MTRN4110 20T2 Project Phase A Task Descriptions (week 1-3)
(Updated 28/5/2020)

1. Overview of the Course Project:

The main project of MTRN4110 20T2 is a simulation-based project adapted from the Micromouse competition. Webots will be used as the simulation platform throughout the course. You will design a mobile robot and implement a controller and a vision program to negotiate a maze autonomously in Webots. The project will contribute **55%** to your final mark of this course.

The project consists of four sequential phases:

- Phase A: Driving and Perception (week 1-3, 15%, individual)
- Phase B: Path Planning (week 4-6, 15%, individual)
- Phase C: Vision (week 7-9, 15%, individual)
- Phase D: Integration and Improvement (week 10-11, 10%, group)

This document will describe the tasks of Phase A.

2. Overview of Phase A – Driving and Perception:

The purpose of Phase A is to build the driving and perception modules of a mobile robot for the final maze-solving demonstration. You are required to complete the tasks of this phase by the end of week 3.

2.1. Expectations:

By the end of Phase A, you are expected to have:

- installed Webots on your computer properly;
- practised by following Webots tutorials (minimum - 1, 2, 4, recommended - 3, 5, 6);
- understood how to run simulations with robots and sensors in Webots;
- been able to create a controller for a robot that can execute a motion plan;
- been able to detect surrounding objects using onboard sensors of a robot.

2.2. Learning Outcomes Associated with this Assignment:

- **LO1**: Apply relevant theoretical knowledge pertaining to mobile robots including locomotion, perception and localisation using onboard sensors, navigation and path planning, for practical problem-solving
- **LO3**: Demonstrate practical skills in mechatronics design, fabrication, and implementation

3. Phase A Task Descriptions:

You will be given a Webots world file containing a five by nine maze (throughout this course the size of the maze will be fixed, see section 4 for more details) and an E-puck robot at the beginning of this phase. As shown in Fig. 1, the robot is placed at the centre of the top-left corner of the maze and heading towards the south of the view (throughout this course we will refer to the top, bottom, left, and right of the maze as North, South, West, and East, respectively).

