WAD Trial Lab Test 2 (1.5 hours) [30 marks]

Download resources: https://smu.sg/hr3

General Instructions:

- You can refer to any offline resources already on your laptop, but you must disable all networking and Bluetooth connections during the test. You must not communicate with anyone via any means during the test.
- Just before the test, you will be given instructions by the invigilator as to how to obtain resource files required for the lab test and how to submit your solutions.
- No questions will be entertained during the test. If necessary, make your own assumptions.
- You are allowed to use only standard PHP classes and functions in your solutions – do not use any third party libraries.
- Use meaningful names for classes, methods, functions and variables, as well as indent your code correctly. Use 4 spaces for indentation. Otherwise, you may attract penalty of up to **20%** of your score for the corresponding question.
- You **MUST** include your name as author in the comments of all your submitted source files. Failure to do so WILL attract a penalty of up to **20%** of your score for the corresponding question.

  For example, if your registered name is "TAN So Tong" and email ID is tan.sotong.2017, include the following comment at the beginning of each source file you write.

```
<!--
      Name:  FAN Bing Bing
      Email: fan.bingbing.2017
-->
```

- You may wish to comment out the parts in your code which cause errors. But commented code will not be marked.
- Unless otherwise stated, you can assume that user inputs are in the correct format.
- Instructions are given for WAMP users considering default setting – the setting that we support for this course. If you are using MAMP, you would need to make necessary changes by yourselves (e.g., modifying connection information in `ConnectionManager.php`).

DO NOT TURN OVER UNTIL YOU ARE TOLD TO DO SO

Load Database & Configure Connection Manager

Given:

- database/
  - ConnectionManager.php
  - trial_lt2.sql

Instructions:

- Import **trial_lt2.sql** into your local MySQL database (via PHPMyAdmin, WorkBench or by other means).
- In **ConnectionManager.php**, verify that the username, password, port number are correct.

Question 1: Student List and Search (Difficulty Level: *) **[8 marks]**

Given:

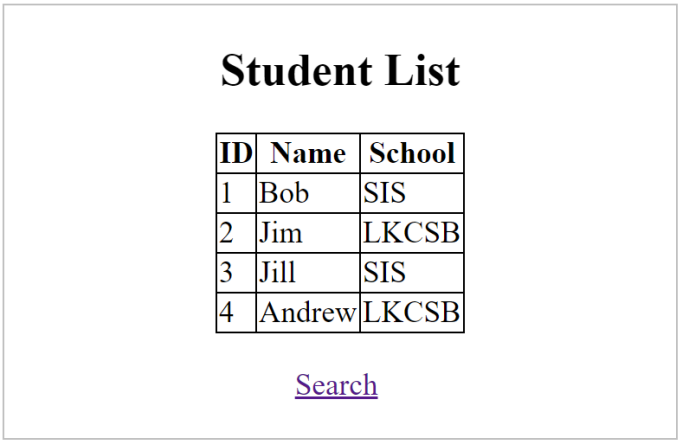
- `StudentDAO.php` ***(partial)***
- `index.php` ***(partial)***
- `search.php`, `process_search.php`, `Student.php`, and `student-list.css` **(complete)**

Part A: Complete `getAll()` in StudentDAO.php (3 marks)

Complete `getAll` method in `StudentDAO.php` such that it selects all students from the database and returns them as an array of `Student` objects.

Part B: Complete `index.php` (3 marks)

Complete `index.php` such that it displays the list of all students as shown below:



Student List

ID	Name	School
1	Bob	SIS
2	Jim	LKCSB
3	Jill	SIS
4	Andrew	LKCSB

Search

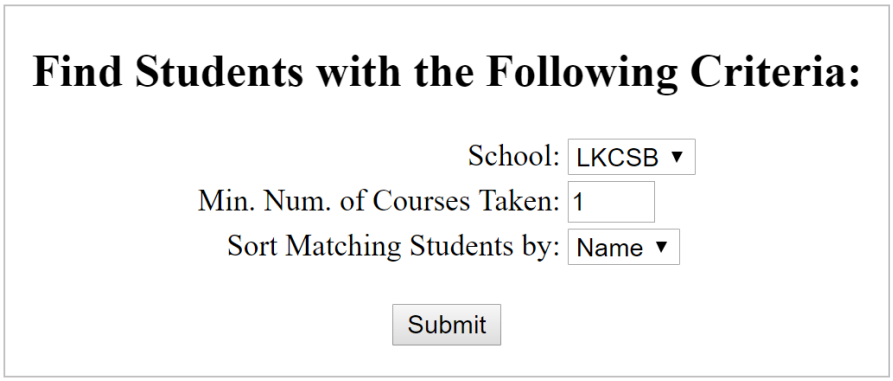
`index.php`

Part C: Complete `getStudents()` in StudentDAO.php (2 marks)

