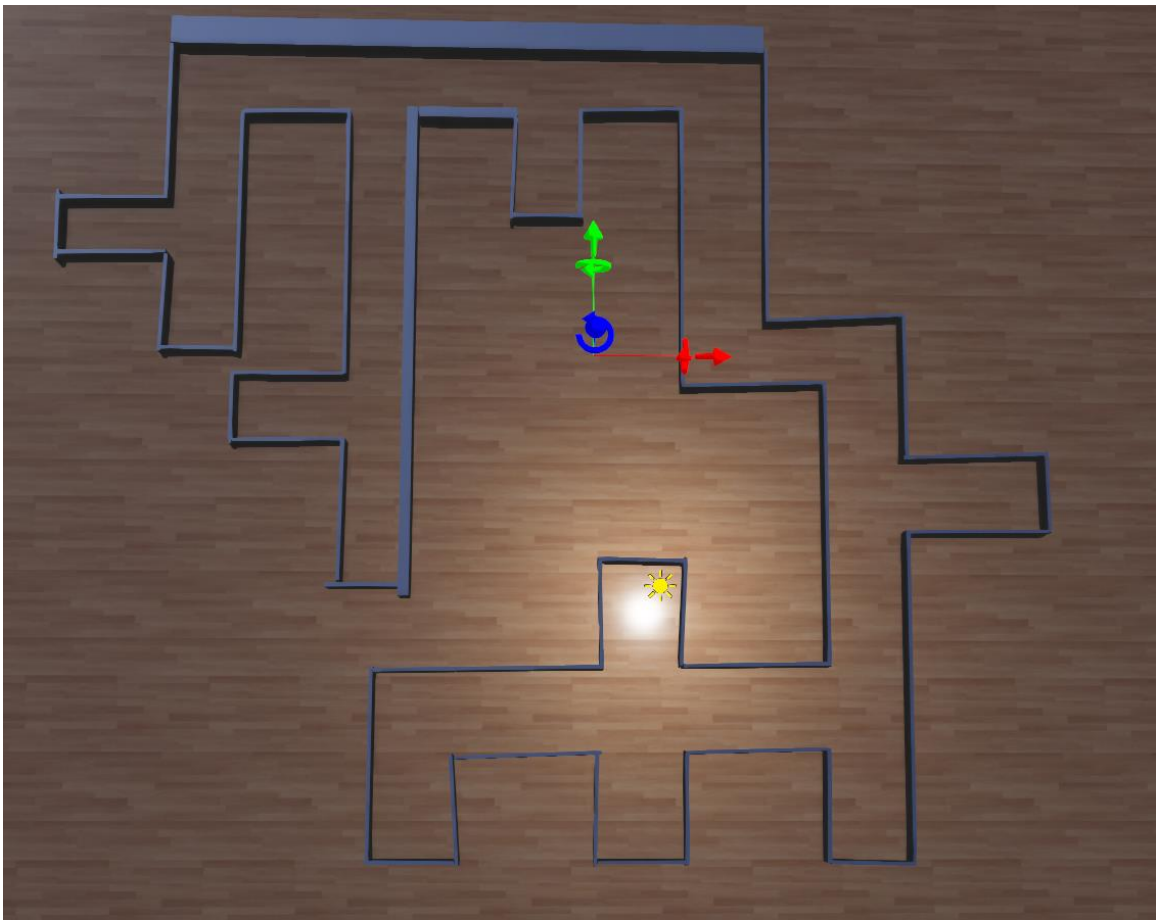
CSCI291 Project

Webots Mobile Robot Application: Maze Solving

Task Description:

In this project, you are required to implement a Webots[1] maze-solving robot to find the location with the maximum amount of light on a map in your assigned Webots project (see Figure 1 for an example), using any of the Webots robots, such as e-puck[2]. There are several maze-solving algorithms[3] and you have the freedom to select one or even design your own. You should equipe the Webots robot with sensors, such as a light sensor and a distance sensor, to implement the application. You can use any type and number of sensors; however, the implementation should be **cost-effective.**



Task Requirements:

Using a maze-solving algorithm, your sensor-equipped Webots robot should navigate the provided map, starting from the initial position (marked on the map) and reaching all the dead-end spots. All light sources are located at the dead-ends (depending on the map, there may be up to nine stations).

[1] https://cyberbotics.com/
[2] https://cyberbotics.com/doc/guide/tutorial-1-your-first-simulation-in-webots#add-an-e-puck-robot
[3] https://en.wikipedia.org/wiki/Maze-solving_algorithm

Once the light source with the highest intensity has been found, the robot should navigate directly from its current position to the station and come to a full stop.

Constraints:

- Your solution should not be hardcoded for the map provided.
- You cannot modify the provided maze environment in any way.
- You must implement the application in the C language.

Recommendations:

- Write dedicated functions to move the robot forward, left, right, and backward correctly.
- Write functions to read sensor readings and control the movement and alignment of the robot accordingly.
- Write functions to recognise different junctions, crossings, and dead ends, and control the movement and alignment of the robot accordingly.
- Use arrays or any dynamic data structures to store the path of the robot to each light source.
- Test each function individually.
- To ensure smooth movement control of the robot, you may need to add a delay function.

Submission:

- A concise report detailing the structure of the robot, the sensors used with explanations, and the selected maze-solving algorithm with rationale. The report should include the design solution of the implementation, listing all function headers and their interactions. **The report should focus on your implementation and not include a review of maze-solving algorithms or a description of the project task**. **Any code taken from third-party sources should be properly acknowledged**. The report should conclude with a reflection section summarising your experience, highlighting both the positive and less effective decisions made throughout the project.

- A demonstration video showing a complete cycle of your robot navigating to all light stations and then proceeding to the brightest one.

- Your GitHub repository showcasing the progress of your project, starting from week 4. Each amendment to the implementation should be submitted as a commit. A minimum of one commit per week is required.

- Your final code, with comments explaining its functionality.

Tentative Marking Rubric: (May alter slightly until week 4)

Marking item	Mark
Robot configuration	4
Sensor readings, junctions/crossings/dead-ends detection and appropriate actions	4
Identification of the brightest light source	1.5
Algorithm implementation and smooth robot navigation on the map	11
Navigating to the target station	4
Progress	5.5
Report	7
Demo presentation quality	3
Total	40