



MICROMOUSE

Nurul Amelia (1103194032)
Robotics and Intelligent Systems



SYNOPSIS

The Micromouse project is a 3D robot simulator used to learn the basics of autonomous agent programming and apply theoretical knowledge. The autonomous robot is simulated to find a path to the center of a maze with 16x16 blocks. The four basic principles for an autonomous robot to achieve its goal are localization, mapping, path planning and motion control. As the robot moves through the maze, it uses a series of sensors to avoid obstacles and record its position in the maze using its initial position reference, the recorded maze map will be used to determine a possible path to the center every time it steps into the next cell.

FOUR BASIC PRINCIPLES

The four basic principles for an autonomous robot to achieve its goal.

LOCALIZATION

The process of determining the location and orientation of a robot (robot pose) with respect to its environment.

MAPPING

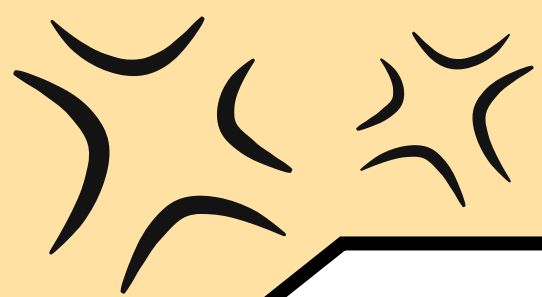
The act or process of making a map.

PATH PLANNING

Is used to find a geometric path from the robot's current location to the target location so that it can avoid obstacles.

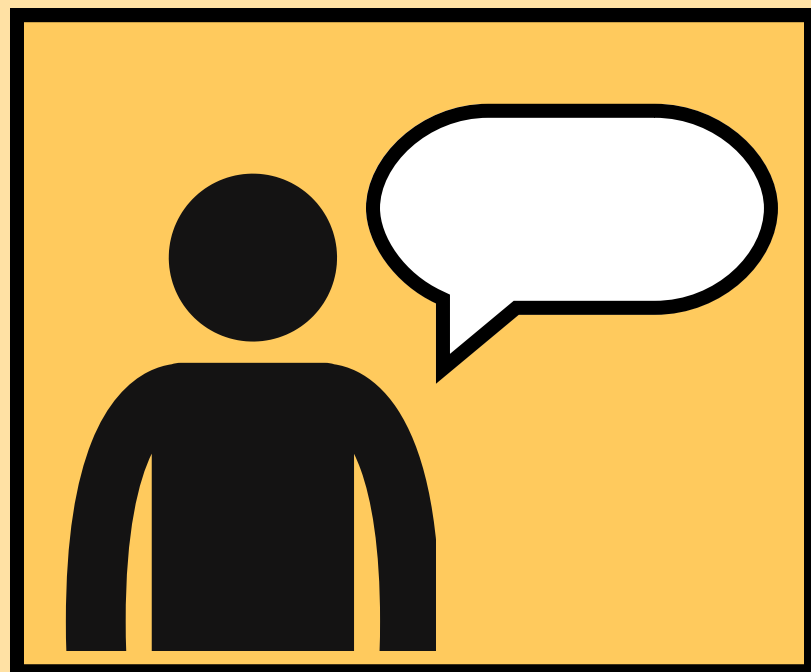
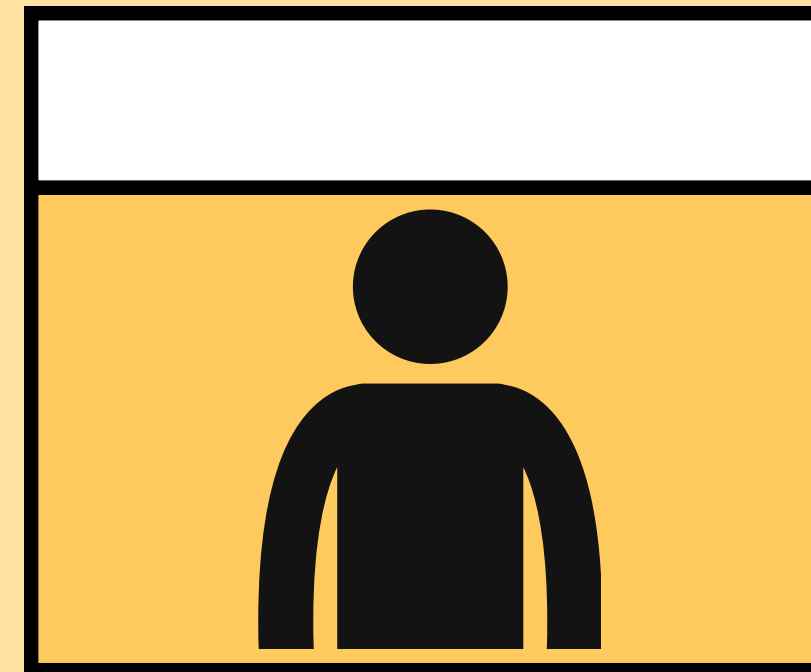
MOTION CONTROL

