Build a game of battling alien spaceships using Javascript.

Earth has been attacked by a horde of aliens! You are the captain of the USS HelloWorld, on a mission to destroy every last alien ship.

Battle the aliens as you try to destroy them with your lasers.

There are six alien ships. The aliens' weakness is that they are too logical and attack one at a time: they will wait to see the outcome of a battle before deploying another alien ship. Your strength is that you have the initiative and get to attack first. However, you do not have targeting lasers and can only attack the aliens in order. After you have destroyed a ship, you have the option to make a hasty retreat.

A game round would look like this:

- You attack the first alien ship
- If the ship survives, it attacks you
- If you survive, you attack the ship again
- If it survives, it attacks you again … etc
- If you destroy the ship, you have the option to **attack** the next ship or to **retreat**
- If you retreat, the game is over, perhaps leaving the game open for further developments or options
- You win the game if you destroy all of the aliens
- You lose the game if you are destroyed

Ship Properties

- **hull** is the same as hitpoints. If hull reaches `0` or less, the ship is destroyed
- **firepower** is the amount of damage done to the **hull** of the target with a successful hit
- **accuracy** is the chance between 0 and 1 that the ship will hit its target

Your spaceship, the USS HelloWorld should have the following properties:

- **hull** - `20`
- **firepower** - `5`
- **accuracy** - `.7`

The alien ships should each have the following *ranged* properties determined randomly:

- hull - between `3` and `6`
- firepower - between `2` and `4`
- accuracy - between `.6` and `.8`

You could be battling six alien ships each with unique values.

Example use of **accuracy** to determine a hit:

```
if (Math.random() < alien[0].accuracy) {
    console.log('You have been hit!');
}
```

 Where to begin?

- Read over the specifications. Make sure you understand them. If you do not understand them, try to clarify them for yourself.
- Plan the game. This is an act of simplification.

From these programming principles

Links to an external site.

1. Use **pseudo code** to get a sketch of your game first.
2. Avoid Creating a **YAGNI** (You aren't going to need it) - You should not try to add functionality until you need it.
3. **Do the simplest thing that could possibly work.**

Often, beginning something is an act of **creative inspiration** to find the **simplest possible case**. The first step is not necessarily a matter of logical deduction. Once you have a few 'clues' you can follow the trail of crumbs to a logical conclusion.

 Actors and then actions

A good rule of thumb is start with the **actors** and then the **actions**. What actors do we need? In this case, we need our spaceship and the alien spaceships. An action these ships can take is to attack something. Perhaps a ship object (an actor) could therefore have an **attack** method (an action).

A repeating action in the game is that these ships attack each other until one of them has been destroyed. This might necessitate a loop or multiple loops.

👾 Start simpler than the instructions suggest

Keep these five things in mind when planning and coding your game:

1. Begin even simpler than the specifications suggest. In this case, perhaps just start with one alien ship instead of many alien ships, and get the code for one ship working first.
2. Root out any 'gotchas' that you really ought to foresee. In this case, will we really want nested loops -- one for a battle, one for iterating over aliens)? How will we exit one loop and then exit the parent loop? Perhaps keeping it to one loop somehow will help us avoid unnecessary difficulties.
3. When coding, form a solid and testable foundation before building upon it with more functionality. In this case, is there a bug where an alien can attack *after* it has been destroyed? Better fix that bug before increasing the complexity of the code.
4. When you have a piece of functionality tested and working, **commit it**. Try not to commit broken code. Unsure of when to commit? **Commit when something works**. You want to save working code.

👾 Code quality and code sharing

From the beginning, you will be writing your code **for other developers.**

Having to read and understand another developer's code is common practice. Get used to it now! In the 'real world', you will be in a position where you inherit someone else's code-base and are told to 'fix it' or to add a feature to the code.

What does this mean for your coding practices?

- Use proper indentation! On the job, your code immediately fails a code review if indentation is not perfect.
- Use semantic variable and function names, and use a verb for your function/method names.
- What your code **does** should be self-evident.
- If a piece of code is hard to read have a comment above those few lines explaining what they do.

