# CSC 361 Project
# Phase #1

**Out date:** October 15, 2017
**Due date:** November 6, 2017

**Abstract**

Artificial Intelligence is increasingly used in many real-life fields. Path finding, for example, is a very popular application of AI. It is concerned with finding a valid path for an agent from an initial configuration to a goal configuration through an environment that may contain obstacles. Path planning is very important in games, map navigation, and robots. In the first phase of this project, you should use the AI knowledge you acquired while studying the course to model a path finding problem environment. In the next phase(s) you will need to write the search algorithms that will solve the path finding problem or an extended versions of it.

## 1 Introduction

We would like to formulate the robot map navigation problem (AKA the maze, or the grid). You have a map consisting of a grid of square cells. some of the cells are blocked (obstacles that the robot cannot walk through) and some are empty (free areas). The robot is located at one of the empty cells, and it wants to go to a goal cell. The robot is allowed to move one step at a time. The possible moves are to the north, south, east, and west.

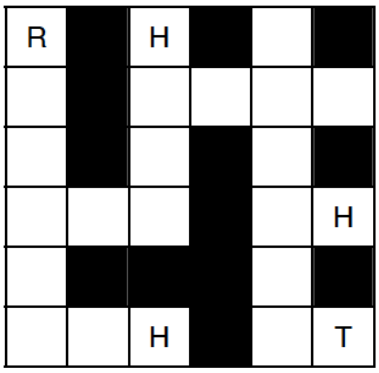


Figure 1: A maze example

Figure 1 shows an example of a maze. The 'R' represents the robot, the 'H' represents a square that contains a hole, and the T represent s a square that contains a treasure (the goal). The black squares are blocked.

To make things easier for you, we represent the map as a text file, where each character is a cell in the map. Map text files have two advantages: (1) it is easy to examine and understand the text-based input map files by reading their contents using a text editor, and (2) it is possible to generate or modify maps yourself. You may extend your code to generate a graphical interface that represents the maps and/or the robot moves.

In this phase, you are supposed to write a full formulation of the robot map navigation problem in your report, explaining the difficulties you faced. You also need to write a code that implements that formulation. Your code should be tested on some text-based input maps that will be provided to you. Then, you should provide the output files. The outputs of your code should include a log of the move attempts made by the robot, along with the final map that shows the final position of the robot.

In the following sections we explain the constraints and assumptions of the maze environment, the program interface, the formats and conventions of the project, the report, and the deliverables and submission process.

# 2 Constraints and Assumptions

The maze is a simple environment that you may have already been exposed to sometime during the course. In this environment, the goal is to make the robot find a path through the map that leads it to the treasure. To get there, the robot can use 4 move actions: `move-N`,`move-S`,`move-E`,and `move-W`.These actions will move the robot from its current cell to the next adjacent cell to the north, south, east, or west, respectively.

Collisions should not happen in this map environment, as well as disobeying other rules. Therefore, we need to put constraints on the robot's actions. The constraints are straight forward. For example, a robot cannot move to a blocked cell. It also cannot walk past the hole, i.e., once the robot falls into a hole, it cannot go out of it. It cannot go beyond the map's edges as well. These constraints need to be formalized in your problem model, in addition to other sensible constraints you think must be there. More about the constraints will be presented in the next sections.

We also make some assumptions to simplify things. These assumptions can be changed or removed in later phases of the project. The assumptions are:

- Currently there is only one robot and at least one treasure.
- The initial position of the robot and at least one of the treasures must be in an empty cell.
- There cannot be more than one copy of the robot or the treasure in one cell. For example: there cannot be 2 treasures in one cell. There can be a treasure and a robot in one cell though.

Given these assumption, however, you should not add more simplifying assumptions. For example, do not assume that the treasure cannot be in a hole.

# 3 Program Interface

Needless to say that your code should compile and run correctly. In this section we introduce how we pass the input to your agent and get the output. The executable file should accept two input parameters:

1. The name of the map file: `[mapFile]`.
2. The name of the command file: `[commandsFile]`.

The output should be stored in two files represented by the following parameters:

1. The log file: `[logFile]`.
2. The final map file: `[finalMapFile]`.

All of these files must be text-based, i.e. you can edit and view them using a text editor. All files' format will be explained in the next sections. Your program must be invokable from the command prompt. The program's name must be `'Agent'`. The usage should be as follow:

```
./Agent [mapFile] [commandsFile] [finalMapFile] [logFile]
```

or if you are using Java you should invoke it as an executable class as follows:

```
java Agent [mapFile] [commandsFile] [finalMapFile] [logFile]
```

or similarly as a jar file:

```
java -jar Agent.jar [mapFile] [commandsFile] [finalMapFile]
   [logFile]
```

where `mapFile, commandsFile, finalMapFile,` and `logFile` are the input and output files that must be located at the same directory as `Agent`. It is important to follow these conventions, since we will test your code using a script which assumes that you followed all the conventions. The following section explains the format of the files mentioned above.

