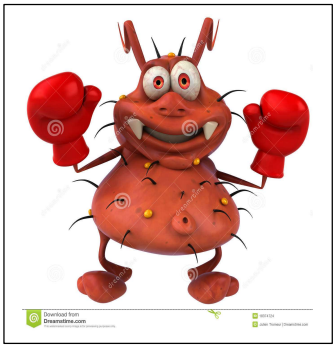
C++

Continuous Assessment
CA3 - Stage 1

Overall Weighting: 30%
Stage 1 (2%), Stage 2 (28%)
Semester 2 – 2024-2025

B.Sc. (Hons) in Computing in Games Development, Year 2, Semester 2
B.Sc. (Hons) in Computing in Software Development,Year 2, Semester 2



Assignment: A Bug's Life **Deadline: See Moodle**

This project is to be completed **in Pairs.**

Aim

Develop a system that will simulate the movement and interaction of various bugs placed on a Bug Board. The Bug Board is marked with grid lines so that it has 10 x 10 cells (squares). When we "Tap" (shake) the board it will cause all bugs to **move** in accordance with their specified behaviour. Bugs arriving on the same cell after a "Tap" will fight, and the biggest bug will eat all others on that cell. An end point is reached when there is only one bug remaining. You are also required to implement a console-based user interface using the menu items specified below.
(Later, in stage 2 you will implement a graphical interface using SFML).

Menu Items

```
1. Initialize Bug Board (load data from file)
2. Display all Bugs
3. Find a Bug (given an id)
4. Tap the Bug Board (cause all to move, then fight/eat)
5. Display Life History of all Bugs (path taken)
6. Display all Cells listing their Bugs
7. Run simulation (generates a Tap every tenth of a second)
8. Exit (write Life History of all Bugs to file)
```

Text File: "crawler-bugs.txt"

Format for a line (a record), delimited by commas (","), ending in newline.

Type of bug	'C' for crawler
ID for a bug	Unique integer ID value (e.g. 101,102,… etc.)
X coordinate	(X,Y) coordinate system where (0,0) is top left hand cell -
Y coordinate	and X increases to right (East), and Y increases as we go down (South)
Direction bug is facing	Direction: 1=North, 2=East, 3=South, 4=West (or better, use *enum*)
Size of bug	Measure of bug size (1-N), bigger bugs eat smaller bugs and grow accordingly

Sample file data:

```
C,101,0,0,4,2
C,102,9,0,1,3
C,103,9,9,3,1
```

Class Details (Specification)

You are required to create the following class using separate header (.h) and source (.cpp) files.

You **MUST** use the fields shown below (as named), and you may need to introduce more fields and/or functions to implement the logic described in the later functional specifications.

Crawler class

`int id;`	Identification number (id) for a bug (101,102, etc…)
`Position position;`	The "position" variable is of type struct Position which you define with two members x and y, (that represent a position on the board - (x,y) co-ordinate. (0,0) is top left cell on board.
`int direction;` *(or better, use enum class)*	The direction in which the bug is currently facing: 1=North, 2=East, 3=South, 4=West (or use *enum*)
`int size;`	Size of the bug (initially 1-20); biggest bug wins in a fight and others on same cell are eaten. Winner grows during a fight by the sum of the sizes of other bugs eaten.
`bool alive;`	Variable ('flag') indicating whether a bug is alive or not. The alive field for all bugs is set to 'true' initially. When eaten, this flag is set to 'false'. true == alive, false == dead
`list<Position> path;`	Path taken by a bug. (i.e. the List of positions (on grid) that a bug has visited).
`move() {}`	Implements the logic to move a bug (in response to a tap on the board). A Crawler bug moves according to these rules: - moves by 1 unit in the direction it is currently facing - if at edge of board and can't move in current direction (because the way is blocked), then, set a new randomly selected direction. (Repeat until bug can move forward). - record new position in the crawler's path history
`bool isWayBlocked() {}`	Checks if a bug is against an edge of the board AND is facing in the direction of that edge. If so, its way is blocked. [This method is used by the move() function]

Bugs are to be stored in a vector i.e. **`vector<Crawler*> crawlers;`**

This is a **vector of pointers to Crawler** objects. The Crawler objects **must be dynamically allocated** from memory (using new), and their addresses added to the vector.

