

# CPS2004 — Object Oriented Programming

## Assignment

**Juan Scerri**

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

December 26, 2022

## Contents

<b>1</b>	<b>Plagiarism Declaration</b>	<b>3</b>
<b>2</b>	<b>Village War Game</b>	<b>3</b>
2.1	Language Choice	3
2.2	User Guide	4
2.2.1	Download, Compiling & Running	4
2.2.2	Playing	4
2.3	Design	10
2.4	Technical Aspects	10
2.5	Testing	10
2.6	Limitations & Improvements	10
<b>3</b>	<b>Minesweeper</b>	<b>10</b>
3.1	Language Choice	10
3.2	User Guide	11
3.2.1	Download, Compiling & Running	11
3.2.2	Playing	12
3.3	Design	15
3.4	Testing	18
3.5	Limitations & Improvements	18

## List of Figures

1	Picking the number of human players . . . . .	4
2	Picking the number of AI players . . . . .	4
3	Menu for the human players . . . . .	5
4	Building an Academy in the player's village . . . . .	6
5	Info about buildings in the player's village . . . . .	6
6	Upgrading an Academy in the player's village . . . . .	7
7	Training Wizards in the player's village . . . . .	8
8	Attacking another village with Wizards in the player's village . . . . .	9
9	Viewing an army marching towards an enemy village . . . . .	10
10	Playing Minesweeper . . . . .	12
11	Hitting a mine . . . . .	13
12	Finishing a game of Minesweeper . . . . .	14
13	Unit tests for Minesweeper (on macOS Ventura 13.1) . . . . .	18
14	Testing for memory leaks (on macOS Ventura 13.1) . . . . .	18

## 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 26, 2022

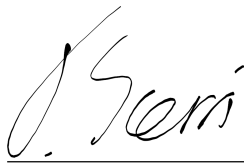
**Student’s full name**

**Study-unit code**

**Date of submission**

**Title of submitted work:** Object Oriented Programming Assignment

**Student’s signature**

A handwritten signature in black ink, appearing to read 'J. Scerri', is written over a horizontal line.

## 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

```
Enter the number of human players  
> 1
```

Figure 1: Picking the number of human players

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

```
Enter the number of AI players  
> 2
```

Figure 2: Picking the number of AI players

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

```
Human 1
Info:
1. Print village stats      2. Print stationed troops stats
3. Print building stats    4. Print costs
5. Print enemy villages    6. Print armies
Actions:
7. Build                   8. Upgrade
9. Train                   10. Attack
11. Pass
> █
```

Figure 3: Menu for the human 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. Every option in the **Info** category is self-explanatory.

The **Actions** category contains options which affect the state of the game such as *building*, *training* and *attacking*. Finally, the player can decide to pass the turn to the next player.

All players by default will start with 50 “food”, “metal” and “mana”. The player can use these resources 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 (medium attack, medium health, high speed, high carrying capacity)

## Building

```

Human 1
Info:
1. Print village stats          2. Print stationed troops stats
3. Print building stats        4. Print costs
5. Print enemy villages        6. Print armies
Actions:
7. Build                        8. Upgrade
9. Train                       10. Attack
11. Pass
> 7

Cost to build Farm: [Food: 10, Metal: 20, Mana: 5]
Cost to build Mine: [Food: 15, Metal: 20, Mana: 5]
Cost to build Mana Tower: [Food: 10, Metal: 25, Mana: 5]
Cost to build Academy: [Food: 10, Metal: 15, Mana: 10]
Cost to build Arena: [Food: 10, Metal: 15, Mana: 10]
Cost to build Foundation: [Food: 10, Metal: 15, Mana: 10]
1. Build Academy
2. Build Foundation
3. Build Arena
4. Build Farm
5. Build Mine
6. Build Mana Tower
7. Go Back
> 1

```

Figure 4: Building an Academy in the player's village

As depicted above in figure 4, the player is prompted with the different costs of the different buildings. If the player does not have enough resources for building he/she is notified.

```

Info:
1. Print village stats          2. Print stationed troops stats
3. Print building stats        4. Print costs
5. Print enemy villages        6. Print armies
Actions:
7. Build                        8. Upgrade
9. Train                       10. Attack
11. Pass
> 3
Resource Buildings:
No buildings!
Troop Buildings:
- Academy, Level 1

```

Figure 5: Info about buildings in the player's village

As shown in figure 7, the buildings can have different levels of productivity. The initial level of productivity for all buildings is 1.