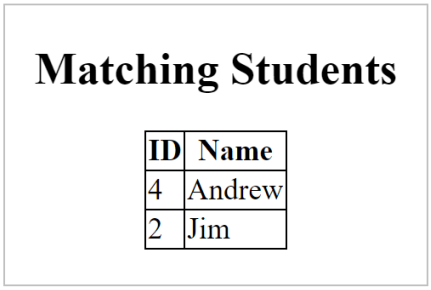
`getStudents()` in `StudentDAO.php` takes in three arguments `$school`, `$min_course_count` and `$sort_by_id`. It selects all students from a particular `$school` who takes at least `$min_course_count` courses from the database. The students are sorted based on their ids if `$sort_by_id` is TRUE. Otherwise, they are sorted based on their names. The method returns the sorted students as an array of `Student` objects.

After this method has been completed, student search functionalities implemented in `search.php` and `process_search.php` would be completed.

Example 1: Find all students from LKCSB that take at least 1 course. Sort the students based on their names.

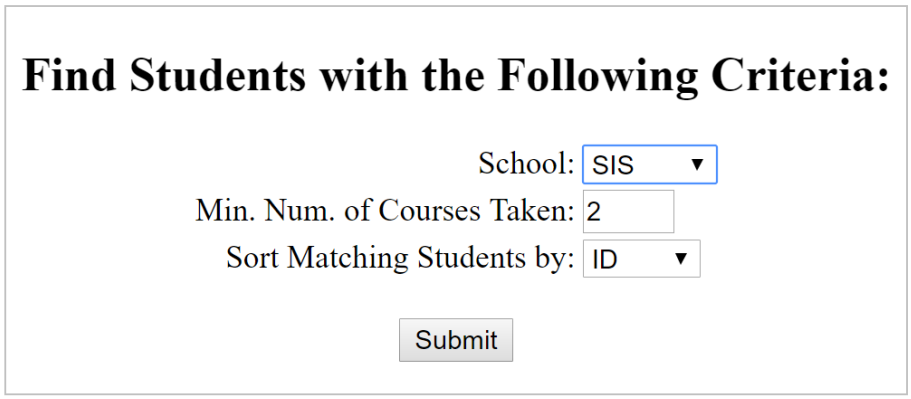


search.php

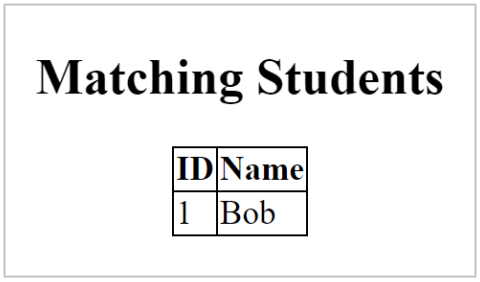


process_search.php

Example 2: Find all students from SIS that take at least 2 courses. Sort the students based on their ids.



search.php



process_search.php

Question 2: (Difficulty Level: **) **[15 marks]**

Given:

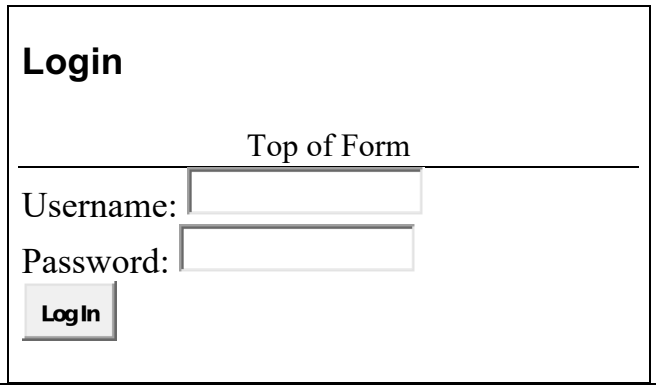
- `q2/include/`
  - `common.php, Account.php` (**complete**)
  - `AccountDAO.php` ***(partial)***
- `q2/`
  - `login-view.html` (**complete**)
  - `login.php` (**complete**)
  - `home.php` ***(partial)***
  - `reset-view.php` (**complete**)
  - `reset.php` ***(partial)***

The user's journey

- `login-view.html -> login.php -> home.php`
- `reset-view.php -> reset.php`

Part A: Complete Authentication (5 marks)

`login-view.html` page allows the user to log in using **username** and **password**.



