second Year)

December 24, 2022

Contents

List of Figures

Listings

1 Plagiarism Declaration

Plagiarism is defined as *"the unacknowledged use, as one's own, of work of another person, whether or not such work has been published, and as may be further elaborated in Faculty or University guidelines"* (University Assessment Regulations, 2009, Regulation 39 (b)(i), University of Malta).

I, the undersigned, declare that the report submitted is my work, except where acknowledged and referenced. I understand that the penalties for committing a breach of the regulations include loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Work submitted without this signed declaration will not be corrected, and will be given zero marks.

Juan Scerri	CPS2004	December 24, 2022
Student's full name	**Study-unit code**	**Date of submission**

Title of submitted work: Object Oriented Programming Assignment

Student's signature

2 Village War Game

2.1 Language Choice

Java was chosen for the Village War Game because it has more complicated entity lifetimes. Programming the game in C++ would have required dealing with pointers to manage lifetimes. Obviously, dealing with pointers brings the possibility of memory leaks. Since Java has a Garbage Collector (GC) there is no need for manual deallocation of memory.

2.2 User Guide

2.2.1 Download, Compiling & Running

1. Clone the repository.

```
$ git clone https://github.com/JuanScerriE/village-war-game
```

2. Compile the game.

```
$ cd village-war-game ; ./compile.sh
```

3. Run the game.

```
village-war-game $ ./run.sh
```

2.2.2 Playing

Starting the game the player is asked to input the number of human players.

Then the player is aksed to input the number of AI players.

The game loop will start. Each human player will be prompted with a menu of options. There are two categories: Info and Actions. The Info category contains options which provide the player with information about his own village, enemy villages, armies and costs. The Actions category contains options which affect the state of the game such as building, training and attacking. Finally, the player can decided to pass the turn to the next player.

Each player by default at the start will have 50 food, metal and mana. The player can then use these resource to build or upgrade buildings and train troops.

The following types of building can be built:

1. Academy (to generate wizards)
2. Foundation (to generate scouts)
3. Arena (to generate brawlers)
4. Farm (to generate food)
5. Mine (to generate metal)

6. Mana Tower (to generate mana)

As described in the above list there are three types of troops:

1. Wizard (high attack, low health, medium speed, low carrying capacity)
2. Brawler (high attack, high health, slow speed, medium carrying capacity)
3. Scout (low attack, medium health, high speed, high carrying capacity)

2.3 Design

2.4 Technical Aspects

2.5 Testing

2.6 Limitations & Improvements

3 Minesweeper

3.1 Language Choice

C++ was chosen for Minesweeper because it has a fixed board size of 16×16. This means that it is possible to stack allocate every object removing the need for dynamic memory allocation. This is facilitated by `std::array` from the Standard Template Library (STL) which allows for the creation of fixed size arrays on the stack.

3.2 User Guide

3.2.1 Download, Compiling & Running

1. Clone the repository.

```
$ git clone https://github.com/JuanScerriE/minesweeper
```

2. Compile the tests and the game.

   Note: Make sure that `gtest` and `ncurses` are installed for the tests and the game, respectively.

```
$ cd minesweeper ; ./compile.sh
```

3. Run the tests.

   Note: Some tests might fail. This is because the implentation of `srand` and `rand` differ between platforms (specifically macOS and Linux).

```
minesweeper $ ./tests.sh
```

4. Run the game.

```
minesweeper $ ./run.sh
```

3.2.2 Playing

```
** 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
00 00 00 00 00 00 00 00 00 00 01 -- -- -- -- -- --
01 00 00 00 00 00 00 01 01 01 01 -- -- -- -- -- --
02 00 00 00 00 00 00 02 -- -- -- -- -- -- -- -- --
03 00 00 00 00 00 00 02 -- -- -- -- -- -- -- -- --
04 00 00 00 00 00 00 01 02 02 -- -- -- -- -- -- --
05 00 00 00 00 00 00 00 00 01 -- -- -- -- -- -- --
06 00 00 00 00 00 01 01 01 01 -- -- -- -- -- -- --
07 00 00 00 01 01 03 -- -- -- -- -- -- -- -- -- --
08 00 01 01 03 -- -- -- -- -- -- -- -- -- -- -- --
09 01 02 -- -- -- -- -- -- -- -- -- -- -- -- -- --
10 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
11 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
12 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
13 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
14 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
15 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --



 Pos:     (00, 00)
 q:       Quit
 h,j,k,l: Left, down, up, right
 SPACE:   Reveal hidden cell
 r:       Reset board
```