# 4 Files Format

There are three kinds of files: the map file, the commands file, and the log file. The map file describes the map and its contents (the robots and the treasures) at a certain point during the search. The command file represents a list of moves which we will be providing you with in order to test how your code will move the robot on the map. Initially you can use the command file given here. The log file represents the output of each of the commands and whether they were applied successfully. In the following we explain each file format in detail.

## 4.1 Map File Format

The map file will have the following format:

```
[n]
[m]
[the cells]
```

where `[n]` is the number or rows, `[m]` is the number of columns, and `[the cells]` is a sequence of lines of characters representing the contents of the map grid. The cells are shown in `n` lines of characters. The number of characters in each line is `m`. The character can represent the static as well as the dynamic features of the cell. Static features cannot change in a cell. For example, blocked cells remain blocked, and holes remain holes. The static cell features are represented by the following characters:

- `' '`:the space character represents an empty cell, or a cell that the robot can move on.
- `'H'`:this represents a hole, or a cell in which the robot can fall. The robot can never go out of the hole once it falls in it.
- `'B'`:this represents a blocked cell. The robot cannot move to or originate at a blocked cell, as well as all other objects (e.g. treasures).

The cell can contain dynamic features on them in addition to the static features. The dynamic cell features are those that change. They are represented by the following characters:

- `'R'`:this is the robot. Currently it is the only object that can move on the grid.
- `'T'`:this is the treasure. Currently it cannot be moved around but in later phases we may change that.

Notice that there is only one robot and at least one treasure.

The dynamic features can exist in a cell with each other and with static features . For example, an empty cell that has a robot in it is simply `'R'`.Here is a list of the characters representing all cases of the cell:

- Empty cell with nothing in it: `' '`.
- Empty cell with a robot: `'R'`.
- Empty cell with a treasure: `'T'`.
- Empty cell with a treasure and a robot: `'U'`.
- Hole cell with nothing in it: `'H'`.
- Hole cell with a robot in it: `'X'`.
- Hole cell with a treasure in it: `'Y'`.
- Hole cell with a treasure and a robot in it: `'Z'`.
- Blocked cell: `'B'`. The blocked cell cannot contain any dynamic features.

An example of the text file representing the map of Figure 1 is as follows:

```
6
6
RBHB_B
_B____
_B_B_B
___B_H
_BBB_B
__HB_T
```

where the underscore symbol is used here to represent the space. Do not use the underscore symbol in your output final map file.

In this phase of the project we will follow these simple notations. However, we may change them in the next phases.

## 4.2 Commands File Format

This input file contains a list of actions that your code will try to achieve. For example, given the map of Figure 1, we may try to make the robot do a number of moves using the following command file:

```
move-N
move-S
move-S
move-S
move-S
move-S
move-E
move-E
```

However, some of these moves cannot be achieved due to blocked cells or being at the edge of the map, or for other reasons. For example the first move (`move-N`) will not be achieved since it tells the robot to go north while it is on the north edge of the map. In fact, this move attempt should cause the program to report a failed action attempt in the log file. The remaining moves should be OK, even though they will lead the robot to fall into a hole.

## 4.3 Log File Format

This output file reflects whether the commands were achieved or failed. So given the command file above, the lines of the log file should be as follows:

```
FAIL
DONE
DONE
DONE
DONE
DONE
DONE
DONE
HOLE
```

Notice that the robot fill into a hole at the end. So, for each command, your code should tell whether it was a successful or a failed move. For failed moves add a `FAIL` line. For the successful moves add a `DONE` line. After a failed move, assume that the robot did not make the move and is still in its previous position. Whenever you realize that you reached a goal state, you should add a `GOAL` line to the log file. If you fall into a hole, add a `HOLE` line. Notice that you may continue to receive commands in all cases.

In addition to this file, your code should also create the final map file, which represents the map after trying to achieve all of the moves in the command file. The final map file will have the same format as a map file, but should usually have different contents from the input map file.

# 5 Optional Extension Features

If you want to do something more, try copying your project and extending it by adding some more constraints and features to the environment. These features will not be part of the evaluation. However, you may get extra marks if you did them right. If you agree to submit an extended version of the code, you should incrementally add some (or all) of the following features: robot battery, power charging station, booster moves, and cell altitudes. Notice that the extensions are ordered, so you cannot do the second before the first, etc. Do not submit these with the main deliverables. Instead, submit them to me by email after you submit the main code. It is important to separate the mandatory part from the extension.