`Figure 4: login-view.html`

Upon clicking on the SUBMIT button, the user is taken to **`login.php`** which performs username & password **authentication** by calling the **`authenticate()`** method defined in **`AccountDAO.php`**.

In **`AccountDAO.php`**, complete the **`authenticate()`** method to perform the following: **(2 marks)**

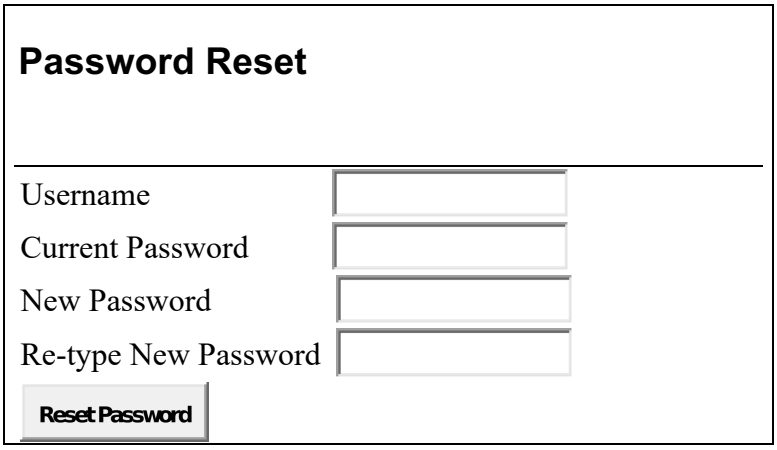
- It must make use of the **`retrieve()`** method in **`AccountDAO.php`**.
- It must perform authentication. Only if the **username** and **password** combination is correct (verify against the database's **account** table), the authentication is known to be **"successful"**.
- If the authentication is **successful**, return **Boolean true**.
- Else, return **Boolean false**.

You can use the following accounts for testing purpose:

S/N	Username	Password
1.	john	john123
2.	paul	paul123
3.	ringo	ringo123

Part B: Complete Password Resetting (10 marks)

`reset-view.php` page allows the user to reset his password.



`Figure 5: reset-view.php`

The user must fill out all FOUR (4) input fields. Upon clicking on the SUBMIT button, the user is taken to **`reset.php`** which performs **form field validation**.

TASK A: Complete reset.php (4 mark)

- First, it checks for all **EMPTY** fields. If any of the fields are empty, **$errors[]** array is populated. The array is saved as a **SESSION variable** and the user is re-directed to **`reset-view.php`**.

Missing Field	Error Message
username	Username field is empty
current password	Current password field is empty
new password	New password field is empty
new password (verify)	Verify new password field is empty

- Next, **`reset.php`** calls the **`AccountDAO`**'s **`authenticate()`** method to see if the username & password combination is correct. If not, it enters an error message ("wrong username/password") into $errors[] array and follows the same re-directing procedure as above.

- Next, **`reset.php`** checks to see if the two "new" passwords match. If not, it follows the same re-directing procedure as above. If the two "new" passwords do not match, enter an error message ("Your new passwords do not match") into $errors[] array.

- If **form field validation** and **authentication** are both successful, **`reset.php`** calls the **`AccountDAO's reset_password()`** method to complete password update in the database. It then prints "Success!" to the screen.

TASK B: Complete AccountDAO.php (3 marks)

In **`AccountDAO.php`**, complete the **`reset_password()`** method to perform the following:

- This method is given two arguments: **account ID** and **new password**.
- This method must perform table update.
- This method does not return anything.

TASK C: "Protect" home.php (3 marks)

In **`home.php`**, complete the code so that:

- When a user attempts to access this page directly without authentication (e.g. via login page with a valid username/password combination), this page will re-direct the user back to **`login-view.html`**.