Figure 1: Playing Minesweeper

The general guide to playing Minesweeper is described above in figure 1.

Note: `vim` motions are used to move the cursor.

```
** 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
00 00 00 00 00 00 00 00 00 00 01 -- -- -- -- -- --
01 00 00 00 00 00 00 01 01 01 01 -- -- -- -- -- --
02 00 00 00 00 00 00 02 XX -- -- -- -- -- -- -- --
03 00 00 00 00 00 00 02 -- -- -- -- -- -- -- -- --
04 00 00 00 00 00 00 01 02 02 -- -- -- -- -- -- --
05 00 00 00 00 00 00 00 00 01 -- -- -- -- -- -- --
06 00 00 00 00 00 01 01 01 01 -- -- -- -- -- -- --
07 00 00 00 01 01 03 -- -- -- -- -- -- -- -- -- --
08 00 01 01 03 -- -- -- -- -- -- -- -- -- -- -- --
09 01 02 -- -- -- -- -- -- -- -- -- -- -- -- -- --
10 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
11 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
12 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
13 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
14 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
15 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

 YOU HAVE HIT A MINE! (Press r to retry)

 Pos:     (07, 02)
 q:       Quit
 h,j,k,l: Left, down, up, right
 SPACE:   Reveal hidden cell
 r:       Reset board
```

Figure 2: Hitting a mine

If the player hits a mine the above message as in figure 2 will be displayed.

```
** 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
00 00 00 00 00 01 02 -- 01 00 00 00 01 01 01 01 01
01 00 00 00 00 01 -- 02 01 01 01 02 02 -- 01 01 --
02 01 01 00 00 01 01 02 01 03 -- 03 -- 02 01 01 01
03 -- 02 00 00 00 00 01 -- 03 -- 04 02 01 00 01 01
04 -- 03 00 00 00 01 02 02 02 02 -- 01 00 00 01 --
05 -- 02 00 00 00 01 -- 01 00 01 01 01 00 00 01 01
06 01 01 01 02 02 02 01 01 00 00 00 00 00 00 00 00
07 01 01 01 -- -- 01 01 01 01 00 01 01 01 00 00 00
08 -- 01 01 02 02 01 01 -- 01 01 02 -- 02 02 02 01
09 01 01 01 01 01 00 01 01 02 02 -- 02 02 -- -- 02
10 00 01 02 -- 02 02 02 02 02 -- 02 01 01 03 -- 02
11 00 01 -- 02 02 -- -- 02 -- 03 02 00 01 02 02 01
12 00 01 01 01 01 03 03 04 04 -- 02 00 01 -- 02 01
13 00 00 00 00 00 01 -- 02 -- -- 03 01 01 01 02 --
14 01 01 01 00 01 02 03 03 03 03 -- 02 01 01 01 01
15 01 -- 01 00 01 -- 02 -- 01 01 01 02 -- 01 00 00

 YOU HAVE CLEARED THE BOARD! (Press r to retry)

 Pos:     (03, 01)
 q:       Quit
 h,j,k,l: Left, down, up, right
 SPACE:   Reveal hidden cell
 r:       Reset board
```