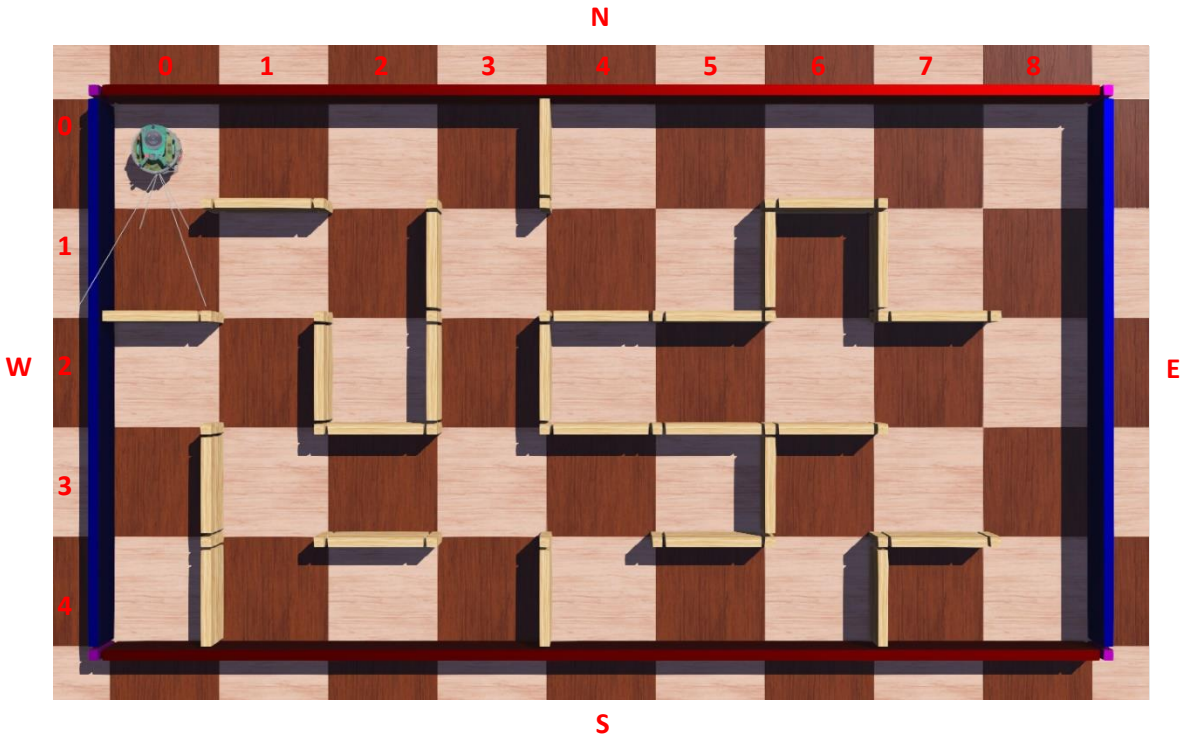

Fig. 1. An example maze layout and an E-puck robot

You are supposed to implement a controller for the robot. The controller should complete the following tasks once started.

3.1. Read a sequence of path plan from a text file and display it in the console

You will be given a text file named “PathPlan.txt” containing a sequence of motion commands, e.g.,

Fig. 2. An example path plan

The motion sequence starts with three characters specifying the initial location and heading of the robot. In the example shown above, the sequence starts with

00S

where the first character stands for the index of the row (0 - 4), the second for the column (0 - 8), and the third for the heading of the robot (N, E, S, W). The index always starts from 0 and from the top-left corner.

Following the three characters is a sequence of motions, represented by a string composed of three characters (F, L, R), where F stands for “Forward for one step”, L stands for “Turn left for 90 deg”, and R stands for “Turn right for 90 deg”.

In the example shown above, a sequence that directs the robot from the initial location to the centre of the maze will be (you can validate it by yourself)

FLFFLFRFRFFFLFRFLFFLFRFLFLFFF

In summary, the sequence of motions in the text file will be like the following:

00SFLFFLFRFRFFFLFRFLFFLFRFLFLFFF

There will be no spaces or any characters other than the ones mentioned above in the text file given to you.

You should read the path plan and display the same string in the console once the simulation is started.

You can choose to hard-code the motion sequence into your program and forfeit the marks associated. To do this, you need to define a variable to store the motion sequence at the beginning of your program. You should also EXPLICITLY indicate you will hard-code the motion sequence in the comment block at the top of your program. Failing to do so would affect the assessment of your submission.

```
/*
 * File:          z1234567_MTRN4110_PhaseA.c
 * Date:
 * Description:
 * Author:
 * Modifications:
 * Hard-coding: The motion sequence is hard-coded.
 */
```

Fig. 3. EXPLICITLY indicate hard-coding in the top comment block of your program

3.2. Display the initial location and heading of the robot, as well as the surrounding walls

You should display the step that the robot is executing. In the initial state, the step should be: Step: 00.

After showing the retrieved path plan, the next step is to parse the information and print the initial location and heading of the robot in the console: Row: 0, Column: 0, Heading: S.

Besides, you need to detect if there are any walls in the front, left, and right of the robot. Note that only the walls of the cell that the robot is located in are considered.

The E-puck robot has some onboard sensors installed by default. You should determine whether these sensors are sufficient to detect the walls or add any sensors provided by Webots as needed.

Display the existence of the left, front, and right walls. If a wall is detected, print Y; otherwise, print N. In the example of Fig 1, the detected walls should be: Left Wall: N, Front Wall: N, Right Wall: Y

In summary, you should display a set in the following format:

Step: 00, Row: 0, Column: 0, Heading: S, Left Wall: N, Front Wall: N, Right Wall: Y

3.3. Drive the robot following the path plan

Drive the robot according to the parsed path plan step by step.

