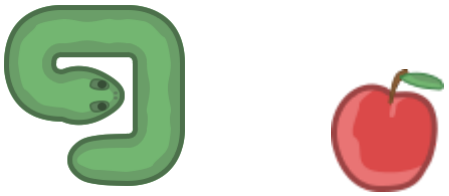
Code a Snake Game in JavaScript

Beginner Level • 13-16 y/o • 1h30

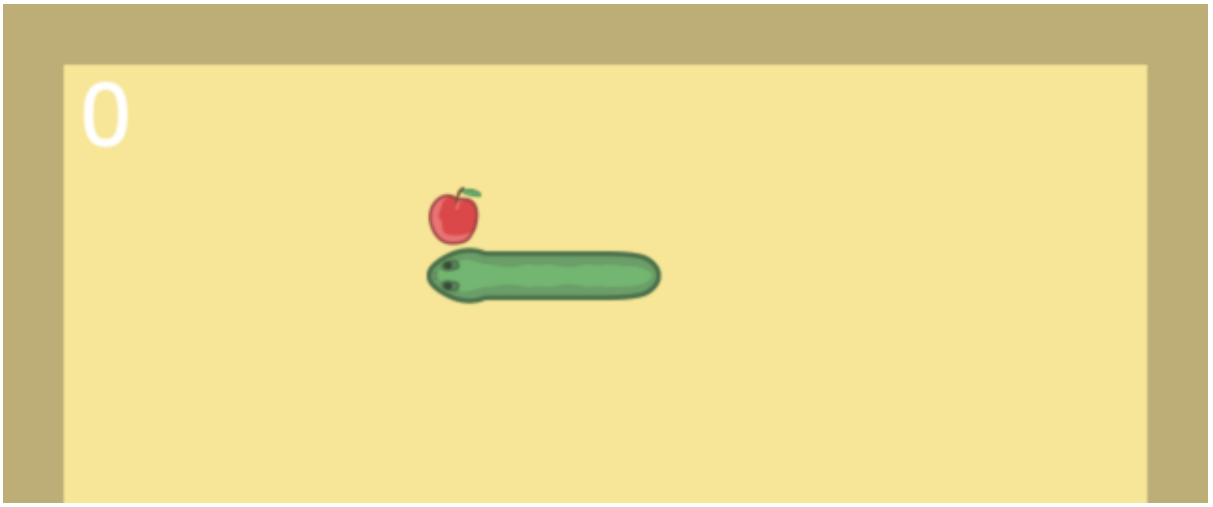
1. Presentation of the game



Snake is a video game genre where the player controls a snake that must eat apples to grow without eating itself. **Snake**'s concept has its origins in arcade games dating from the 1970s, which Nokia popularized as it was offered on all of the brand's phones.

Some parts of **Snake**'s code have been lost . . . It's up to you to complete the game code to get it working again!

In this exercise, you will learn how to program loops, conditions in **JavaScript**, and a web game's overall functioning.



2. Tools & Resources

2.1 Resources

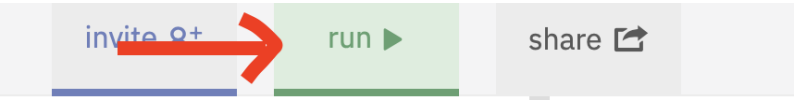
To get started, go to https://repl.it/@EmmaEpitech/Snake.

The screen is split into several parts: the file part, on the left: this is where you will code the game. And the rendered part, on the right: this is where you will see your progress and can test your game.

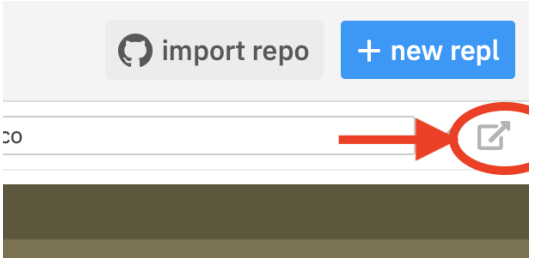
2.2 Getting started with the platform

On the left side of the screen, you can see a list of files.
HTML files, images, and **JavaScript** files.
Today, you will only need to edit the "**game.js**" file. However, you can go and see what happens in the other files if you wish.

