**CONNECT FOUR GAME (TREEHOUSE)**

**How Does It Work?**

Connect Four (also known as Captain's Mistress, Four Up, Plot Four, Find Four, Four in a Row, Four in a Line, Drop Four, and Gravitrips (in Soviet Union)) is a two-player connection game in which the players first choose a color and then take turns dropping one colored disc from the top into a seven-column, six-row vertically suspended grid. The pieces fall straight down, occupying the lowest available space within the column. The objective of the game is to be the first to form a horizontal, vertical, or diagonal line of four of one's own discs. Connect Four is a solved game. The first player can always win by playing the right moves.

**STAGE 1**

Instruction

Review Provided Code

### **Assignment:**

1. Download the project files [here](https://s3.amazonaws.com/treehouse-project-downloads/ObjectOrientedJavaScriptByExample.zip).
2. Look over the HTML and CSS provided.
3. Take note of the different directories.
4. If you have any questions, check the Treehouse Community for related discussions, and then post any unique questions there.

### **Course format**

Welcome to your first instruction step! You'll be receiving all of your instructions and guidelines in these text-based steps so you can easily reference them while you're coding. It's important that you read each of the instruction pages (including this one!) thoroughly. At the top of each instruction page I'll provide the assignment, with any relevant details, guidelines, notes, and resources to follow.

This course doesn't use workspaces. All of your starting project files are provided for you [here](https://s3.amazonaws.com/treehouse-project-downloads/ObjectOrientedJavaScriptByExample.zip). You're going to be creating many files as you work through the course project, so the provided code may seem a little empty. All you'll find right now are empty folders, an index.html file, and a style.css file.

### **What to do if you get stuck**

If you get stuck on a challenge or assignment, don't worry! Make an attempt, and then just head to the solution video. If you really want to try it on your own, asking questions in the Treehouse community can be a great start, but try not to get too hung up.

Instruction

Planning the Classes

### **Assignment:**

1. Consider the different game components discussed so far and think about how these components could be represented as objects (if you think they should be represented as objects).
2. Brainstorm or write out responses to the questions on this page.
3. Write down the classes you think this game should include.
4. Optionally, discuss your class choices in the Treehouse Community.

### **A Little Bit About App Architecture**

Now that you're familiar with the game of "Four in a Row" and you've had a chance to review the HTML and CSS of the game, it's time to start thinking about object design.

App architecture can be very complicated, even in simple games like "Four in a Row". There are countless approaches, design patterns, frameworks and tools you could utilize when architecting an app.

As you become more involved in the industry you'll find that the opinions on these things vary drastically. You'll also find that nothing is set in stone: In almost every circumstance developers will rework a solution or iterate on it several times before putting it in production. Even then, apps are continuously in cycles of refactoring and improvement.

What's important is not focusing on the "best" way to do something, but instead focusing on developing a codebase that is planned, logical, organized, documented, and tested.

That's why before you start coding "Four in a Row", you'll take the time to map out the different ways you could organize your objects and if you like, you'll discuss these considerations with other students in the Treehouse Community.

### **Object Design in "Four in a Row"**

JavaScript objects can be a way to help us model real-life objects, or they can be used to provide structure and organization to real-life concepts and ideas.

When we build our Four in a Row game, we'll be modeling all of its parts, pieces, and concepts with JavaScript objects. To do that, we first need to brainstorm the best way to map these parts and concepts to classes.

To help you identify the different classes you might use in the development of this game, consider the following questions:

1. What are the different components of a "Four in a Row" game? How do players interact with the game? Think about a physical version of the game, and see if that helps.
2. Can you break down the game components into smaller, or individual parts? Do some components serve others?
3. Where will you handle the game state and game logic: switching players, checking for winning moves, resetting the board, etc.?

Note: These brainstorming exercises are to practice figuring out "where to start". The goal is for you to gain experience thinking like a developer. While I hope that you take advantage of this practice time, please note that we'll be using a previously developed solution as we go through this course.

After completing the assignment, join me in the video that follows to see my solution and the classes we'll be using moving forward.

**Instruction**

# Creating Class Files

### **Assignment:**

Project file directory : Stage1-Step5

1. Inside the js directory, create the following files: Player.js, Token.js, Board.js, Space.js, and Game.js. Be sure to capitalize the names of the files, like you see here.
2. Inside each of the files, declare the associated class. Don't add any constructor methods, properties, or other methods to the class declarations yet.

**Note:** There won't be a solution video following this assignment. You'll have a chance to double check your class declarations against mine once we start the next stage.

### **Declaring Classes**

If you've forgotten how to declare a class, here are some resources and guidelines to help you.