Board class

Board class encapsulates the vector and cells. No access to the internal workings of board is to be 'leaked' outside the Board class, so no references or pointers to any internal objects are to be returned. Return only **copies** of data if required.

Features/Functionality *(Complete in the order shown below. (Increasingly challenging))*

You **MUST** use GitHub or GitLab, and make your lecturer a collaborator, and **commit & push** each feature as you complete it. Not having a verifiable record of regular incremental commits to your GitLab code repository will incur substantial penalties. You must share your Repository with your lecturer.

1. Initialise the bug Board.
Read the "crawler-bugs.txt" file and populate the **crawlers** vector. Assume the file contains valid data. Crawler objects must be dynamically allocated from the Heap using the 'new' keyword.

2. Display All Bugs
Display all bugs from the vector, showing: *id, position, direction, size, and status* -
in a neatly formatted, human readable form. E.g.

```
101 Crawler (3,4) 18 East Dead
102 Crawler (5,8) 13 North Alive
```

3. Find a Bug
User to be asked to input a bug id, and the system will search for that bug.
Display bug details if found, otherwise display "bug {id} not found".

4. Tap the Bug Board
This option simulates tapping the bug board, which prompts all the bugs to move. This will require calling the *move*() function on all bugs.

5. Display Life History of all bugs
Display each bug's **details** and the **path** that it travelled from beginning to death. The history will be recorded in the **path** field (which is a chronological **list** of positions). (A **list** container must be used)
101 Crawler Path: (0,0),(0,1),(1,1),(2,1),(3,1) Eaten by 203

(On **Exit -** Write the life history of all bugs to a text file called "bugs_life_history_date_time.out")

6. Display all Cells
Display all cells in sequence, and the **type** and **id** of all bugs currently occupying each cell.

```
(0,0): empty                    // meaning: cell (0,0) is empty
(0,1): empty
(0,2): Crawler 101, Crawler 103 // i.e. 2 Crawler bugs in this position
(etc…)
(1,0): Crawler 102
(1,1): Crawler 105
```

In order to achieve this, you must design and implement a mechanism to record which bugs are occupying which cells (squares). This is a challenging task.
You should discuss your approach to this with your lecturer before implementation.

7. (**Expand option 4) Eat functionality**
Implement functionality that will cause bugs that land on the same cell to fight. This will happen after a round of moves has taken place – invoked by menu option 4. (Tap ….). The biggest bug in the cell will eat all other bugs and will grow by the sum of the sizes of the bugs it eats. If the two biggest bugs in a cell are the same size, then select one at random to be the dominant one. The eaten bugs will be marked as dead ('alive=false'). We can keep 'tapping' the bug board until all the bugs are dead except one – the Last Bug Standing. Two or more bugs equal in size won't be able to overcome each other so the winner is resolved at random.
You should discuss your approach to this with your lecturer before implementation.

8. Run Simulation
Implement functionality to simulate the tapping of a board every 0.1 seconds until the game is over. Display progress on screen as simulation proceeds and write results to file.

In stage-2 of the project, we will add several different types of bugs to the system and we will refactor the design and implementation of the system accordingly. The specification for Stage-2 will be given in a separate document.

Grading Criteria

Understanding of code presented and ability to explain it is essential. Functional correctness, adherence to specification, quality of code (Maintainability, exploitation of standard library data structures, efficiency) and quality and functionality of User Interface will form part of the criteria for grading.

Evidence of consistent work recorded in Git history is required.

Upload Requirements

Upload to contain:

1. Zipped file containing all **source code** and **data files.** (Repo URL alone is not acceptable) ZIP the project folder for upload. Please name your project folder "Lastname_Firstname_CA3-Stage-1_Bugs" and your zip file "Lastname_Firstname.zip"Upload completed project as a single zipped file to Moodle by the deadline.

Late submission of this stage will have a "maximum mark of 40%" applied for stage 1.