UNIVERSITY OF JOHANNESBURG

FACULTY OF SCIENCE

ACADEMY OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING

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

SEMESTER TEST 2

DATE: 2020-05-22

ASSESSOR(S)	**PROF DA COULTER**
INTERNAL MODERATOR	**MR A MAGANLAL**
DURATION 3 HOURS	**MARKS 100**

NUMBER OF PAGES: 3 PAGES

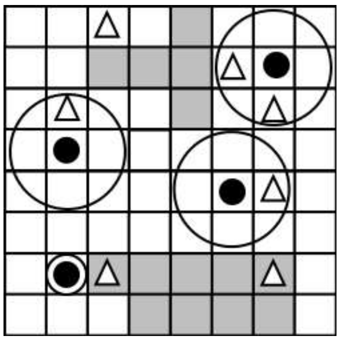
1. **You must complete this test yourself within the prescribed time limits.**
2. **You are bound by all university regulations please special note of those regarding assessment, plagiarism, and ethical conduct.**
3. **You may not directly use any code from any source including your own previous submissions. All code must be written by yourself during this the test.**
4. **You must complete and submit the** "`Honesty Declaration : Online Assessment`" **document along with your submission to EVE. No submissions without an accompanying declaration will be marked.**
5. **You may submit scanned pages as per the instructions on EVE**
6. **Your code together with the declaration must be submitted in a zip archive named in the following format.**
   ```
   STUDENTNUMBER_SURNAME_INITIALS_SUBJECTCODE_ASSESSMENT
   e.g. 202012345_COULTER_DA_CSC01A1_ST2.zip
   ```
7. **Additional time for submission is allowed for as per the posted deadlines on EVE.**
8. **No communication concerning this test is permissible during the assessment session except with Academy staff members.**
9. **This paper consists of 3 pages excluding the cover page**

If you already uploaded the Honesty Declaration into the Honesty Declaration prac, then you don't have to complete another one.

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 with structures • Allocation (5) • Initialisation (5) • Deallocation (5) • Structures (5)	/20
C10	Algorithm implementation • Logical Correctness (5) • Effectiveness / Efficiency of approach (5) • Correct output (5)	/15
B	Bonus	/10
Total:		**/100**
Markers Signature:____		

Boring Beetles and Bluegrass

Following the aftermath of "The Event" the Utopian Parks and Record Structures department is in need of a Serious Game to train the next generation of park staff. Unfortunately during "The Event" the care and maintenance of the parks has fallen behind. Two invasive species are threatening the ecosystem of the country's green spaces namely: The Rotbore Beetle which infests and kills trees and Appalachian Bluegrass which chokes the well manicured lawns.



Player (double circle), Lawn (empty squares), Infested Trees (black circles), Normal Tree (triangle), Bluegrass (filled squares), and Infestation Range (large circle)

In the game you will need to move a player controlled character around a two-dimensional playing area. Your logic must be placed in the `ParkSpace` **`namespace`**.

Initialisation:

- The size of the environment, number of trees, and initial number of bluegrass squares are specified via command line arguments.
- The player occurs in a random location in the first two columns of the game world.
- Trees cannot be placed on the player's location.
- Bluegrass cannot be placed on the player's location.
- Trees and bluegrass can occupy the same square.
- 25% of the trees start out infested with Rotbore Beetles.

Movement:

- The player may move one step in each direction (including diagonals). The player may not move outside of the game area.
- If the player moves onto a square containing a normal tree nothing happens
- If the player moves into a square containing an infested tree it is cured and the infestation is removed (the player does not occupy the square afterwards).
- If the player moves into a square containing bluegrass it is removed and replaced with lawn (the player does occupy the square afterwards).
- The player moves into a square containing both bluegrass and an infested tree the tree is cured and the bluegrass is removed (the player does not occupy the square afterwards).

Update:

- A square with lawn has a **cumulative** 5% chance of becoming filled with bluegrass **for each** bluegrass square directly adjacent to it (in the up, down, left, and right directions).
- A normal tree has a 25% chance of becoming infested with bore beetles if there is **at least one** infested tree in a one square radius around it (including diagonals).
- Cured trees can become infested again.
- Infested trees have 10 turns until they die. Curing them does not reset this counter to 10. If they get infested again the countdown resumes from where it left off. Dead trees are still infested with the beetles.

End-game:

- The game ends in failure if **either**
  - All the trees die **or**
  - 75% of the lawn squares are filled with bluegrass.
- The game ends in victory if **both** of the following are true
  - There are no infested trees **and**
  - There are no bluegrass squares.

Consider the competencies as laid out in the mark sheet.

- C0 – Create a program design. The entire design including 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 array and during the update function.
- 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 movement, update rules, and end conditions for the game.
- Bonus – Make use of C++11/14/17/20 features in your code.

.