UNIVERSITY OF JOHANNESBURG

FACULTY OF SCIENCE

ACADEMY OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING

MODULE	CSC01A1 Introduction to algorithm development (C++)
CAMPUS	**APK**

EXAM
PAPER B

DATE: 2022-06-xx

ASSESSOR(S) **DR JL DU TOIT**

INTERNAL MODERATOR **MR S SITHUNGU**

DURATION 3 HOURS **MARKS 100**

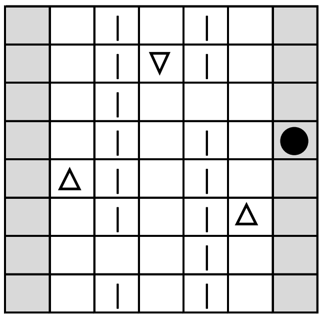
1. **You must complete this test by yourself within the prescribed time limits.**
2. **You are bound by all university regulations, with a special note regarding assessment, plagiarism, and ethical conduct.**
3. **You must regularly save your work to the VPL system https://acsse.cloud during the assessment. We recommend working directly on VPL in case the computer you are using crashes so that your work is still accessible. You may access your VPL login details under marks**
4. **Should you wish to use CodeBlocks, you MUST save your project to the T-drive. Should the computer hang and work is not saved to the T-drive, then no extra time will be granted, and you will have to start over.**
5. **At the end of the assessment, your completed code must be submitted in a zip archive to EVE. This zip file is the one downloaded from the VPL system.**
6. **No communication concerning this test is permissible during the assessment session except with Academy staff members.**

NUMBER OF PAGES: 3 PAGES

Mark sheet		
Competency	*Description*	*Result*
C0	Program Design	/10
C1	Boiler plate code • Standard namespace (1) • System library inclusion (3) • Indication of successful termination of program (1)	/5
C2	Coding style • Naming of variables (1) • Indentation (1) • Use of comments (1) • Use of named constants (1) • Program compiles without issuing warnings (1)	/5
C3	Functional Abstraction • Task decomposition (5) • Reduction of repetitive code (5)	/10
C4	Separate Compilation • Header file (1) • Guard conditions (2) • Inclusion of header file (1) • Appropriate content in header file (1) • Use of programmer defined namespace (5)	/10
C5	User Interaction • Menu System (5) • Appropriate use of input, output and error streams (5)	/10
C6	Command Line Argument Handling: • Appropriately overloaded main function (1) • Handling incorrect argument counts (1) • Use of supplied arguments (3)	/5
C7	Error Handling • Use of assertions (2) • Use of conventional error handling techniques (3)	/5
C8	Pseudo-random number generation (5)	/5
C9	Dynamically allocated two-dimensional array handling • Structures (5) • Allocation (5) • Initialisation (5) • Deallocation (5)	/20
C10	Algorithm implementation • Logical Correctness (5) • Effectiveness / Efficiency of approach (5) • Correct output (5)	/15
B	Bonus	/10
Total:		**/100**

<u>SPY VS SPY</u>

The Utopian National Security Agency (UNSA) needs to infiltrate the rogue pirate compound. The UNSA needs to train their operatives how to avoid the automated dumb sentries posted inside the pirate compound. UNSA asked you to write a training simulator in the form of a game that will teach their operatives how to avoid the automated sentries while moving through the compound.



Player (Black Circle), Wall (Vertical line), Sentry moving up (Triangle), Sentry moving down (Inverted triangle), Start and end blocks (Grey coloured blocks), Empty areas (Empty blocks)

The aim of the game is to move the player from the right-side of the game area (start blocks) to the left side of the game area (end blocks). Inbetween the start and end blocks are walls, each with an opening. The player can only traverse the area through the empty areas.
You must place the logic of the game in the `SpySpace` **`namespace`**.

Initialisation:

- The size of the environment are given as command-line arguments.
- The world is filled in the following order.
    - Every square in the game area starts empty.
    - Every even-numbered column contains a set of walls.
    - Every odd-numbered column contains one sentry.
        - Each sentry starts in a random row.
        - Each sentry randomly faces either up or down.
    - A empty square must be created in a random row for each column that contains a wall.
    - The player starts in a random row in the last column (start blocks).

Moving:

- The player may move up (north), down (south), left (west) and right (east).
- The player may not move outside the game area or move on top of a wall.
- When a sentry spots the player, the player can only move once in every two rounds.

Updating:

- Sentries move in the direction they are facing (Either up or down).
- Sentries move one square per turn.
- A Sentry will spot the player, when the player moves into the column of the sentry and in the rows that the sentry is currently moving towards (facing in that direction).
- A Sentry cannot spot the player if the player moves into a row behind the sentry (Sentry is facing away from the player).
- When a sentry reaches the top row, or the bottom row it changes direction.
- Sentries cannot move outside the game world and cannot move on top of a wall.

End-game:

- The game succeeds when the player successfully moves into any one of the end blocks (last column).
- The game fails when the player and a sentry tries to occupy the same block.

Using your knowledge of good software engineering principles and C++, you must design and implement such a simulation. Consider the competencies as laid out in the mark sheet.

- C0 – Create a program design. The entire design, including UML, must model the initial placement of the sentries.
- C1 – Use your knowledge of basic C++ program structure and make sure to utilise the appropriate system libraries.

- C2 – Your program must be readable by human beings in addition to compiler software.
- C3 – Demonstrate your knowledge of the divide and conquer principle using functions.
- C4 – Your program must make use of programmer-defined source code libraries.
- C5 – Create a menu system that will ask the user which action they wish to take.
- C6 – The user must provide the required input used by the game (with error handling).
- C7 – Provide assertion based error handling as well as conventional error handling.
- C8 – Random numbers are used when initialising the 2D array.
- C9 – Use dynamic 2D arrays to implement your simulation. The game state must be output to the screen using printable ASCII characters.
- C10 – Pay careful attention to checking the handling of movement, update rules, and end conditions for the game.
- Bonus – Make use of C++11/14/17/20 features and/or enumerations in your code.

.