If the motion step to be executed is 'F', move the robot forward for one cell.

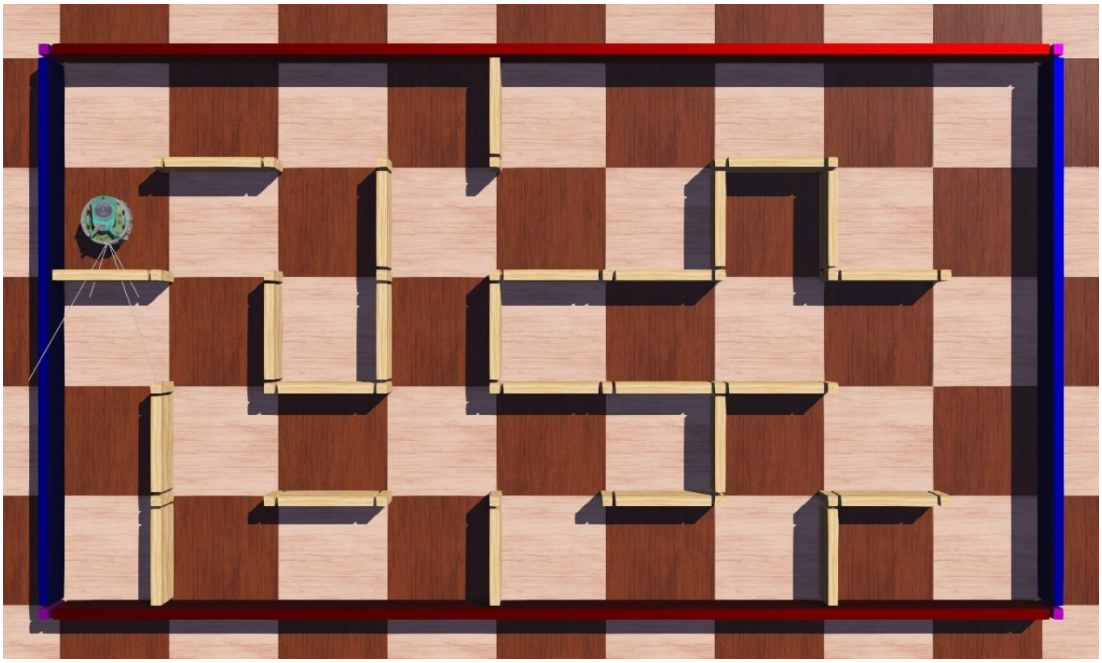

Fig. 4. Move forward for one cell

If the motion step to be executed is ‘L’, turn the robot 90 deg to its left.