Your code should be as coherent to another developer as possible.

🚀 Bonuses

- The aliens send a random number of ships to attack Earth. Think of a reasonable range and implement it.
- Scientists have developed a super targeting computer for your lasers. You now are asked which of the aliens you would like to hit with your lasers.
- Scientists have improved your ship's shields. They don't work that consistently, and only improve your hit points by a random number each time
- Scientists have put missiles on your ship. You only have a limited number of them, but they do a lot of damage. You

can say before each battle if you want to use one of your missles.

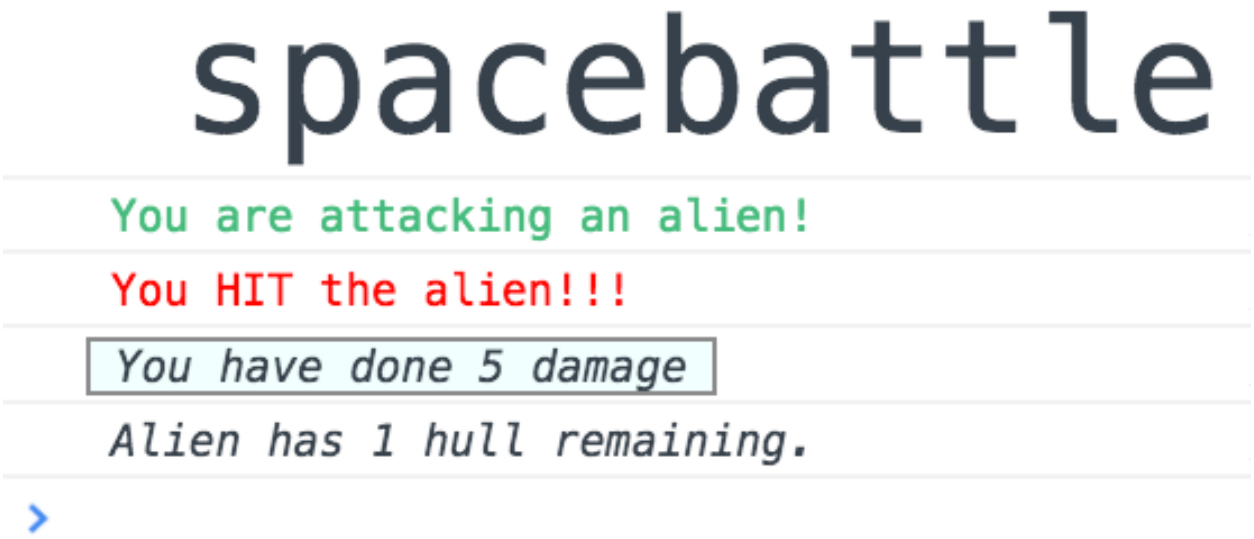
- The aliens have gained emotions and now can attack more than one at a time.
- Evil alien scientists have created an alien mega-ship. This mega-ship contains a number of "weapon pods" that each have their own individual hit points. These "weapon-pods" (objects) must all be destroyed before you can begin doing damage to the main ship, which also has its own hit points.

🚀 Bonus Bonuses

- When the game is over, prompt the user if they would like to play again, and make it so the user can play again with all the values set back to default.
- So far the entire game is just one battle, with many aliens. Implement a game that consists of multiple battles where enemies respawn for a new battle at the end of the old battle. Keep track of points for the number of wins the player has.
- After every battle you are asked if you want to return to base and recharge your shields.
- Make the players and enemies stronger after each battle
- Distribute medals and power ups to the player depending on points

Extra fun: style the console

Output in Chrome console:



You can use CSS in your Chrome console messages. Above is a simple example where messages are easier to differentiate.

Formula, use `%c` in the first argument to console log, and provide CSS to the second argument:

```
console.log('%c spacebattle', 'font-size: 40px');
```

Use multiple CSS properties:

```
console.log('%c You have done ' + player.firepower + ' damage ', 'font-style: italic;
background: azure; border: 1px solid grey;');
```