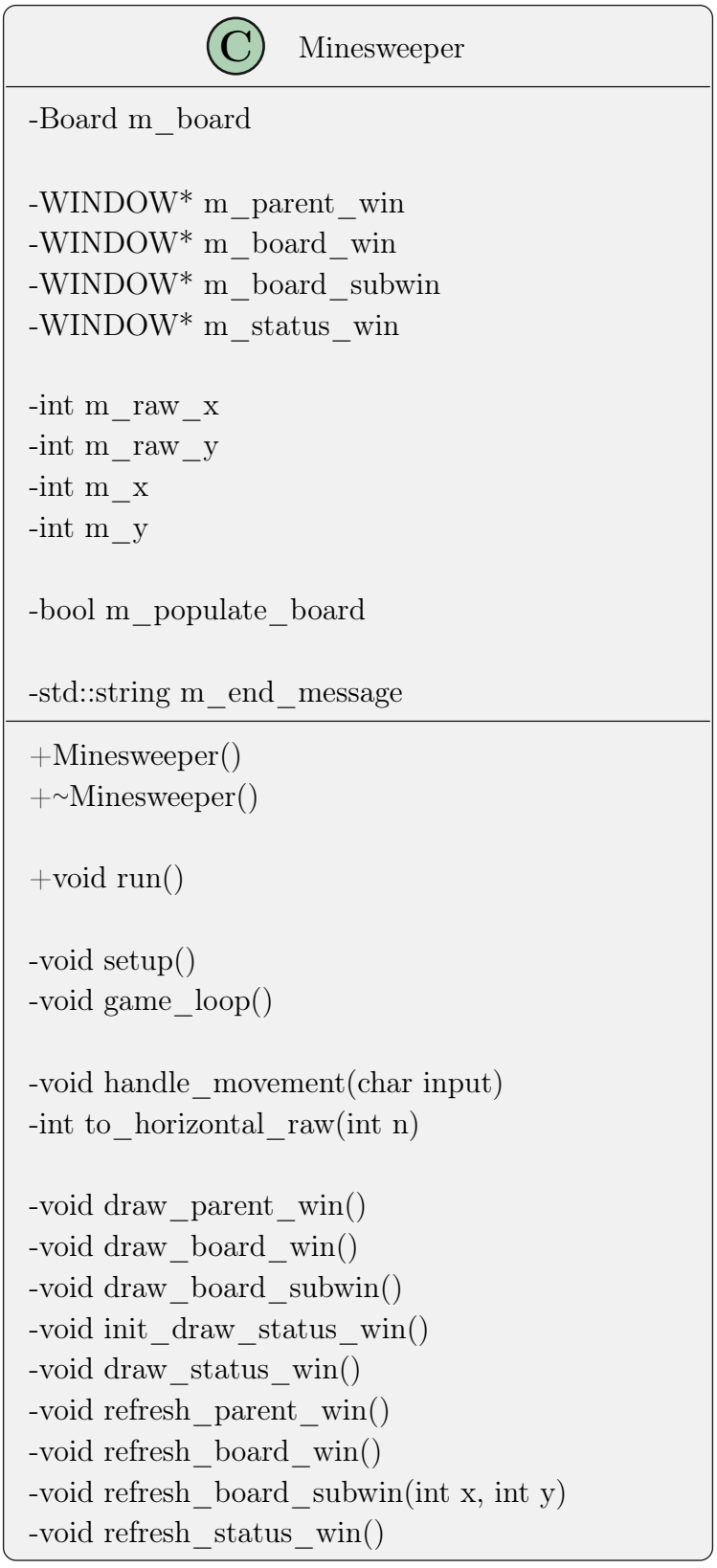
Figure 3: Finishing a game of Minesweeper