## Upgrading

```

Info:
1. Print village stats          2. Print stationed troops stats
3. Print building stats        4. Print costs
5. Print enemy villages        6. Print armies
Actions:
7. Build                        8. Upgrade
9. Train                       10. Attack
11. Pass
> 8

Cost to upgrade Farm: [Food: 5, Metal: 25, Mana: 5]
Cost to upgrade Mine: [Food: 10, Metal: 20, Mana: 10]
Cost to upgrade Mana Tower: [Food: 5, Metal: 20, Mana: 15]
Cost to upgrade Academy: [Food: 5, Metal: 10, Mana: 20]
Cost to upgrade Arena: [Food: 5, Metal: 10, Mana: 20]
Cost to upgrade Foundation: [Food: 5, Metal: 10, Mana: 20]
1. Upgrade Academy
2. Upgrade Foundation
3. Upgrade Arena
4. Upgrade Farm
5. Upgrade Mine
6. Upgrade Mana Tower
7. Go Back
> 1

```

Figure 6: Upgrading an Academy in the player's village

Upgrading the building increases the building's level of productivity. This has the effect of generating more resources or troops per round. Similarly, if the player does not have enough resources for upgrading he/she is notified.

Additionally, the building which is upgraded first is the oldest building.

## Training

```
Info:
1. Print village stats          2. Print stationed troops stats
3. Print building stats       4. Print costs
5. Print enemy villages       6. Print armies
Actions:
7. Build                      8. Upgrade
9. Train                     10. Attack
11. Pass
> 9

Cost to train Wizard: [Food: 2, Metal: 0, Mana: 3]
Cost to train Brawler: [Food: 2, Metal: 3, Mana: 1]
Cost to train Scout: [Food: 2, Metal: 2, Mana: 1]
1. Train Wizards
2. Train Brawlers
3. Train Scouts
4. Go Back
> 1
Input number of troops
> 2
```

Figure 7: Training Wizards in the player's village

Training troops increases the overall stats of the chosen troops. Similarly if the player does not have enough resources or troops he/she is notified.

Additionally, the training scheme is the same as for buildings i.e. the oldest troops are trained first, and each troop has an initial level of 1 and a maximum level of 3.



## Attacking

```

Info:
1. Print village stats          2. Print stationed troops stats
3. Print building stats        4. Print costs
5. Print enemy villages        6. Print armies
Actions:
7. Build                        8. Upgrade
9. Train                       10. Attack
11. Pass
> 10
Enemy Villages:
1. AI 1
Location: (5, 9)
Health: 1000
Resources: [Food: 30, Metal: 10, Mana: 51]
2. AI 2
Location: (27, 4)
Health: 1000
Resources: [Food: 15, Metal: 8, Mana: 34]
Input village number
> 1
<Station>
Number of Wizards: 9
Number of Brawlers: 0
Number of Scouts: 0
Total Attack Power: 184
Total Carrying Capacity: 47
Slowest Movement Speed: 2
Input number of wizards
> 9
Input number of brawlers
> 0
Input number of scouts
> 0

```

Figure 8: Attacking another village with Wizards in the player's village

To attack another village the player needs to pick a enemy village to attack. Then the player must specify the the composition of the army.

Again if the player does not have enough troops he/she is notified.

The troops will be moved from the village station into the army. This means that the village will have less defensive power making it more vulnerable to attack.