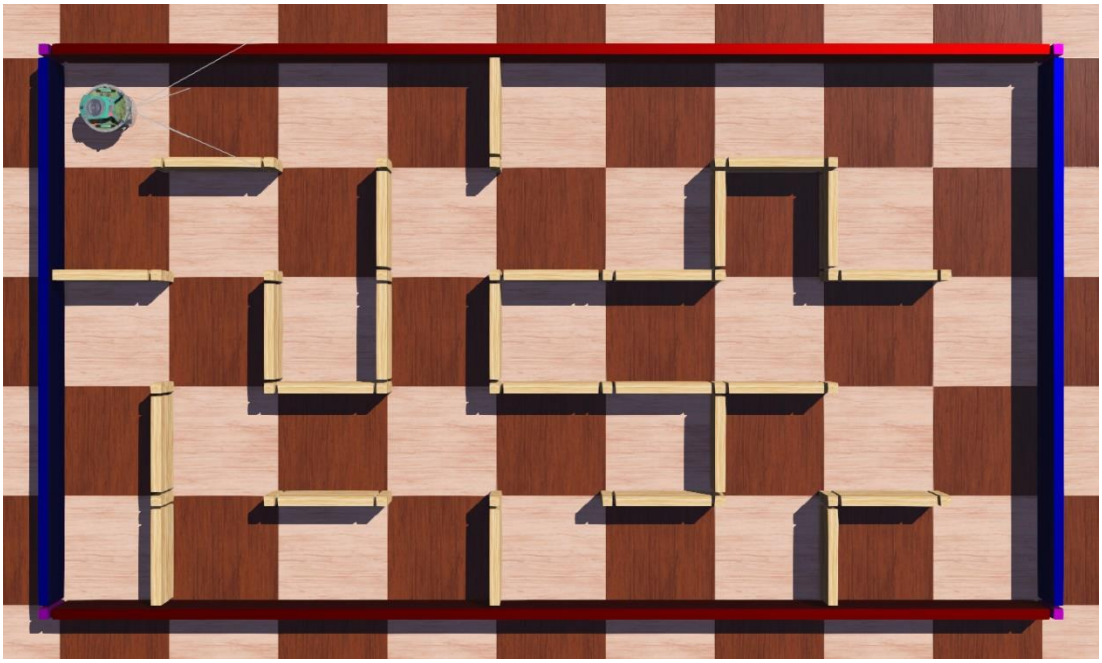

Fig. 5. Turn left for 90 deg

If the motion to be executed step is ‘R’, turn the robot 90 deg to its right.

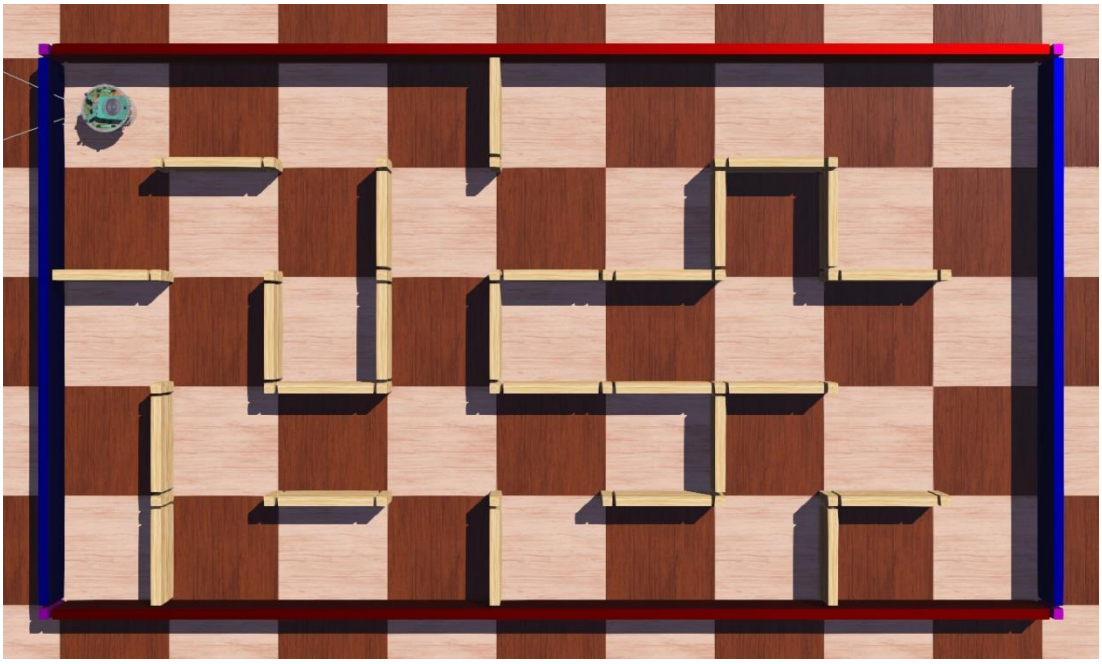

Fig. 6. Turn right for 90 deg

The robot should keep clear of the walls when moving. A penalty will be incurred if the robot hits the walls.

3.4. Display the location and heading of the robot, as well as surrounding walls after each motion step

At the end of each step, display in the console the new location and heading of the robot, as well as the left, front and right walls of the new cell that the robot is located in.