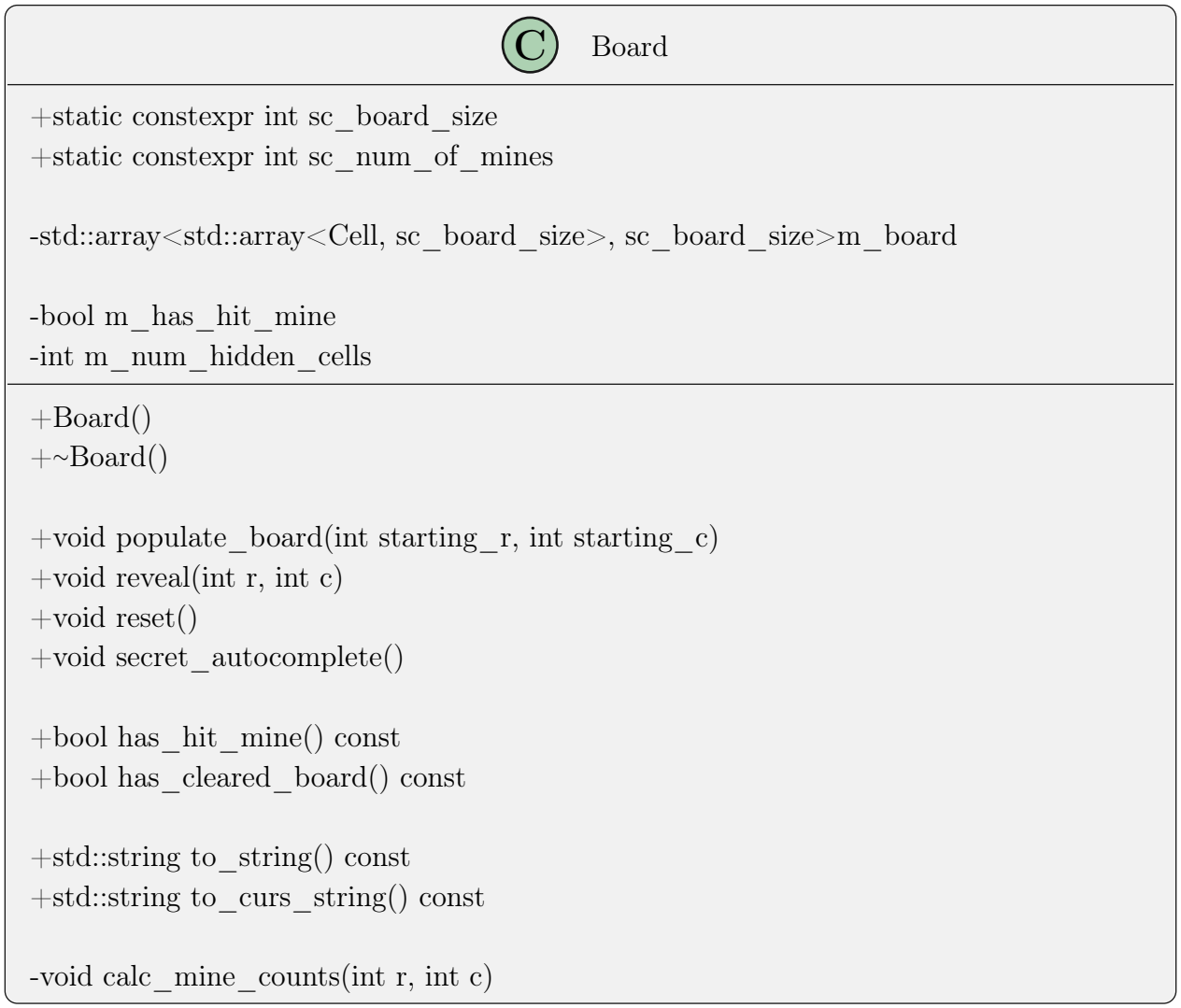
If the player manages to clear all the cells without hitting a mine the above message as in figure 3 will be displayed.

Finally, there is an additional *secret* command to automatically complete the board without hitting a mine. The user needs to press `W` (`shift + w`).