Question 3: Path Finder (Difficulty Level: *)** **[7 marks]**

Given:

- `maze.php`, `Point.php`, `animation.js`, `maze.css` and a few image files (`mario.png`, `obstacle.png`, `empty.png`, `end.png`, and `red_X.png`) ***(complete)***
- `utility.php` **(partial)**

Instructions:

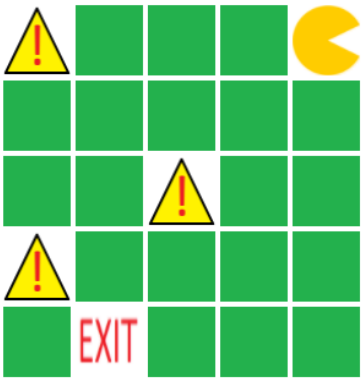
Your task is to **complete four functions in `utility.php`.** These functions would be called by `maze.php` to render a maze and find a path from a starting to an ending point in the maze. You are allowed to add additional functions in `utility.php` to help you complete this task. Only functions in `utility.php` would be marked and you are not allowed to change any other files given to you.

Part A - Complete `display_maze` (1 mark)

This method takes as a parameter `$maze` which is a 2-dimensional array. An example is shown below:

1	0	0	0	S
0	0	0	0	0
0	0	1	0	0
1	0	0	0	0
0	E	0	0	0

The example `$maze` should be rendered as follows when `maze.php` is opened in the browser:



As can be observed from the above example, there are four types of possible values in `$maze`: 0, which denotes an **empty** cell; 1, which denotes an **obstacle** cell; S, which denotes a **start** cell (only one such cell exists in `$maze`); and E, which denotes an **end** cell (only one such cell exists in `$maze`).

Your task to complete `display_maze` is to write a PHP code that can create an HTML table, similar to the one shown above, for any given input `$maze`. Note that `$maze` can be of different sizes. Use the image files given to you; for empty, obstacle, start, and end cells display `empty.png`, `obstacle.png`, `mario.png`, and `end.png`, respectively.

Part B - Complete `get_start_end_points` (2 marks)

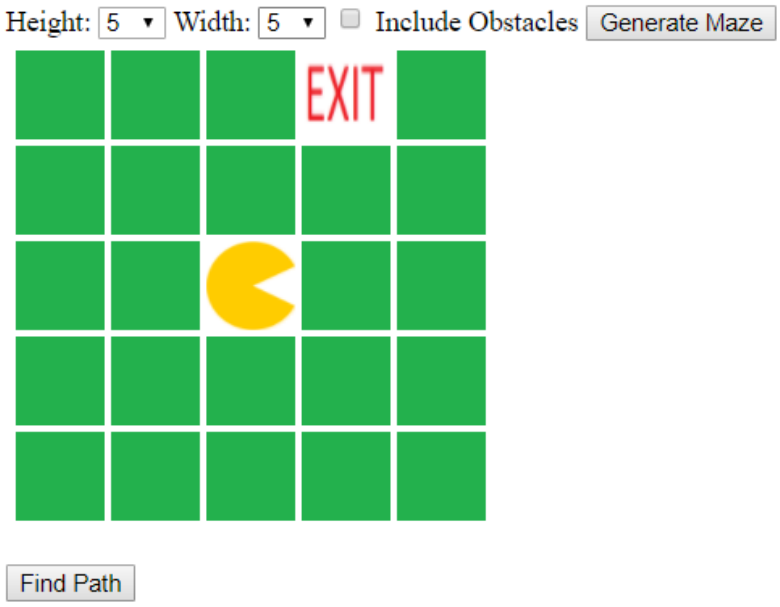
This function takes as a parameter `$maze` which is a 2-dimensional array described in Part A. Your task is to identify the position of the start and end cells in it. The function returns the positions of the start and end cells as an array of two `Point` objects. Assume that `$maze` always contains one start and one end cell.