Motion control encompassing the systems or sub-systems involved in moving parts of machines in a controlled manner.



MAZE SOLVING

Move or find the best path to reach the center, using a search algorithm to calculate the shortest path.



PERFORMANCE

The shortest path is not always the fastest. Straight lines enable the mouse to accelerate.



WHAT IS MICROMOUSE?

<https://www.youtube.com/watch?v=NqdZ9wbXt8k>

In the video above the micromouse completes 5 paths to the center of the maze. The first track is a search track and the rest are race tracks. The mouse has no other information other than its starting position (which is always in the corner of the maze with the left side facing the maze frame), the size of the maze, and that it must reach the center of the maze. To calculate the best path to the center, the first run is used to map the maze and is called a search run. When a mouse does a race run, it may try to explore the maze on its way back to the starting cell. The best path to the center is not necessarily the shortest path because a mouse can go faster when it does not have to make a turn.



APPROACH

The project is divided into two basic development stages: the creation of a virtual environment on Webots, called a world, and the programming of autonomous agents.

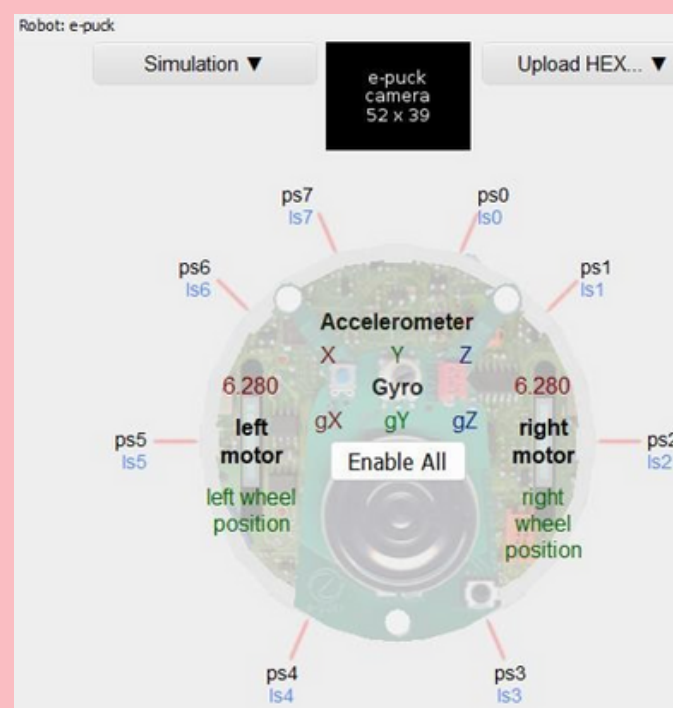
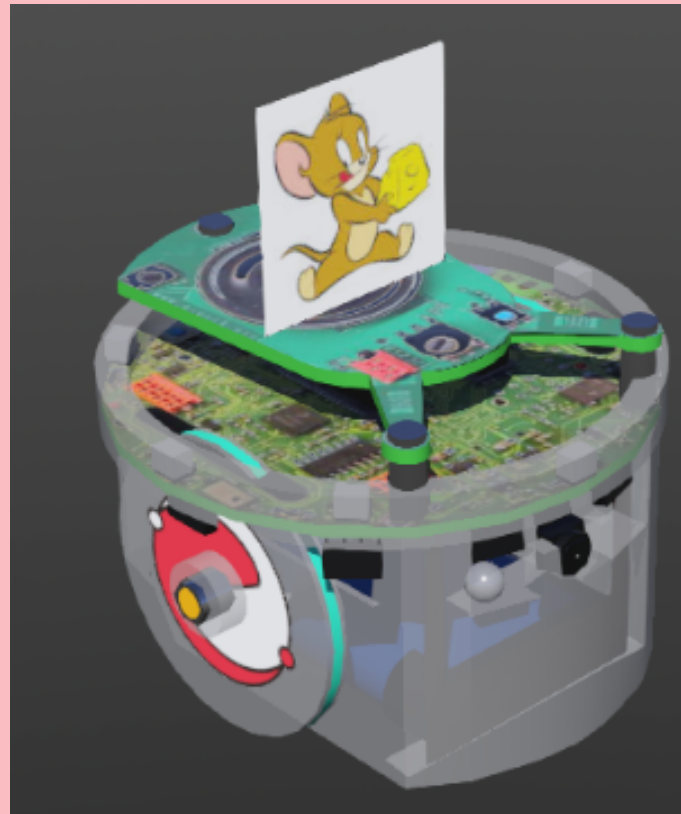
For the Micromouse world, I used a built-in Webots demo called "Rat's Life" which also uses a maze.

