FACULTY OF SCIENCE

ACADEMY OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING

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

SEMESTER TEST 2
PAPER A

DATE: 2019-05-10

ASSESSOR(S) **PROF DA COULTER**

INTERNAL MODERATOR **MR BR GREAVES**

DURATION 3 HOURS **MARKS 100**

SURNAME, INITIALS (or ID NUMBER): _____

STUDENT NUMBER: _____

COMPUTER NR: _____

CONTACT NR: _____

NUMBER OF PAGES: 3 PAGES

REQUIREMENTS: NON-PROGRAMMABLE CALCULATORS ARE PERMITTED

Marker:			Submission overseen by:	
Sort Rank	Result	Moderation	Correction	**Submission**
				CD:
				USB:
				EVE:

Mark sheet		
Surname:		
Initials:		
Computer:		
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 • Allocation (5) • Initialisation (5) • Deallocation (5)	/15
C10	Algorithm implementation • Logical Correctness (5) • Effectiveness / Efficiency of approach (5) • Correct use of appropriate selection / iteration structures (5) • Correct output (5)	/20
B	Bonus	/10
Total:		**/100**

Markers Signature:_____

I declare that I am eligible to write this summative assessment according to the rules and regulations of the Academy of Computer Science & Software Engineering, the Faculty of Science and the University of Johannesburg. I declare that the work submitted is my own and that I have verified the correctness of my electronic submissions.

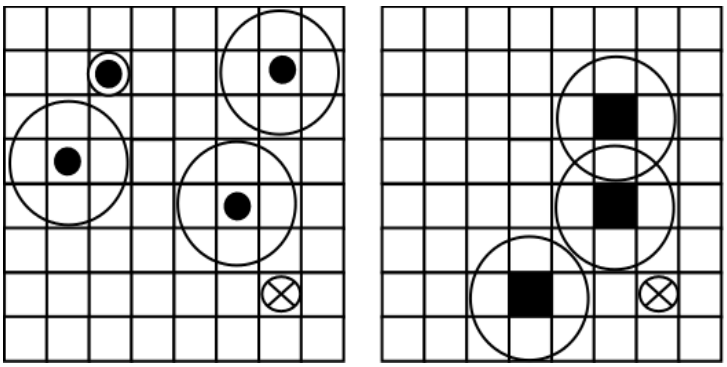
I UNDERSTAND THAT NON-COMPILING CODE CANNOT BE AWARDED A PASSING MARK

Student Signature:_____

.

Those bouys of mine

The Utopian Navy is planning for a mission in which one of its submarines the USS Crush Depth will need to get to a designated launch position in order to launch one of it's friendship missiles at target positions on the mainland. Unfortunately the launch position is on the middle of a heavily defended stretch of ocean with defence bouys floating on the surface and sea-mines suspended beneath the waves. Luckily, friendship missiles can be launched from either under water or from the surface. Furthermore defence bouys and sea-mines only have a 1 square detection radius. Additionally, bouys can only detect vessels on the surface and sea-mines can only detect submerged ones.



Player (Double circle) Empty squares (space) Bouys (black circles) Sea-mines (filled squares) Detection range (large circle) Launch position (x-filled circle)

In the game you will need to move a player controlled character around a layered two-dimensional playing area. One layer represents the surface while the other represents underwater. Your logic must be placed in the `SubSpace` **`namespace`**.

Initialisation:

- The size of the environment and the number of defences are specified via command line arguments.
- The number of defences must be an even number and is split evenly between bouys and sea-mines
- The player and launch position are placed randomly in the environment. The player cannot be placed directly on the launch position.
- Bouys cannot be placed in a one square radius of each other or the launch position and neither can sea-mines. It is possible for sea-mines and bouys to overlap.

Moving:

- The player may move one step in each direction. The player may not move outside of the game area. The player may chose to surface or dive/submerge instead of moving.
- The player can only see objects on the surface if they are surfaced and may only see objects below the surface if they are submerged.

End-game:

- The game ends in failure if the player moves into a detection radius on the same layer as them and in victory if they reach the launch position.

Consider the competencies as laid out in the mark sheet.

- C0 – Create a program design. Your UML must model the movement function.
- 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 which will ask the user which action they wish to take.
- C6 – The user must provide the required inputs 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 arrays.
- C9 – Use dynamic 2D arrays to implement your simulation. The game state must be output to screen using printable ASCII characters.
- C10 – Pay careful attention the handling of the two layers.
- Bonus – Make use of C++11/14 features, structures, and/or enumerations in your code.

.