1. Remember, the name of the class is always capitalized.
2. Class declarations don't include parenthesis, just open and closed curly brackets.

Check out the Treehouse [JavaScript Wiki Reference](https://github.com/treehouse/References/blob/master/javascript.md#class) for syntax examples and links to course videos where the syntax is taught.

**STAGE 2**

**Instruction**

# Practice Brainstorming Properties

### **Assignment:**

1. Consider the different potential properties, or attributes, of a Player object.
2. Brainstorm or write out responses to the questions on this page.
3. Write down what you've identified as potential properties for the Player class.
4. Discuss your choices in the Treehouse Community.

### **Player Property Brainstorming**

To complete this exercise there are a few things to keep in mind. Brainstorm responses to the following questions, and use your responses to guide your choices.

1. How will you differentiate between player objects? Will players have a name, or an ID? How would an ID be created or set?
2. How will a player's tokens be stored? How many tokens should each player have? How will you store or set that number?
3. How will each player's tokens be differentiated from the other player's, visually?
4. What are some ways to keep track of whose turn it is?

After you complete this practice exercise, join me in the next step to see the Player properties we'll be using as we build the game.

**Instruction**

# Token Properties Brainstorming

### **Assignment:**

1. Consider the different potential properties, or attributes, of a Token object.
2. Brainstorm or write out responses to the questions on this page.
3. Write down the properties you've identified for the Token class.
4. Discuss your choices in the Treehouse Community.

### **Token Property Planning**

To complete this exercise there are a few things to keep in mind. Brainstorm responses to the following questions, and use your responses to guide your choices.

1. How will you differentiate between token objects? How can you use the Player ID to create a token identifier?
2. How will you keep track of which player an individual token belongs to?
3. How will you know if a token has been played or not?

**Instruction**

# Build the createTokens() Method

### **Assignment:**

1. Inside the Player class, write a method called createTokens() using the guidelines and requirements below.
2. In the Player class constructor method, change the value of the tokens property from an empty array to a call to the createTokens() method.
3. When you call the createTokens() method pass in the integer 21, for the number of tokens to be created.

### **The "Why"**

The Player objects in our JavaScript interpretation of "Four in a Row" should be pretty reminiscent of a real-life player of Four in a Row.

One thing that players have to do in a game of Four in a Row is collect their tokens before the game starts, so they are ready to use.

In our JavaScript version, there are no physical tokens to collect. We have to write the code that both creates these tokens for our Players, and associates the tokens with the appropriate player. That's what the createTokens() method will do.

By creating the Token objects from within the Player class, we can make sure that each Player has their own set of properly identifiable Token objects.

### **Let's Talk About "Creator Methods"**

If you recall, when you wrote the constructor method for the tokens property inside the Player class, you set the value equal to an empty array. In this step, you're going to write a method that populates this array.

Throughout this course you will be writing what I will call "Creator Methods". These are not formal JavaScript method types. They are a way for you to routinely create several objects at once. Tucking this functionality into a method keeps your code organized and neat, and ensures that a caller could use the method without having to know the details of how new objects are created.

### **Requirements for the createTokens() method**

/\*\*

\* Creates token objects for player

\* @param {number} num - Number of token objects to be created

\* @returns {Array} An array of the newly created token objects

\*/

1. It should receive an argument called num that represents the number of tokens to be created.
2. It should return an array of newly created token objects.
3. A loop can be used to generate a new Token object on each iteration (up to num iterations). Be sure to pass in the appropriate arguments to the Token constructor method.

For a refresher on the correct syntax for instantiating a new object, check out the [Treehouse JavaScript Wiki Reference](https://github.com/treehouse/References/blob/master/javascript.md" \l "instance" \t "_blank).

**Instruction**

# Board and Space Class Constructor Methods

### **Assignment:**

Project file location: stage2-step8

1. Create a constructor method for the Board class according to the requirements and guidelines below.
2. Create a constructor method for the Space class according to the requirements and guidelines below.
3. Visit the Treehouse Community and post any questions or feedback you have for other students regarding the Board and Space class constructor methods and properties.

### **The "Why"**

The Board and Space classes each have their own constructor methods.

The Board class has the following properties (outlined further in the requirements section below): rows, columns, and spaces.

If you look at a physical version of a Four in a Row, it's easy to see that it's laid out in a grid, with a series of rows and columns. Every time a row intersects with a column there is a cut out space that could eventually be home to a token.

On the Board class, the rows and columns properties indicate the number of rows and columns that the board has, and the Spaces property is an array of all the spaces on the board. Each of these spaces has it's own properties and methods as well, which is why we created a Space class.

The Space class has the following properties (outlined further in the requirements section below): x, y, id, and token.