A function that makes it possible to read more than 400 mazes.

Using the e-puck robot.



E-PUCK



The e-puck robot has 2 motors on the right and left side (right motor and left motor) which functions as an actuator on the robot. Motor as an actuator for load the robot wheels. In addition, the e-puck robot also has several sensors including is an accelerometer sensor, which functions as a sensor to measure the acceleration or change in speed of the robot. Change in speed of the robot. Then, there are several sensors such as Ultrasound Sensor which is used to measure distance (there are 8 ultrasound sensors found on the e-puck robot). on the e-puck robot). The e-puck robot also has a camera with a revolution of 52 x 39 pixels.




THE CONTROLLERS



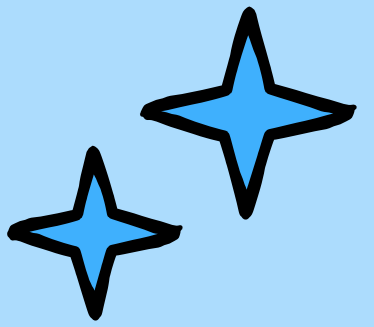
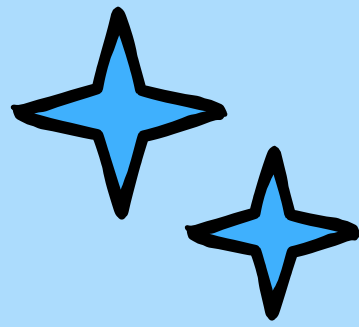
THE ENVIRONMENT

Half is created using the Webots World Editor UI (adding walls, wall joints, etc.) and half by programming the controller (in C) in a number of functions of which the most altered is maze_generation.c file containing the function for importing mazes from the maze archive.

THE ROBOT



The controller (in Java) is the “brains” of the Autonomous Agent and is responsible to guide the robot to the centre of the maze.



DEVELOPMENT

Basic principles behind agent logic.

ODOMETRI

Using wheel spin sensors and IR sensors for correction.

WALL

Using IR sensors.

DETECTION

LOCALIZ- TION

Using the starting position as a reference point, we can move through the maze and at each new cell, we can record the surrounding walls.

FLOOD FILLING ALGORITHM

This is a very useful search algorithm for path planning in mazes. By giving zero value to the weight of the destination cell, we can A the robot to follow the shortest path to the center.

CONTEST_MANAGER

For the maze, written in C language.



CONTEST_MANAGER.C

Basic routines of the program.



PROGRAM

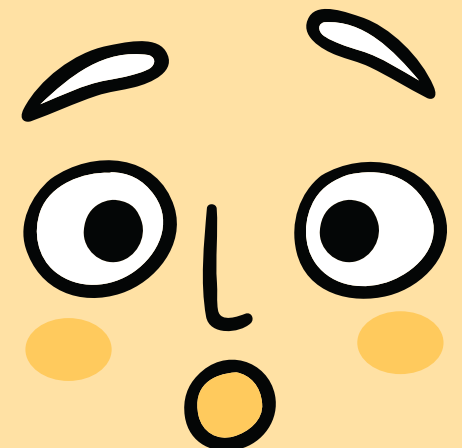
MAZE_GENERATOR.C

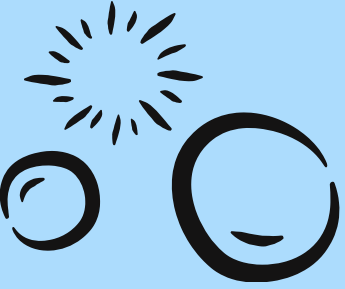
Maze generator.



