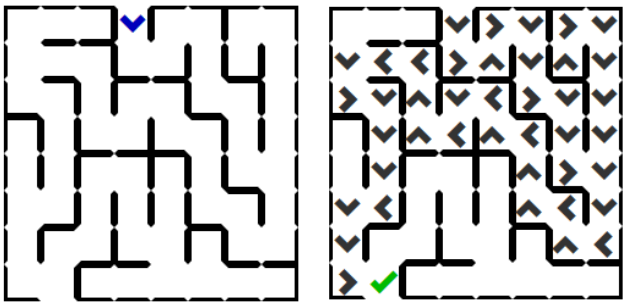
CS 105 | Machine Problem #1: aMAZEing

Due: Tuesday, Feb. 11 by 6:00pm

Introduction

One of the classic problems in a course on Computer Science involves using the computer to solving a maze. Solving a maze is particularly interesting since it illustrates exactly how a computer "thinks".

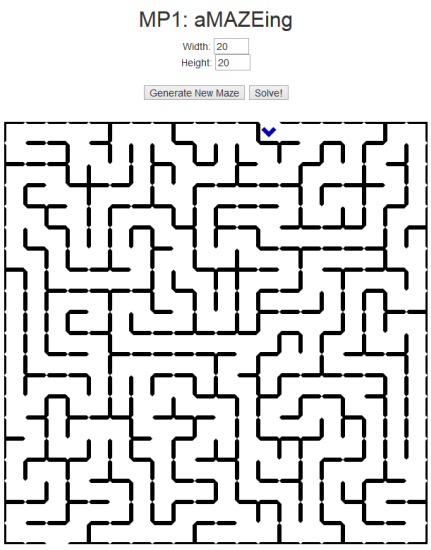


For this MP, you will be writing a solver for a maze in JavaScript. You will find that creating a very basic solver will be quite easy!

Remember: This MP is a **solo assignment**. You should review the academic integrity policy on the course website if you have any concerns about what is reasonable to do when completing this MP.

Getting Ready

Similar to the labs, we have prepared a base set of files for you. You can find these files on the CS 105 website, under the "Assignments" tab, and under the "MP1" link. The file you will download is a zipped file and needs to be **extracted**.



Once you have extracted the file, you will find five files. If you open **`aMAZEing.html`**, you browser should display a maze (similar to the image to the right ➔).

If you do not see the maze, you may have opened the zip file instead of **extracting** it. If you are certain you extracted the file, you may be using a browser that is configured not to run JavaScript. You should try to open the HTML page in a different browser to be sure this is not the case. In CS 105, we recommend using either Mozilla Firefox or Google Chrome.

Exploring the Maze Interface

Before you jump into solving the maze, click the "Generate New Maze" button on the HTML web page. You will immediately see a new maze appear and you can customize the size of the maze by entering different widths and heights. Each maze is completely unique – you will never see the same maze twice!

Besides the maze generator, the files we provided for you also include a number of informative error messages when something happens that the maze did not expect. For example, if you click the "Solve" button before you have programmed anything, you will find yourself with the following message:

Yikes -- something bad happened!

The makeMove() function did not actually make a move! You must add some code into `solver.js` in order to solve the puzzle.

This error message should make a little sense to you: there is a function, **`makeMove()`**, that you will need to code to complete this MP and you can find it in **`solver.js`**.

Moving Around the Maze

Remember from both your weekly reading assignment and your lab that, in order to edit JavaScript, you will need to open the file inside a **plain text editor**. This includes programs such as Notepad++ on Windows or TextWrangler on a Mac. You will find `solver.js` has one, empty function:

```
/**
 * Makes the next move, via a call to a move() function.
 */
function makeMove()
{
}
```

As the comment alludes to, in order to make a move in the maze you will need to call a `move()` function. The `move()` function is a function we have already written that actually moves the cursor around the maze based on your instructions. To use this `move()` function, you only need to tell it which direction to move: "Forward", "Left", "Right", and "Backward" – and you do that by sending it as a parameter to the function.

For example, the code to move forward in the maze is simply:

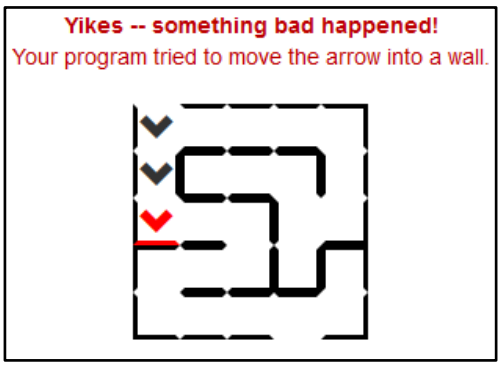
```
move("Forward");
```

You should go ahead and add this line to your JavaScript function. Your function should now look like the following:

```
function makeMove()
{
  move("Forward");
}
```

Save your JavaScript, reload the web page, and press "Solve". Two things should have happened:

- The cursor on the maze should have advanced forward until it hit a wall. This is exactly what we told it to do: every time it needed to make a move, it should move forward.
- The cursor on the maze should have eventually hit a wall. When it did, the cursor and the wall it hit will change color to red, an error message will appear, and the program will stop.



If both of these things did not happen, be sure to read the error messages that the maze gives you. For example, JavaScript is case-sensitive and the direction "forward" is not the same as "Forward". The maze only understands "Forward" and will give you an error if the "F" is not capitalized.