For example, at the end of the second step, the robot moves to the following location with its heading towards EAST,

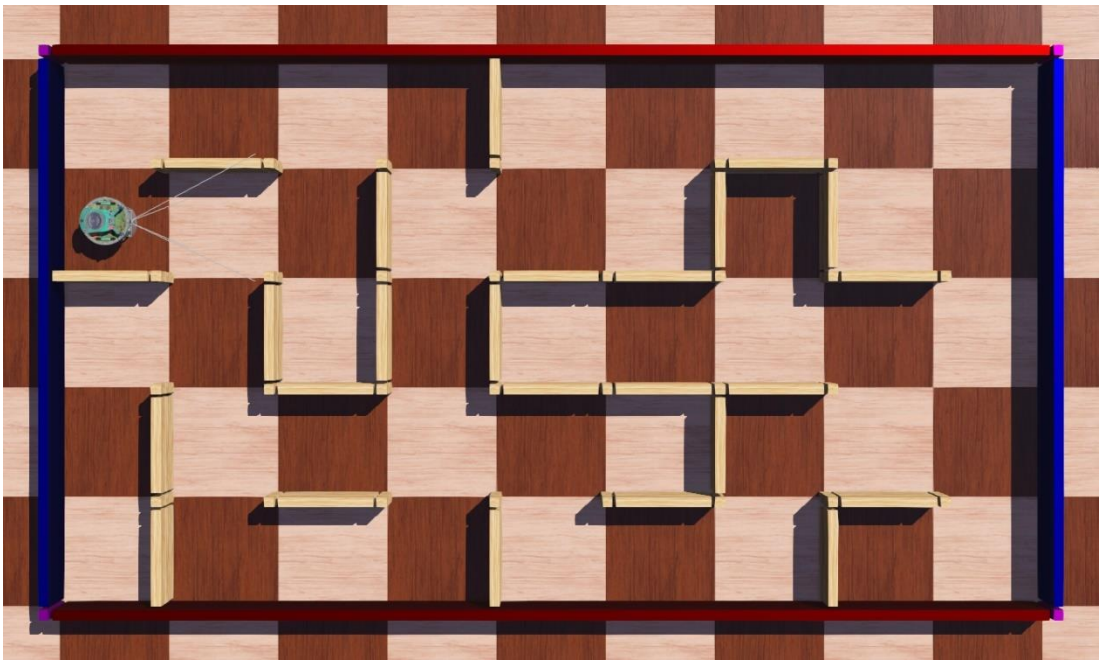

Fig. 7. Example robot location and heading

Display in the console:

Step: 02, Row: 1, Column: 0, Heading: E, Left Wall: N, Front Wall: N, Right Wall: Y

3.5. Repeat tasks 3.3 and 3.4 until all the motion commands are executed

Repeat tasks 3.3 and 3.4 until all the motion commands are executed. Print a message “Done – Path plan executed!” after the robot completes all the motions.

If the path plan is:

00SFLFFLFRFRFFFLFRFLFFLFRFLFLFFF

when all the motion commands are executed, the robot should have reached the following state:

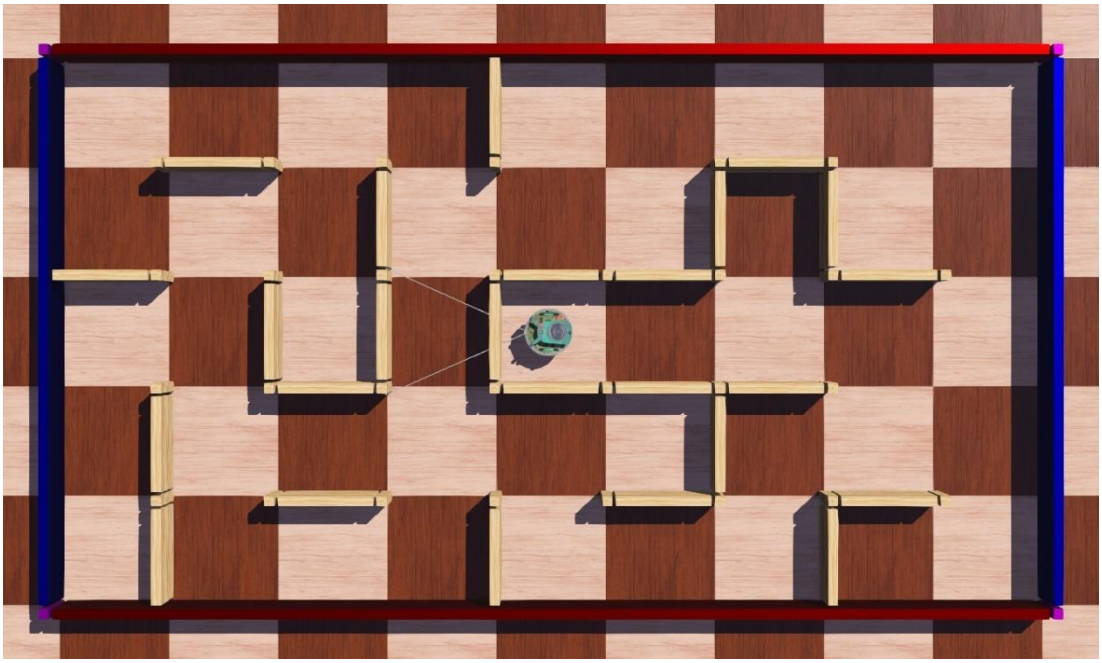

Fig. 8. Robot reaching the centre after execution of all motion commands

And messages printed in the console should have been:

Console

```
[z1234567_MTRN4110_PhaseA] Start - Read path plan from C:/Code/Webots/MTRN4110/PathPlan.txt:
[z1234567_MTRN4110_PhaseA] 00SFLFFLFRFRFFFLFRFLFFLFRFLFLFFF
[z1234567_MTRN4110_PhaseA] Done - Path plan read!
[z1234567_MTRN4110_PhaseA] Start - Execute path plan!
[z1234567_MTRN4110_PhaseA] Step: 00, Row: 0, Column: 0, Heading: S, Left Wall: N, Front Wall: N, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 01, Row: 1, Column: 0, Heading: S, Left Wall: N, Front Wall: Y, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 02, Row: 1, Column: 0, Heading: E, Left Wall: N, Front Wall: N, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 03, Row: 1, Column: 1, Heading: E, Left Wall: Y, Front Wall: N, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 04, Row: 1, Column: 2, Heading: E, Left Wall: N, Front Wall: Y, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 05, Row: 1, Column: 2, Heading: N, Left Wall: N, Front Wall: N, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 06, Row: 0, Column: 2, Heading: N, Left Wall: N, Front Wall: Y, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 07, Row: 0, Column: 2, Heading: E, Left Wall: Y, Front Wall: N, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 08, Row: 0, Column: 3, Heading: E, Left Wall: Y, Front Wall: Y, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 09, Row: 0, Column: 3, Heading: S, Left Wall: Y, Front Wall: N, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 10, Row: 1, Column: 3, Heading: S, Left Wall: N, Front Wall: N, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 11, Row: 2, Column: 3, Heading: S, Left Wall: Y, Front Wall: N, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 12, Row: 3, Column: 3, Heading: S, Left Wall: N, Front Wall: N, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 13, Row: 3, Column: 3, Heading: E, Left Wall: N, Front Wall: N, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 14, Row: 3, Column: 4, Heading: E, Left Wall: Y, Front Wall: N, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 15, Row: 3, Column: 4, Heading: S, Left Wall: N, Front Wall: N, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 16, Row: 4, Column: 4, Heading: S, Left Wall: N, Front Wall: Y, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 17, Row: 4, Column: 4, Heading: E, Left Wall: N, Front Wall: N, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 18, Row: 4, Column: 5, Heading: E, Left Wall: Y, Front Wall: N, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 19, Row: 4, Column: 6, Heading: E, Left Wall: N, Front Wall: Y, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 20, Row: 4, Column: 6, Heading: N, Left Wall: N, Front Wall: N, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 21, Row: 3, Column: 6, Heading: N, Left Wall: Y, Front Wall: Y, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 22, Row: 3, Column: 6, Heading: E, Left Wall: Y, Front Wall: N, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 23, Row: 3, Column: 7, Heading: E, Left Wall: N, Front Wall: N, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 24, Row: 3, Column: 7, Heading: N, Left Wall: N, Front Wall: N, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 25, Row: 2, Column: 7, Heading: N, Left Wall: N, Front Wall: Y, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 26, Row: 2, Column: 7, Heading: W, Left Wall: N, Front Wall: N, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 27, Row: 2, Column: 6, Heading: W, Left Wall: Y, Front Wall: N, Right Wall: N
[z1234567_MTRN4110_PhaseA] Step: 28, Row: 2, Column: 5, Heading: W, Left Wall: Y, Front Wall: N, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Step: 29, Row: 2, Column: 4, Heading: W, Left Wall: Y, Front Wall: Y, Right Wall: Y
[z1234567_MTRN4110_PhaseA] Done - Path plan executed!
```