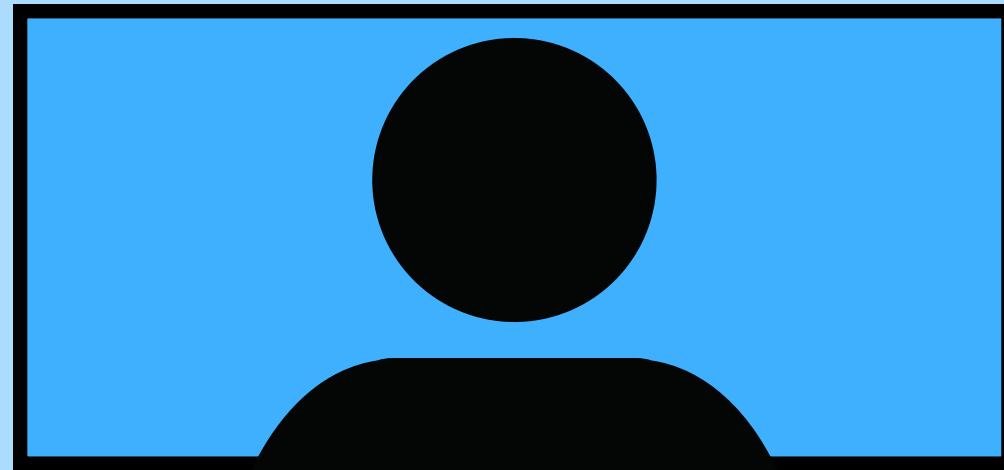
RATO.JAVA

For electronic chips, written in Java.



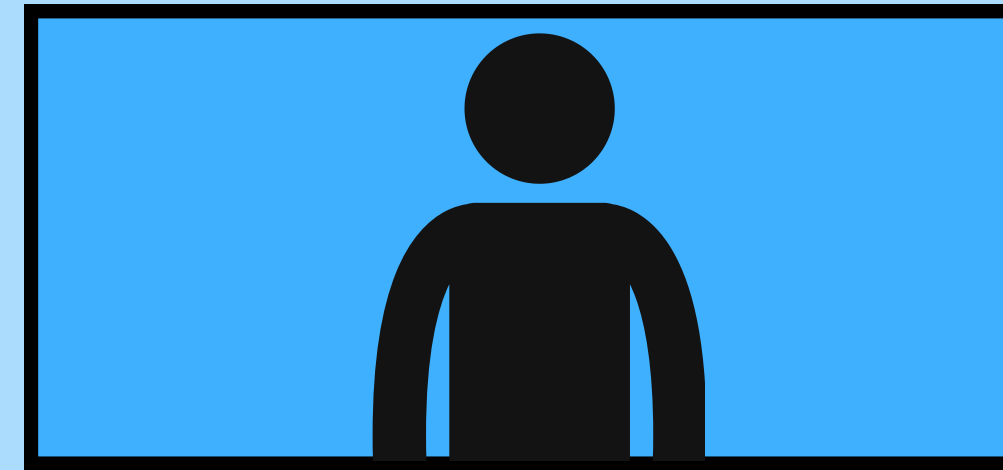


CREATING A MAZE



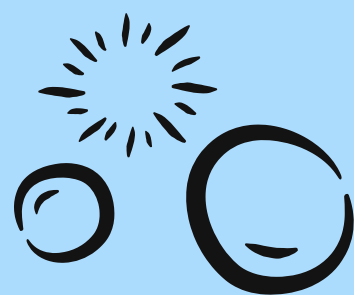
VOID GENERATE_SAVE_MAZE

Contains a table of the basic structure that contains information about the maze, the locations where the walls should be placed and the original position and orientation of the walls should be placed and the original position and orientation of the robot.
robot.