## 5.1 Battery

The robot will have a limited amount of power to move. Therefore, it may run out of power while it tries to do the tasks. Add a battery to the robot.

## 5.2 Charge Stations

If you implement the battery, you will notice that the robot may run out of power as it tries to do its tasks. Add a charger stations to the map (as a static feature), and add a `charge` action that will give the robot the ability to move further.

## 5.3 Booster moves

Sometimes the robot wants to go fast. By moving fast the robot can get to its goal in less time. However, the increase in speed means more force and therefore consuming more energy.

You should add a new kind of actions: the booster moves. For example, if you already have a `move-N, move-S, move-E,`and `move-W,` you should add four more actions: `move-N-B,move-S-B,`... etc, representing the booster moves. These new booster actions should move the robot two squares forward instead of one. Also, they should spend three times the amount of power.

### 5.4 Altitude

Some empty cells are at higher altitude than the cell where the robot is standing, so the robot may need to spend more energy to climb and reach them. Some cells are lower than the robot's cell that the robot does not need to spend the normal amount of energy to reach them. Think about a way to include this idea into your model. Implement it if you want!

## 6 The Report

In a small report (no more than 5 pages), write the formal problem definition as discussed in class and in the book. Namely, find:

- the state description, specifying explicitly the initial state using the given example.
- the actions and their transition model.
- the goal test.
- the path cost (the step cost of the actions.)

The report should include the following sections: an introduction, a problem description, the modeling (the formulation), a discussion, and a summary or a conclusion (if you have any). In the problem description section, you should write the problem in your own words. In the next section, you should specify the model. In the discussion, you should clearly explain the points of weakness and strength in your work. That is, write about any difficulties you faced in the project and any positive surprises you found. For example, if your code did not produce a correct log file, write about it and explain the possible reasons. You should also write about anything you think is positive and is important to mention. For example, you may have represented the maps graphically in your code! If so, then write about it. However, if you produce a graphical map, then you should do it in addition to the text based maps, and submit it separately. You may also add the extra features and explain how you used them in the discussion section.

## 7 Deliverables

You should submit the following:

1. The **report** that explains the work. **(40/100 marks)**
2. The compilable, runnable **source code** with instructions on how to build the executable. Also you must include the **executable file**. It should be the same file that results after compiling. **(20/100 marks)**
3. The **output files** that resulted after running the code on the provided input files. Notice that every student will be provided with input files for the test, and these files may differ from student to student. **(40/100 marks)**

# 8 How to Submit the Deliverables?

First, let's assume that your student ID is 433000123. Change this number to your real ID when you are about to submit. You should submit everything using LMS. You will find 3 links, one for each deliverable:

1. Under the `Project/Phase 1/REPORT-SUBMIT` link in LMS, you should submit a PDF copy of your report. The file name should be:

   `433000123.report.pdf`.

   The report will go through a plagiarism test, where the percentage of similarity with other reports (e.g. your colleagues', the Internet) is shown. If you have a high similarity percentage, then I will check the report further, since you might have copied something from other reports and did not cite it. **If I discover that this was the case, then it is considered cheating, which will cause you to FAIL the project, and perhaps the course.**

2. Under the `Project/Phase 1/SOURCE-AND-EXEC-SUBMIT` link, you should submit the **source code** and the **executable file**. The source code will also go through a code plagiarism test. All files in this case should be put in a **flat** ZIP file (i.e. there are no directories) called:

   `433000123.source-exec.zip`.

3. Under the `Project/Phase 1/OUTPUT-SUBMIT` link, you should submit the **output files** that resulted after running the code on the provided input files. Notice that every student will be provided with input files for the test, and these files may differ from student to student. Submit **your** output files only. I will check if these were your original output files by running your code. All files in this case will be put in a **flat** ZIP file called:

   `433000123.output.zip`.

   More about the input and output files naming convention might be given to you later.

Notice that submitting on time is your responsibility. Late submissions will *not* be accepted.

# 9 Important Guidelines

- Write your name, student ID number, section number, and project phase number on the report and as a comment in the code.
- Work individually. Discussing the project with your colleagues is encouraged as long as there is not a transfer of code or text. **Cheating will not be tolerated.**
- Use LMS to submit the soft copy for all of the deliverables, including the report, as described above. Emails will not be accepted.
- Make your report be as clear, precise and concise as possible.