Fig. 9. Messages displayed after execution of all motion commands

The robot should complete the motions within two minutes (the real runtime may differ due to different computing power; this time limit is based on the virtual time shown in the ribbon of Webots that is independent of computing power).

Fig. 10. Virtual time shown in the ribbon of Webots

3.6. Task summary:

Task	Description
1	Read a sequence of path plan from a text file and display it in the console
2	Display the initial location and heading of the robot, as well as the surrounding walls
3	Drive the robot following the parsed path plan
4	Display the location and heading of the robot, as well as surrounding walls after each motion step
5	Repeat tasks 3 and 4 until all the motion commands are executed

4. Specifications and Hints:

4.1. Specifications:

Maze:

1. At the beginning of Phase A, you will be given the same maze layout as shown in the example.
2. The initial location and heading of the robot will also be the same as illustrated.
3. This setup is for your practice. For assessment, you may be tested with a different (but similar) maze layout and the initial location and heading of the robot may also be different.
4. Consequently, the path plan for assessment may also be different, but should always direct the robot from its initial location to the centre of the maze.
5. The maze will always be the same size (five rows by nine columns) with closed borders.
6. The distance between neighbouring cells is always 0.165 m.
7. The thickness of the walls is always 0.015m.
8. The robot will always start from the centre of a cell (the origin of its coordinate is always placed at the centre).

Robot:

9. For Phase A, you MUST use E-puck robot for the tasks.
10. You CAN modify the robot by adding sensors to the <turretSlot> node or the <groundSensorsSlot> node. Note, however, that only the sensors that are provided by Webots are allowed.
11. You can modify the maze layout for your practice. But in the submission for assessment you should NOT change anything except the sensors of the robot and the controller.
12. The characteristics of E-puck robot can be found here. Note, however, that the wheel radius and axle length should be corrected as in the following table.

Characteristics	Values
Diameter	71 mm
Height	50 mm
Wheel radius	20 mm
Axle Length	56.6 mm
Weight	0.16 kg
Max. forward/backward speed	0.25 m/s
Max. rotation speed	6.28 rad/s

Implementation:

13. You MUST use either C or C++ to implement the controller. Note that Webots supports both C and C++ implementation, but does not support a mixed-use of C and C++.
14. For portability, you MUST use the pre-installed MinGW C/C++ compiler for Windows or Xcode for macOS.
15. The text file storing the motion sequence should be named "PathPlan.txt". You should define a path variable at the beginning of your controller program indicating the path of this text file.

    #define PATH_PLAN_FILE_NAME "**/PathPlan.txt"

    where ** is the ABSOLUTE path of the text file stored on your computer.
16. The naming rules for the world file and controller file are specified in section 5.