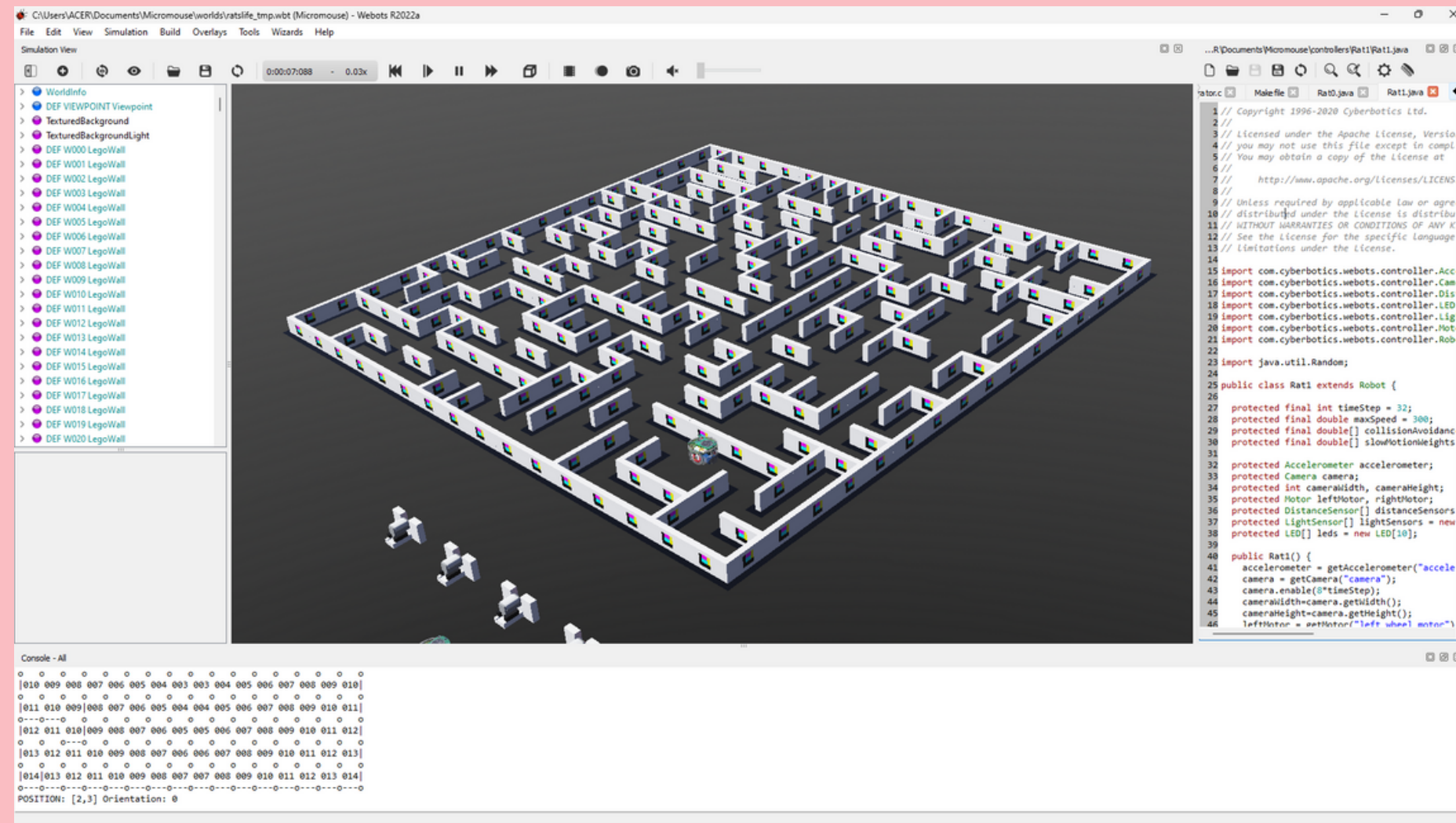


VOID PRINT_MAZE

Utility function for debugging.
It prints to the file debugPrintFile.txt file with the result of the above function.
(Needs a path change to work).



RUNNING RESULT



ANALYSIS

From the simulation results, the robot will record the path traveled. It should be noted that the best path to the center is not necessarily the shortest path, as a mouse can go faster if it does not have to turn. Sensors will detect obstacles in the maze in the form of walls so that the robot can avoid them.



THANK YOU!