Hint: Use `print_r` and `echo` to check whether your function behaves correctly

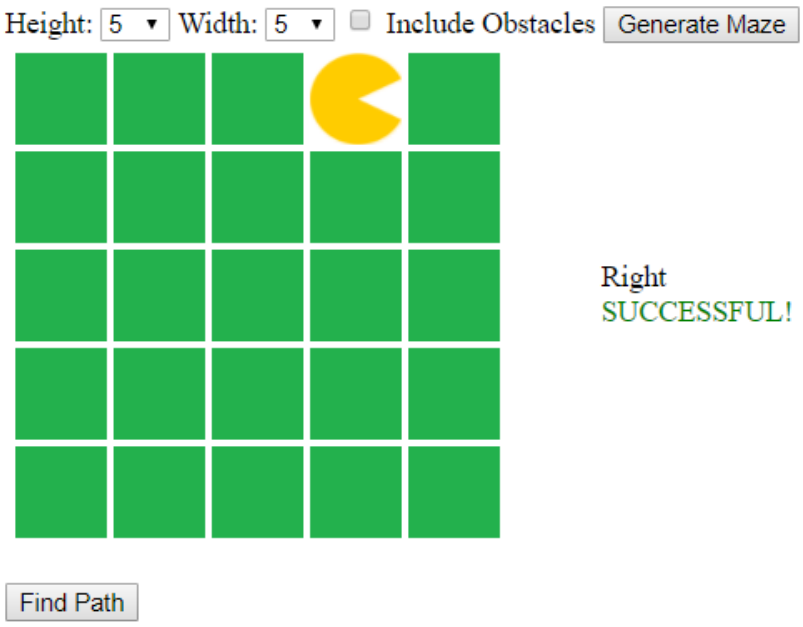
Part C - Complete `find_path_no_obstacle` (2 marks)

This function takes in three parameters: `$maze`, `$start`, and `$end`. `$maze` is the same 2-dimensional array described earlier, while `$start` and `$end` are two `Point` objects describing the locations of the start and end cells in `$maze`. Your task is to implement a code that can find a path from the start to the end cell. This path is an array of strings; each string specifies one move step. Four kinds of moves are possible: "L" (move one cell to the left), "R" (move one cell to the right), "U" (move one cell up) and "D" (move one cell down). This array needs to be returned by the method. You can assume that `$maze` does not contain any obstacle cell.

Example. Consider the following maze that does not contain an obstacle:



Clicking **Find Path** would animate the character to move from **cell (2,2) to cell (0,3),** one cell at a time. maze.php will call `find_path_no_obstacle` and use its returned array to perform the animation. The final part of the animation is shown below:

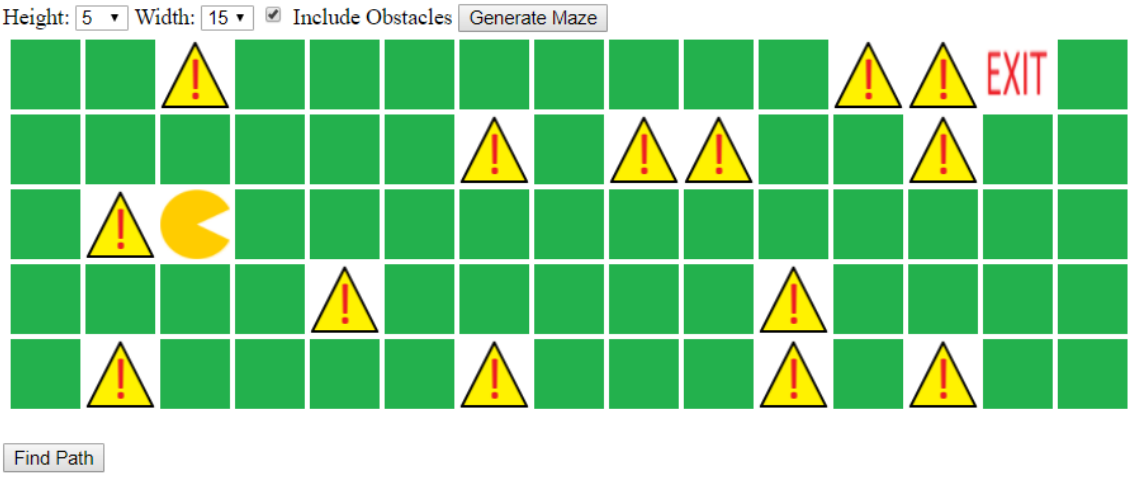


Part D - Complete `find_path` (2 marks)