4.2. Hints:

1. Consult the lecturer/demonstrators if you are unclear about anything.
2. You can use standard printing functions for debugging. If you want to check the value of a variable during the simulation, you can print it in the console. If you want to see until which line the controller runs successfully, you can also add printing sentences at certain steps.
3. Try decreasing the speed of the robot if you use position control mode and the robot does not move to a position as specified.
4. Make the robot stop for a while before reading the sensors to get robust wall detection.
5. You should use forward slash / or double backward slash \\ instead of single backward slash \ to define the path of the file.

5. Assessment:

5.1. Submission of your work

You should zip your project folder and rename it as "z*******.zip" where ****** is your zID. Submit this zip file to Moodle.

In the folder, you should include both the <worlds> folder and the <controllers> folder.

In the <worlds> folder, you should include your world file, named as "z*******_MTRN4110.wbt" where ******* is your zID.

In the <controllers> folder, you should include your controller folder, named as "z*******_MTRN4110_PhaseA" where ******* is your zID.

5.2. Marking criteria:

This assignment will contribute 15% to your final mark.

You will be assessed twice with different setups. Your final mark will be the average of the marks of the two attempts., i.e.,

$$mark_{final} = 50\% * mark_{attempt\_1} + 50\% * mark_{attempt\_2}$$

Each attempt will be assessed by using the following criteria.

Task	Description	Marking (out of 100%)

1	Read a sequence of path plan from a text file and display it in the console	+10% if all correct, otherwise
		• (8% maximum)
		  ○ +2% each for every 3 consecutive characters correctly displayed. A character is considered correctly displayed if and only if this character and all the preceding characters are correctly displayed.
		• (no mark)
		  ○ if hard-coding the path plan into program

2	Display the initial location and heading of the robot, as well as the surrounding walls	+10% if all correct, otherwise • +1% for correctly displaying the initial location and heading • +3% for correctly displaying the left wall • +3% for correctly displaying the front wall • +3% for correctly displaying the right wall

3	Drive the robot following the parsed path plan	+40% if all correct, otherwise
		• (36% maximum)
		  ○ +5% for the first successful move[1]
		  ○ +5% for the second successful move[1]
		  ○ +2% for each additional successful move[1]

4	Display the location and heading of the robot, as well as surrounding walls after each motion step	+40% if all correct, otherwise
		• (18% maximum)
		  ○ +3% for correctly displaying the robot location and heading after the first move
		  ○ +3% for correctly displaying the robot location and heading after the second move
		  ○ +1.5% for each subsequent move (correctly displaying robot location and heading)
		• (18% maximum)
		  ○ +3% for correctly displaying the detected walls after the first move
		  ○ +3% for correctly displaying the detected walls after the second move

		○ +1.5% for each subsequent move (correctly displaying detected walls)
5	Repeat tasks 3 and 4 until all the motion commands are executed	• (-10% maximum) ○ -2% for every 10 secs exceeding the 2-minute virtual runtime limit

[1]A move is successful if the robot is correctly and fully moved to a new cell or rotated for 90 deg without colliding with the surrounding walls

5.3. Deadline

The submission deadline is 17:00 AEST 21 June 2020 (Sunday Week 3).

If your assignment is submitted after this date, each 1 hour it is late reduces the maximum mark it can achieve by 1%. For example, if an assignment worth 74% were submitted 10 hours late, the late submission would have no effect. If the same assignment were submitted 30 hours late, it would be awarded 70%, the maximum mark it can achieve at that time.

6. Additional Resources:

- Webots user guide: https://cyberbotics.com/doc/guide/index
- Webots reference manual: https://cyberbotics.com/doc/reference/index
- E-puck: https://cyberbotics.com/doc/guide/epuck
- C File I/O: https://www.tutorialspoint.com/cprogramming/c_file_io.htm
- C++ File I/O: https://www.tutorialspoint.com/cplusplus/cpp_files_streams.htm