To run your code, press the green arrow "**RUN**" at the top of the screen.



If your screen is too small to display the entire game, you can open it in a new tab by clicking on the following button:



2.3 Game.js

Several functions are present in the **game.js** file. Your role will be to complete them.

```
function gameLoop(game, nextBlock)
```

The **gameLoop** function is called on each turn of the game loop as soon as the **Snake** advances. This is the **game loop**.

```
function keyPressed(keyCode, snake)
```

The **keyPressed** function will allow you to perform actions on the **Snake** when a key on the keyboard is pressed.

3. Discovery of the game

3.1 Move Snake

 If you launch the game, you can see that **Snake** cannot move or change direction!

The first step will therefore be to allow **Snake** to move.

To do this, we will add the following instruction at the start of the **gameLoop** function:

```
game.snake.move();
```

Test your code by clicking on “**RUN**”!

You’ve probably noticed that you can’t get **Snake** to change direction. **Do not panic!** We’ll code this in the next two steps.

3.2 Detect keyboard keys

 Now that **Snake** is moving, you can see that he cannot change direction!

Therefore, the second step will be to detect when the user presses one of the arrows on the keyboard. This will help get **Snake** to change direction in the next step.

For this, we will complete the **keyPressed** function.
The **keyCode** variable allows us to know which key was pressed, according to the following correspondence table:

◂	▲	▼	▸
37	38	40	39

 We have 4 keys to detect. We could use **if** conditions, but we'll see a new concept today: the **switch**. This will make the code easier to read.

The **switch** allows you to compare the same variable with several different values.
It is written as in the example below:

```
switch (variable) {
   case valeur1:

      break;
}
```

Add a switch in the **keyPressed** function, and replace **variable** with **keyCode**.

```
switch (keyCode) {
```

Replaces the following variable **value1** in the switch, then adds the other cases (**case**).

```
   case 37:
```

 There are **4** values to test, so 4 **case** to add. Look in the table above.

To see which keys have been pressed, add the following line between each line **case X**: and **break**.

```
   case valeur1:
      console.log("Direction");
      break;
```

3.3 Changing direction

Now that we can detect which key is pressed, we just need to update the **Snake**'s direction.

This line allows you to change the direction of **Snake**:

```
snake.direction = 2;
```

You must assign the value corresponding to the direction in which the **Snake** must go according to the following table:

Up	Right	Down	Left
0	1	2	3

For each key pressed, it is therefore necessary to update the direction of **Snake**.

Launch the game to test your code!

3.4 Eat apples


In Snake, to earn points, you have to eat apples that appear on the screen.

For that, it is necessary to detect the collisions between **Snake** and the apples. We'll do this in the game loop (**gameLoop** function).

In the game, there are several types of blocks, according to the following table:

(empty)	Wall	Apple 🍎
0	1	2

The following line detects whether the next block is an apple. Add it after the **gameLoop** function:

```
if (nextBlock == 2) {

}
```

If the **Snake** collides with an apple, remove the apple from the screen.

We can do that with the following line:

```
game.removeBlock(nextBlock);
```

Relaunch the game to see what happens!

In **Snake**, when an apple is eaten, you earn a point, and **Snake** grows.
To update your score, add the following line:

```
game.score = game.score + 1;
```

Then we will be able to make Snake grow with the following function:

```
game.snake.grow();
```

3.5 Add an apple

Once **Snake** grows up after eating an apple, it is necessary to add a new apple to the game.

To do this, call the following function after the line added in the previous step:

```
game.addApple();
```

*Press **RUN** and try to earn as many points as possible!*

Good luck to you!

4. Bonus

If you have completed all the previous steps, first of all **Well Done!**

We have prepared some bonus steps you can do to improve the game or make it more difficult.