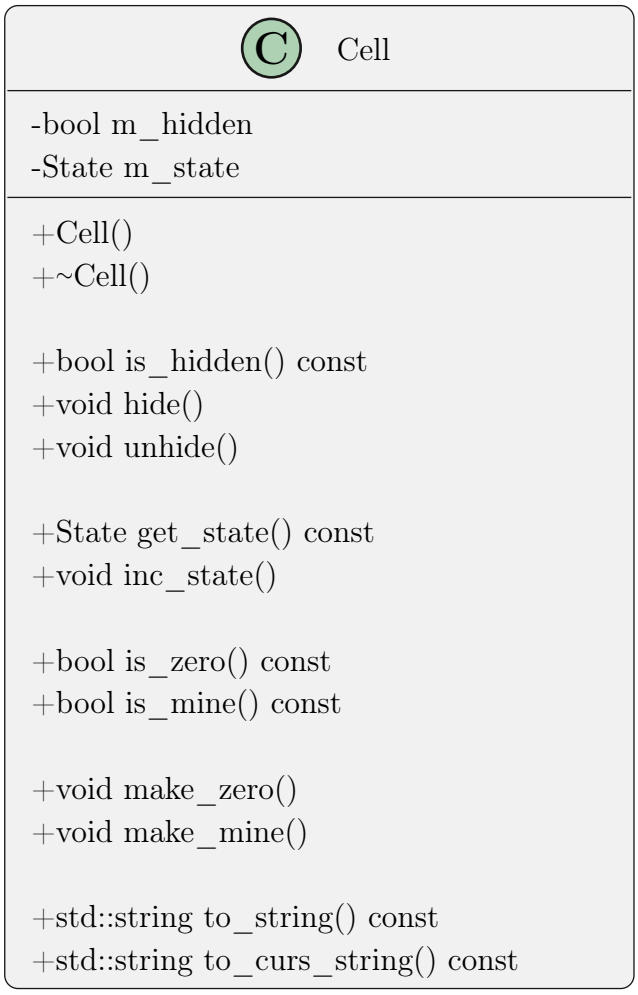
Note: This only works if the user has at least revealed one cell. This is because the board is populated with all the mines after the first reveal to ensure a player never hits a mine on his first try.

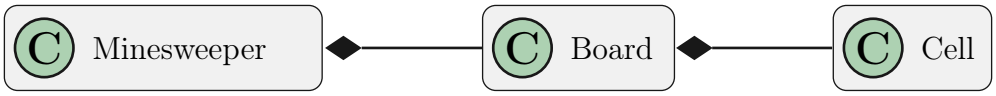
3.3 Design


Minesweeper
-Board m_board
-WINDOW* m_parent_win
-WINDOW* m_board_win
-WINDOW* m_board_subwin
-WINDOW* m_status_win
-int m_raw_x
-int m_raw_y
-int m_x
-int m_y
-bool m_populate_board
-std::string m_end_message
+Minesweeper()
+~Minesweeper()
+void run()
-void setup()
-void game_loop()
-void handle_movement(char input)
-int to_horizontal_raw(int n)
-void draw_parent_win()
-void draw_board_win()
-void draw_board_subwin()
-void init_draw_status_win()
-void draw_status_win()
-void refresh_parent_win()
-void refresh_board_win()
-void refresh_board_subwin(int x, int y)
-void refresh_status_win()


Board
+static constexpr int sc_board_size
+static constexpr int sc_num_of_mines
-std::array<std::array<Cell, sc_board_size>, sc_board_size>m_board
-bool m_has_hit_mine
-int m_num_hidden_cells
+Board()
+~Board()
+void populate_board(int starting_r, int starting_c)
+void reveal(int r, int c)
+void reset()
+void secret_autocomplete()
+bool has_hit_mine() const
+bool has_cleared_board() const
+std::string to_string() const
+std::string to_curs_string() const
-void calc_mine_counts(int r, int c)

State	
MINE	-1
ZERO	0
ONE	1
TWO	2
THREE	3
FOUR	4
FIVE	5
SIX	6
SEVEN	7
EIGHT	8





The Minesweeper class contains the user interface code, that is it contains all `ncurses` specific code. The class also handles user input.

