Programming Studio 2
COSC2804
Assignment 3 Marking Guide
Semester 2 - 2023

Team Number 3

Marker Hermawan Mulyono

Video - 6 marks

Did all team members participate (rubric category "No marks" for any missing student)	**Yes**
Does the video clearly show functionality of the program?	**Good**
Are the design choices (to make the program efficient and accurate) clear?	**Good**
Are the explanations of design choices precise and correct ?	**Good**

BFS: vector for keeping track of the visited nodes? Not efficient.

Rubric Category	**Outstanding**	**6**

Teamwork - 0 marks

Has each team member committed to the GitHub repo?	**Yes**
Are the commits substantial (not just comments or boiler plate code)?	**Yes**
Does the commits and the task allocations match?	**Yes**
Does the checkpoint observations or GitHub indicate any group issues?	**Yes**

< comment if there are any Issues>

Mark individually or in group	**"Group"**

Base Implementation & Testing - 30 marks

Milestone 1: Black box test cases & menu navigation (8 marks)

Does the program compile with g++ (with -Wall -Werror flags)?	Yes

<add comments if you had to modify or remove error checking>

Adequate and functional test cases that covers all edge cases (need to be accurate)?	**Good**

Things to check (comment if you find missing tests):
- Irrelevant menu inputs
- Reading maze and parameters
- Building before creating maze - placing player before building etc
- Random maze generation in test mode (wrong if trying to do in normal mode)
- Wall follower Path - test mode

Menu is implemented correctly and error free **Good**

Things to check (comment if you find missing items):
- Can navigate to all relevant menus.
- Defensive programming used adequately (incorrect menu item inputs, incorrect operations - build maze before creating) COMMENT: Segmentation fault with input 1, 2, 2, 80 67 174, 7 9
- Adhered to style guide (specifically the use of structured programming)

Rubric Category	Good	7

Milestone 2: Reading maze and finding the path (10 marks)

Error-free implementation of read a maze, build, and place the player inside. **Good**

Things to check:
- Read maze done properly with error checking. COMMENT: Case sensitivity. Lowercase x should work.
- Flatten is implemented correctly (every tile placed needs to be supported by one below it)
- Build match the input (ignore any flipping of the maze)
- Player is placed at random in normal mode and furthest from start in test mode.
- Vector or array used to hold maze structurer (under the hood)

Error-free implementation of the wall follower algorithm? **Outstanding**

Comment if you find any issue

Clearly defines ADTs - Assumptions clearly documented? **Outstanding**

Things to check (comment if you find missing items):
- ADTs are clear with proper encapsulation.
- Contracts are clearly defined in the header where applicable (for "outstanding", students must use advanced concepts like compiler contraints (consts) and proper encapsulation where relevant).
- By design of the ADTs the maze structure should not be accesible by Agent solving the maze.
- Adhered to style guide (specifically the use of structured programming)

Rubric Category	Outstanding	10

Milestone 3: Generating the maze & cleanup (12 marks)

Error-free implementation of the functionality to generate a maze using recursion. **Basic**

Things to check: COMMENT: Cannot run maze random generation due to segmentation fault.
- Generate random maze functions correctly - both in normal mode and test mode.
- Used recursion correcly - for high marks with static reasoning logic.
- Check for correct randomization.
- Check both rectanguler and square mazes

Error-free implementation of the cleanup **Outstanding**

Things to check:
- Data structure used to hold changes must be efficient to resize (list, deque)
- Should not memorise a cube of blocks around the place where the maze is build (not efficient memory usage)
- Correct clearing the memory.
- What happens if another maze build in the same place as an existing maze (one can always clean maze before generating/reading another - but NOT build on top of another maze)

Clearly defines ADTs - Assumptions clearly documented? **Good**

Things to check (comment if you find missing items):
- ADTs are clear with proper encapsulation.
- Contracts are clearly defined in the header where applicable (for "outstanding", students must use advanced concepts like compiler contraints (consts) and proper encapsulation where relevant).
- should not allow to random modify elements that are memorised.

Rubric Category	Good	9

Enhancements - 9 marks

Milestone 4: Enhancements (9 marks)

E1 - Build maze on unflattened ground functional and error free? **No**

The program just readjust the heights, but it does NOT build around obstacles. Not to mention it was not verified due to segmentation fault. Still appreciate the effort here.

E2 - Shortest Path functional and error free? **Yes**

Comment if you find any issue

Rubric Category	**Good**	**5**

[Optional] Individual Comments (only if individually marked)

Student Number or Name

Comment on contribution or lack of contribution

Final Mark

[Optional] Individual Comments (only if individually marked)

Student Number or Name

Comment on contribution or lack of contribution

Final Mark

[Optional] Individual Comments (only if individually marked)

Student Number or Name

Comment on contribution or lack of contribution

Final Mark

[Optional] Individual Comments (only if individually marked)

Student Number or Name

Comment on contribution or lack of contribution

Final Mark