4.1 Snake grows faster depending on the number of points

To make **Snake** grow faster depending on the number of points, we will call the **grow** function several times.

Every 3 points, **Snake** will grow faster. For this, we will check by how much the number of points is a multiple of 3.

```
var repetition = game.score / 3;
```

Then we will make a loop according to this number:

```
for (i = 0; i <= repetition; i += 1) {

}
```

The last step is to move this line inside the loop!

```
game.snake.grow();
```

Launch the game and try to reach 10 points!

4.2 Collision with oneself

You may have noticed that you lose immediately if you go to the right and press the left arrow (since you collide with yourself).

In the real game of **Snake**, this is not possible.

Back to the **keyPressed** function, where we'll change that.

Before each line of change of direction, we will add a condition to remove this collision. The goal will be to only change direction **if you do not go in the opposite direction.**

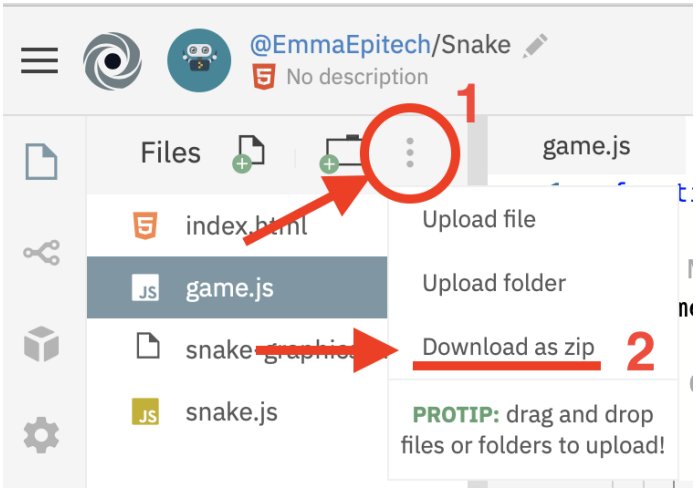
For example, if the new direction is 2 (down), the old direction cannot be 0 (up).

```
if (snake.direction != 0) {
    snake.direction = 2;
}
```

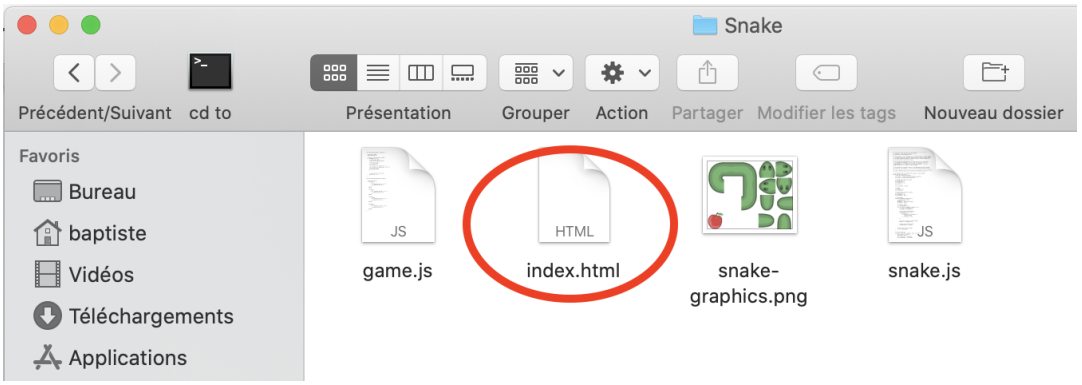
So you have 4 conditions like this to add.

5. Save your Snake

You can save your game on your computer to edit or play it later.
To do this, click on "**Download as zip**".



To launch the game on your computer, open the zip archive, then double-click on the **index.html** file.



6. Few useful links

To learn JavaScript:

➔ https://www.w3schools.com/js

To redo the exercise:

➔ https://repl.it/@EmmaEpitech/Snake

To see our other exercises:

➔ https://repl.it/@EmmaEpitech

For more information on our activities:

➔ https://www.e-mma.org