The Board class contains the majority of the game logic. It is a part of the Minesweeper class, that is if a Minesweeper object ceases to exist so does the Board object. Further more the actual board `m_board` is a grid of Cell objects. Also the lifetime of the objects is managed by the Board since when the board ceases to exist so do the cells.

3.4 Testing

```
[       OK ] CellTest.ZeroCellToString (0 ms)
[----------] 11 tests from CellTest (0 ms total)

[----------] 7 tests from BoardTest
[ RUN      ] BoardTest.PopulateBoard
[       OK ] BoardTest.PopulateBoard (0 ms)
[ RUN      ] BoardTest.HasHitMine
[       OK ] BoardTest.HasHitMine (0 ms)
[ RUN      ] BoardTest.HasNotHitMine
[       OK ] BoardTest.HasNotHitMine (0 ms)
[ RUN      ] BoardTest.HasNotClearedBoard
[       OK ] BoardTest.HasNotClearedBoard (0 ms)
[ RUN      ] BoardTest.RevealZeroCellAndNeighbouringCellsRecursively
[       OK ] BoardTest.RevealZeroCellAndNeighbouringCellsRecursively (0 ms)
[ RUN      ] BoardTest.HasClearedBoardWithoutHittingMineManually
[       OK ] BoardTest.HasClearedBoardWithoutHittingMineManually (0 ms)
[ RUN      ] BoardTest.HasClearedBoardWithoutHittingMineWithSecretAutocomplete
[       OK ] BoardTest.HasClearedBoardWithoutHittingMineWithSecretAutocomplete (0 ms)
[----------] 7 tests from BoardTest (0 ms total)

[----------] Global test environment tear-down
[==========] 18 tests from 2 test suites ran. (0 ms total)
[  PASSED  ] 18 tests.
```

Figure 4: Unit tests for Minesweeper (on macOS Ventura 13.1)

Black-box Testing and Unit Testing were used to test the application. For unit testing `gtest` is required.

```
% sudo leaks -atExit -- ./minesweeper
Password:
minesweeper(23642) MallocStackLogging: could not tag MSL-related memory as no_footprint, so those pages will be included in process footprint - (null)
minesweeper(23642) MallocStackLogging: recording malloc and VM allocation stacks using lite mode
Process 23642 is not debuggable. Due to security restrictions, leaks can only show or save contents of readonly memory of restricted processes.

Process:         minesweeper [23642]
Path:            /Users/USER/Desktop/*/minesweeper
Load Address:    0x1005b4000
Identifier:      minesweeper
Version:         0
Code Type:       ARM64
Platform:        macOS
Parent Process:  leaks [23641]

Date/Time:       2022-12-24 16:37:32.414 +0100
Launch Time:     2022-12-24 16:37:16.228 +0100
OS Version:      macOS 13.1 (22C65)
Report Version:  7
Analysis Tool:   /Applications/Xcode.app/Contents/Developer/usr/bin/leaks
Analysis Tool Version:  Xcode 14.2 (14C18)

Physical footprint:         3985K
Physical footprint (peak):  3985K
Idle exit: untracked
----

leaks Report Version: 4.0, multi-line stacks
Process 23642: 599 nodes malloced for 553 KB
Process 23642: 0 leaks for 0 total leaked bytes.
```

Figure 5: Testing for memory leaks (on macOS Ventura 13.1)

Furthermore, to test for memory leaks, on macOS, `leaks` was used and, on Linux, `valgrind` was used. `leaks` reported not memory leaks whilst `valgrind` reported leaks from `ncureses`.

3.5 Limitations & Improvements

Limitation: The unit tests are not cross-platform; four of the unit tests fail on Linux. This is mostly due to differing implementations of `srand()` and `rand()` on different platforms.

Solution: Create a custom random number generated. This guarantees the same result across different platforms.

Limitation: The `ncurses` library leaks memory on Linux whilst on macOS it does not.

Solution: This seems to be intended behaviour from `ncurses`. Read the man page at `https://man7.org/linux/man-pages/man3/curs_memleaks.3x.html`