Additionally, when the army arrives at it destination the player will be notified of whether the attack was successful or unsuccessful.

## Incorrect Player Input

In all instances where the player input is unexcepted or incorrect, the player is notified. All error messages are handled in the `HumanPlayer` class.

Moreover, error handling was performed by returning a `Status` enum which contains a number of

possible errors which the program might encounter.

**Note:** Exceptions were avoided at all costs as they obscure control flow. Moreover, most errors do not arise because of exceptional circumstances. Exceptions were handled only when thrown from the standard library.

```

Info:
1. Print village stats          2. Print stationed troops stats
3. Print building stats        4. Print costs
5. Print enemy villages        6. Print armies
Actions:
7. Build                       8. Upgrade
9. Train                      10. Attack
11. Pass
> 6
<Army>
Sent By: Human 1
Attacking: AI 1
Location: (27, 35)
Number of Wizards: 9
Number of Brawlers: 0
Number of Scouts: 0
Total Attack Power: 184
Total Carrying Capacity: 47
Slowest Movement Speed: 2

```

Figure 9: Viewing an army marching towards an enemy village

## 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  $16 \times 16$ . 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 implementation 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	00	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 10: Playing Minesweeper

The general guide to playing Minesweeper is described above in figure 10.

**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 11: Hitting a mine

If the player hits a mine the above message as in figure 11 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
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

```

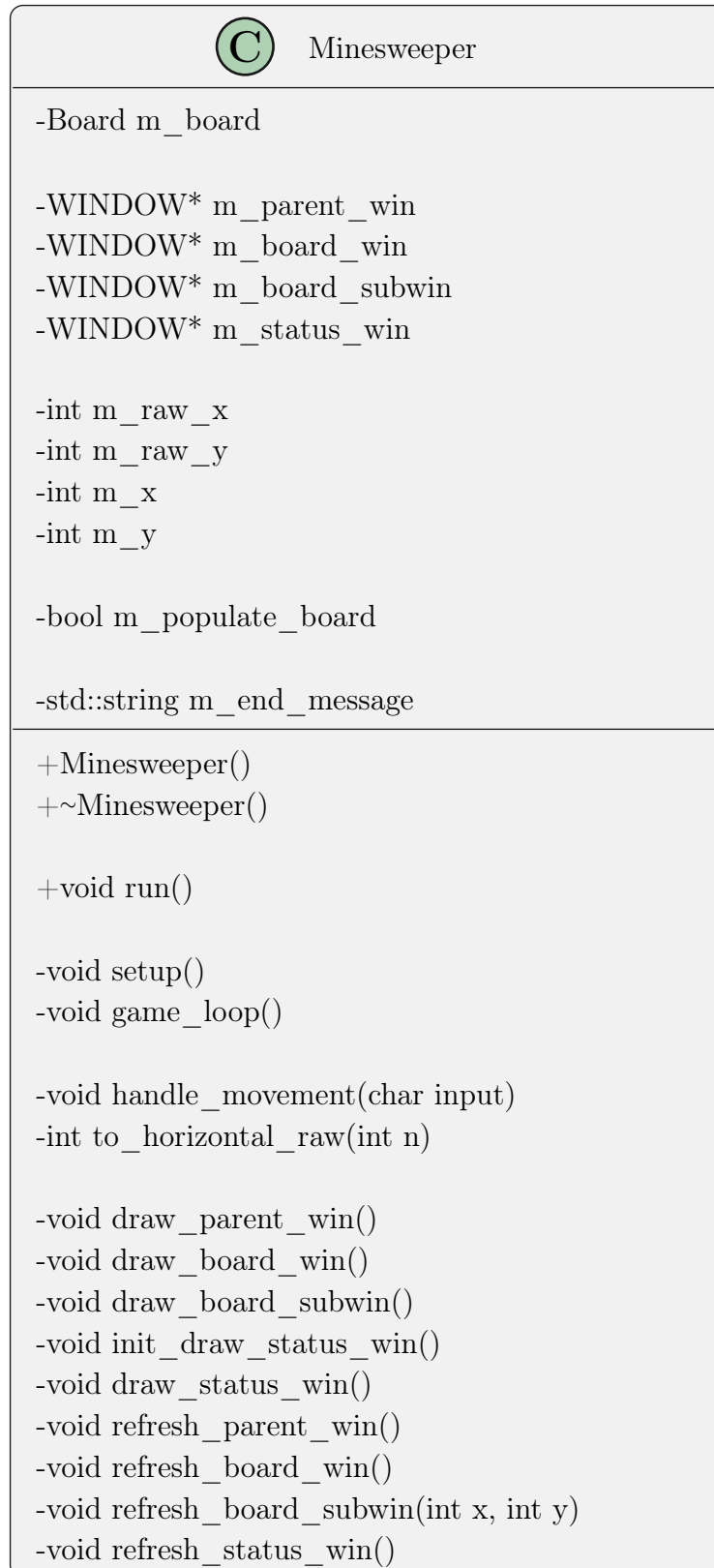
Figure 12: Finishing a game of Minesweeper

If the player manages to clear all the cells without hitting a mine the above message as in figure 12 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





## 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()

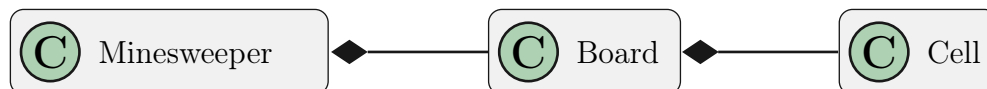
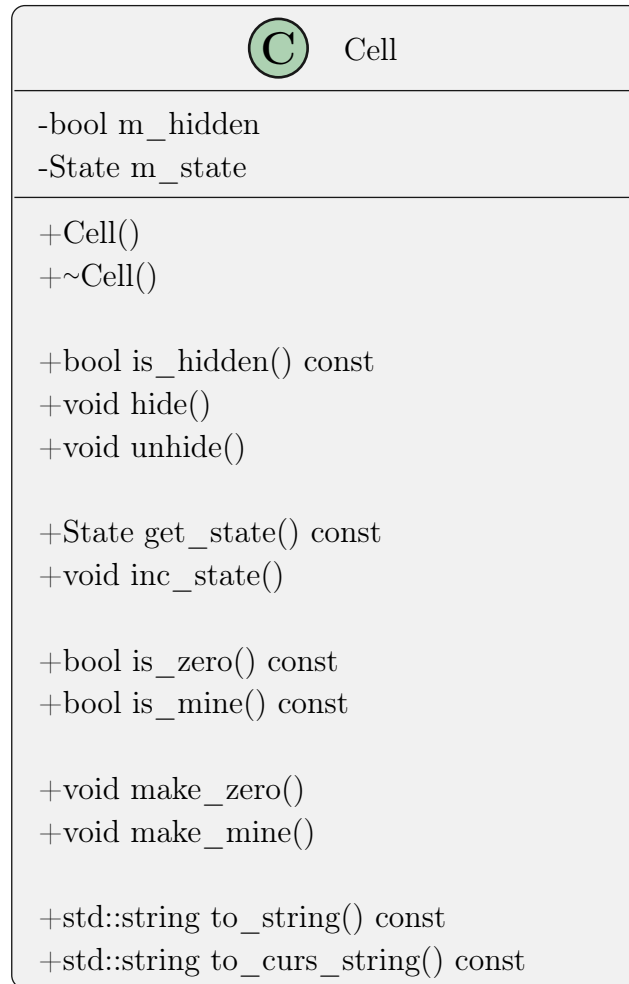
+bool has_hit_mine() const
+bool has_cleared_board() const

+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 13: 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 14: 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 `ncurses`.

### 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](https://man7.org/linux/man-pages/man3/curs_memleaks.3x.html)