Now, onto those pesky walls…

Detecting Walls in the Maze

In addition to the **`move()`** function, there is one more function that is provided for you: **`isWall()`**. The **`isWall()`** function also takes in a single parameter, a direction of either "Forward", "Left", "Right", and "Backward", and returns **`true`** if there is a wall in that direction and **`false`** is there is not a wall.

Remember from lecture: a conditional takes a Boolean. Therefore, we can use the **`isWall()`** function directly inside of an if-statement. To check if there is a wall in front of us, and to do something if there is, our if-statement would be:

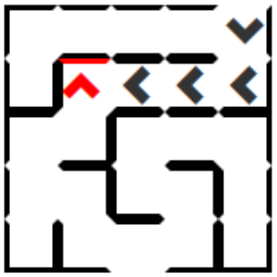
```
if ( isWall("Forward") == true )
{
  ...
}
```

Now that we can detect walls, rather than simply moving forward lets go ahead and turn right every time there is a wall in front of us. We can express this as **pseudo-code** first:

```
When I make a move:
  if there is a wall in the Forward direction:
    move Right
  otherwise:
    move Forward
```

The pseudo-code is written in JavaScript with the following code:

```
function makeMove()
{
  if ( isWall("Forward") == true )
  {
    move("Right");
  }
  else
  {
    move("Forward");
  }
}
```



Save your JavaScript, reload your HTML page, and run your new solution.

For mazes where you go straight and make a right-hand turn, we have made progress! For a small enough grid, you may find mazes where your code already gets you through the maze successfully:



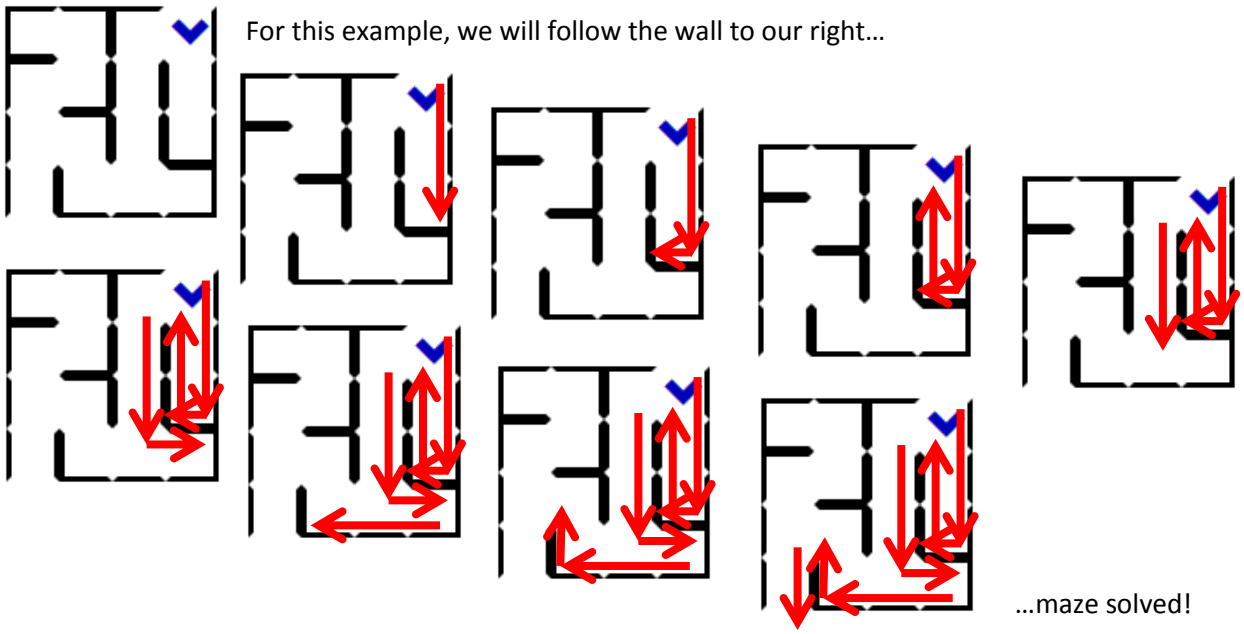

However, the goal of this MP is to create a maze solver that will work for any maze that is generated. We will tackle that next.

Solving the Maze

When you generate a maze on the HTML web page, you might notice that every maze that is generated in such a way that there is **never a cycle within the maze**. That is, there are never cells in way that you can continuously move around in a circle – every path either leads to a dead-end or the exit.

For mazes that do not contain a cycle (an "acyclic maze"), there is a clever trick that will guarantee that you will solve the maze. Imagine that you are a human in the maze and start by touching the first wall that you see. Simply **follow that wall**, never lifting up your hand, **and you will eventually find the exit**.

Consider an example maze:



If this "hugging" of a wall makes sense to you, then you already have an **algorithm** to solve this MP. All you need to do is to **code this algorithm** in JavaScript, test it on a variety of mazes to make sure everything works, and then you have finished your first JavaScript MP!

If it helps, write your algorithm in **pseudo-code** first on a sheet of paper (not even using a computer). Then, use this MP description and your class notes to translate that pseudo-code into JavaScript. Type it into your **`solver.js`** file only once you have a written program on paper that you believe is correct.

Remember, we have provided two functions for you: **`move(direction)`** and **`isWall(direction)`**. Your solution will need to use them both.

Scoring and Submission

This MP will be graded for correctness of solving mazes. If your algorithm solves any generated maze, you will receive the full 20 points for this MP (your solution does not need to be optimal, following the algorithm described above will get you full credit).

To submit MP1, follow the link on the MP1 page on the CS 105 website for the submission page.