Find Your Hat

CodeAcademy Project Report

Amir Wan

28 May 2025

# Introduction

Project is don on VS Code.

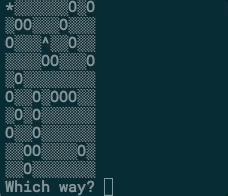
# Project Tasks

**Prerequisites**

**1.** In order to start this project, you should have completed lessons at least through Classes in our [Introduction to JavaScript](https://www.codecademy.com/learn/introduction-to-javascript) curriculum.

In this project, you’ll be building an interactive terminal game with JavaScript classes. Because the goal involves user input, make sure you read [our guide on working with user input in Node.js](https://www.codecademy.com/articles/getting-user-input-in-node-js). If you complete this project on our platform, you can test and run your code with the command node main.js in our terminal.

Watch this video to get a sense of what you’ll be building:



**Project Requirements**

**2.**Your project is centered on a Field class. This and the following tasks will describe how the class should function at a high level, and it will be up to you to figure out the implementation in code. As you go, test your code by creating instances of the class and calling its methods.

The Field constructor should take a two-dimensional array representing the “field” itself. A field consists of a grid containing “holes” (O) and one “hat” (^). We use a neutral background character (░) to indicate the rest of the field itself. The player will begin in the upper-left of the field, and the player’s path is represented by \*.

\*░O  
░O░  
░^░

Copy to Clipboard

Your class should take a single argument representing the field:

const myField = new Field([  
  ['\*', '░', 'O'],  
  ['░', 'O', '░'],  
  ['░', '^', '░'],  
]);

Copy to Clipboard

**3.** Give your Field class a .print() method that prints the current state of the field. You can choose to format this however you want, but it will be much easier to play the game if you print out a string representation of the board instead of the raw array.

**4.** Your game should be playable by users. In order to facilitate this, build out the following behavior:

* + When a user runs **main.js**, they should be prompted for input and be able to indicate which direction they’d like to “move”.
  + After entering an instruction, the user should see a printed result of their current field map with the tiles they have visited marked with \*. They should be prompted for their next move.

This should continue until the user either:

1. Wins by finding their hat.
2. Loses by landing on (and falling in) a hole.
3. Attempts to move “outside” the field.

When any of the above occur, let the user know and end the game.

**5.** Add a .generateField() method to your Field class. This doesn’t need to be tied to a particular instance, so make it a static method of the class itself.

This method should at least take arguments for height and width of the field, and it should return a randomized two-dimensional array representing the field with a hat and one or more holes. In our solution, we added a third percentage argument used to determine what percent of the field should be covered in holes.

**Project Solution and Extensions**

**6.** Great work! Visit [our forums](https://discuss.codecademy.com/t/find-your-hat-challenge-project-javascript/462380) to compare your project to our sample solution code. You can also learn how to host your own solution on GitHub so you can share it with other learners! Your solution might look different from ours, and that’s okay! There are multiple ways to solve these projects, and you’ll learn more by seeing others’ code.

**7.** If you’d like to extend your project past our requirements, here are a few ideas:

* + Have the character start on a random location that’s not the upper-left corner.
  + Create a “hard mode” where one or more holes are added after certain turns.
  + Improve your game’s graphics and interactivity in the terminal. There are many [helpful packages](https://github.com/cronvel/terminal-kit) to assist with this, and you can really get creative with how you approach terminal games.
  + Create a field validator to ensure that the field generated by Field.generateField() can actually be solved. This might be pretty difficult! You’ll essentially be creating a version of a [maze solver](https://en.wikipedia.org/wiki/Maze_solving_algorithm).