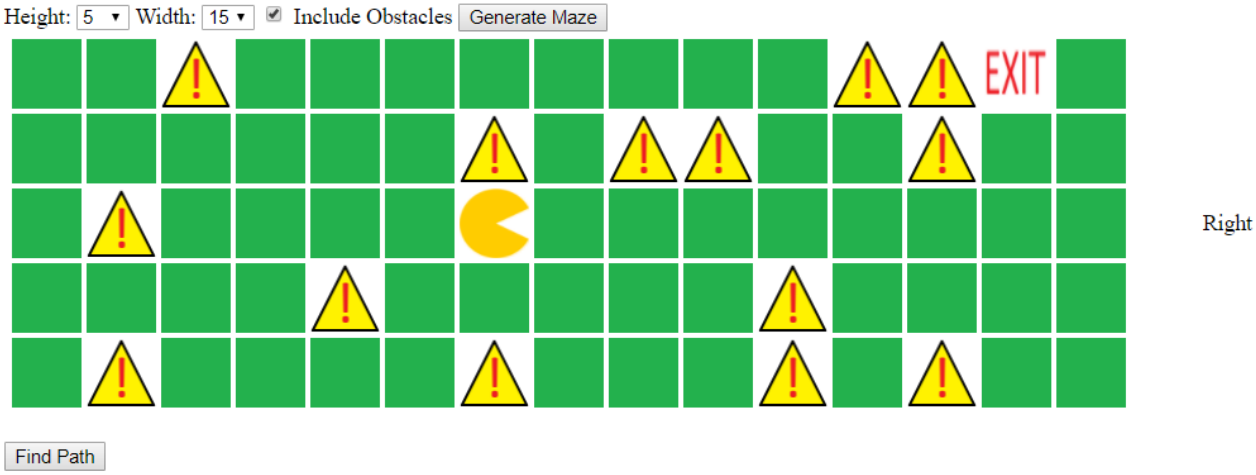
This function is a more general version of **`find_path_no_obstacle`**. It takes the same inputs (`$maze`, `$start`, `$end`) and also returns an array of strings describing a series of move steps from the start to the end cell. However, now `$maze` can possibly contain obstacle cells and your code needs to take them into consideration while identifying a path from `$start` to `$end`. If no path is possible (i.e., all paths from start to end cells are blocked by obstacle cells), this function returns an empty array.

Example.

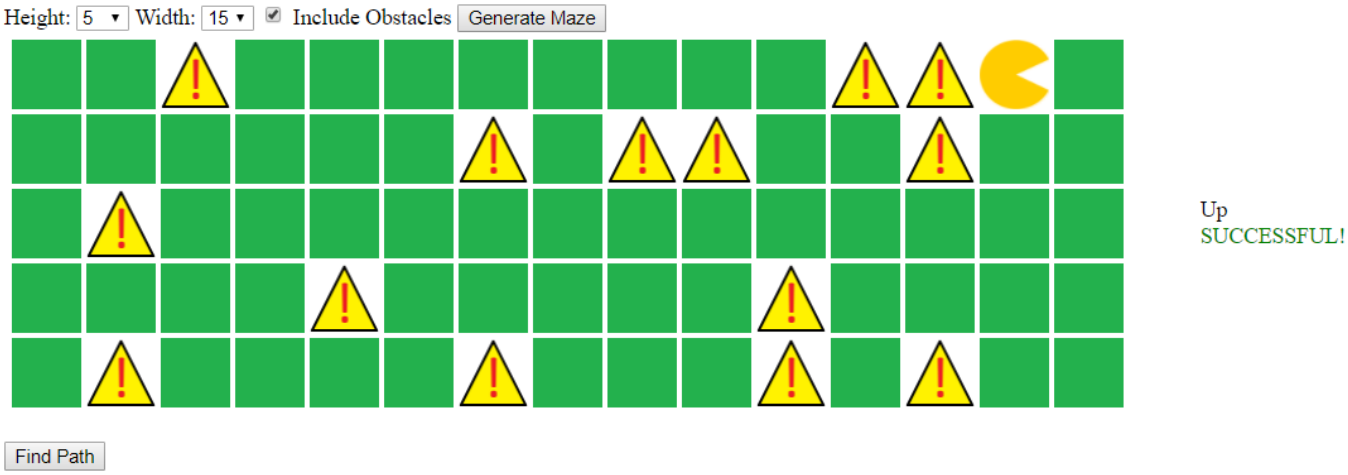
After **Generate Maze** is clicked:



After **Find Path** is clicked and halfway in the animation:



Final Step:



- END -