On the Space class, the x and y properties represent the column and row, respectively, of the space's location on the board. The id property gives us a way to identify the space when it's drawn on our HTML to be seen in browser.

The token property, initially set to null, is a way to associate a Token object with the space it's in dropped into.

### **Requirements and Guidelines for Board Class Constructor Method**

1. The constructor method should define the following properties: rows, columns, and spaces.
2. Set the value of the rows property to 6.
3. Set the value of the columns property to 7.
4. Set the value of the spaces property to an empty array.

**Note:** So you know, for the spaces property, we'll be mimicking the pattern we used for the Player object when we created tokens: the property will eventually be set to a creator method call that returns an array of spaces. We're using the empty array as a placeholder until we get to that point.

### **Requirements and Guidelines for Space Class Constructor Method**

1. The constructor method should define the following properties: x, y, id, and token.
2. Set the value of the x and y properties to the x and y values passed in as arguments to the constructor method.
3. Set the id property to template literal space-${x}-${y}.
4. Set the token property to null.

**Instruction**

# Build the createSpaces() Method

### **Assignment:**

1. Write a method called createSpaces() inside the Board class, below the constructor method using the requirements and guidelines below.
2. In the Board class constructor method, change the value of the spaces property from an empty array to a call to the createSpaces() method.
3. Visit the Treehouse Community to discuss your approach and ask questions if you need to.

### **The "Why"**

Now that you've written the constructor method for the Space class, we can start building Space objects and adding them to the Board class's spaces property (which so far is just an empty array). You can do this in another creator method, just like you did when you created Token objects in the Player class.

The Space objects all belong to the Board, and are stored inside a property on the Board object. That's why they are created inside the Board class.

### **Requirements and Guidelines for the createSpaces() Method**

/\*\*

\* Generates 2D array of spaces.

\* @return {Array} An array of space objects

\*/

1. It should receive no arguments.
2. It should return a two-dimensional array of newly created Space objects.
3. A nested for-loop can be used to iterate through the rows and columns to generate a new Space object for every location on the board. Be sure to pass the appropriate arguments to the Token constructor method.

For a refresher on the correct syntax for instantiating a new object, check out the [Treehouse JavaScript Wiki Reference](https://github.com/treehouse/References/blob/master/javascript.md" \l "instance?" \t "_blank).

**Instruction**

# Game Class Constructor Method

### **Assignment:**

1. Create the constructor method for the Game class using the requirements and guidelines below.
2. Inside the Game class, write a creator method called createPlayers() using the requirements and guidelines below.

### **The "Why"**

The final constructor method that needs to be written is the Game class constructor method.

The Game class has three properties: board, players, and ready.

The players property is going to function just like the tokens property on the Player class, and the spaces property on the Board class. We'll use another creator method, createPlayers(), to instantiate the Player objects and attach them to the Game object by storing them in the players property.

This means that when a new Game object is created, so are the new Player objects. And, as you know, when the Player objects are created, so are all of their respective Token objects!

Similarly, the Game's board property is set to a new instance of the Board class. So when the Game object is created, a new Board object is created as well. And, of course, when the Board object is created, so are all the Space objects! Since there is only one board per game, we don't need a creator method. We can create the Board instance right from the board property initialization.

The ready property represents the game's ready state. In other words, whether or not the game can or should be interacted with. When we first initialize the game, the ready state will be false. Throughout the game, this property will be changed to true and then back again, and when it's false, the user will be prevented from interacting with the game.

### **Requirements and guidelines for the Game class constructor method**

1. The constructor method should define the following properties: board, players, and ready.
2. The board property will be set to a new Board object.
3. The players property will hold an array of Player objects. This will be done using the same pattern we used to create Token objects inside the Player class, and create Spaces inside the Board class.
4. The ready property will be set to false.

### **Requirements and guidelines for the createPlayers() method**

/\*\*

\* Creates two player objects

\* @return {Array} An array of two Player objects.

\*/

1. It should receive no arguments.
2. It should return an array of two newly created Player objects

Note: Since the game only has two players, a loop is not needed to create them. Be sure to pass in the appropriate arguments to the Player constructor method. For details on what values to pass, see the next section.

### **Player Object Property Values**

#### **Player 1**

* The name parameter should receive a string value of Player 1 (or name of your choice)
* The color parameter should receive a string value of #e15258
* The id parameter should receive an integer value of 1
* The active parameter should receive a boolean value of true

#### **Player 2**

* The name parameter should receive a string value of Player 2 (or name of your choice)
* The color parameter should receive a string value of #e59a13
* The id parameter should receive an integer value of 2
* Because the active parameter in the Player constructor method has a default value of false, no argument for the value of active needs to be passed.