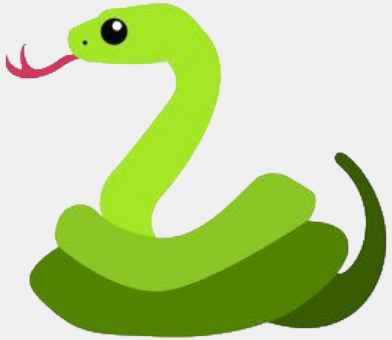
Announcements

- Theme clarification 💌🍓
- Fill out the feedback form for yesterday's lecture! weblab.is/feedback
- Mini-OH right after lecture! Come ask us questions 👉👈
- Send random thoughts to abby and tony! 🍪🥛 weblab.is/milkandcookies
- weblab.is/home 🔗 has all the links + info you need! And if there's anything missing or confusing, let us know via milk and cookies ^
- Milestone 0 (team creation + 10 ideas) due tomorrow (Wednesday) night
- Find teams!!

W1: Javascript

Mark Tabor and Enrique Casillas

Agenda: Make Something With JS



Demo

Things We Need

1. **Game setup**
2. **Snake**
3. **Respond to inputs**
4. **Food**
5. **Add Game Over**

Setup and Starter Code

Let's get started!

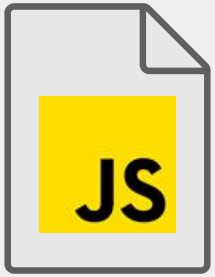
`cd` into your `catbook-react` folder

Run **`git fetch`**

Run **`git reset --hard`**

Run **`git checkout w1-starter`**

Starter Code



`game.js`

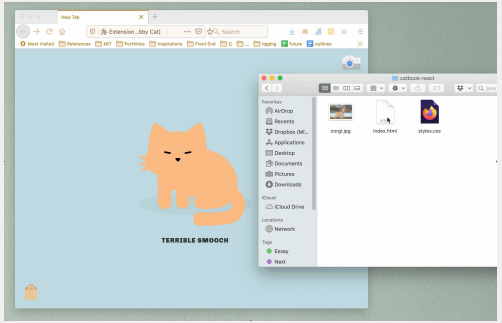
`index.html`

`style.css`

***No need to edit!**

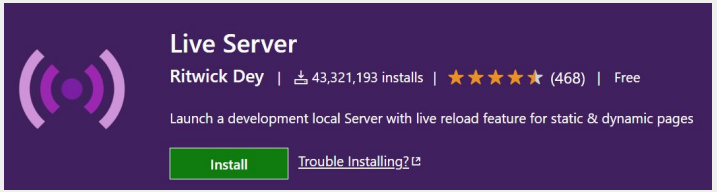
Running the game

- Open **`index.html`** in finder/file explorer or drag the file into your browser

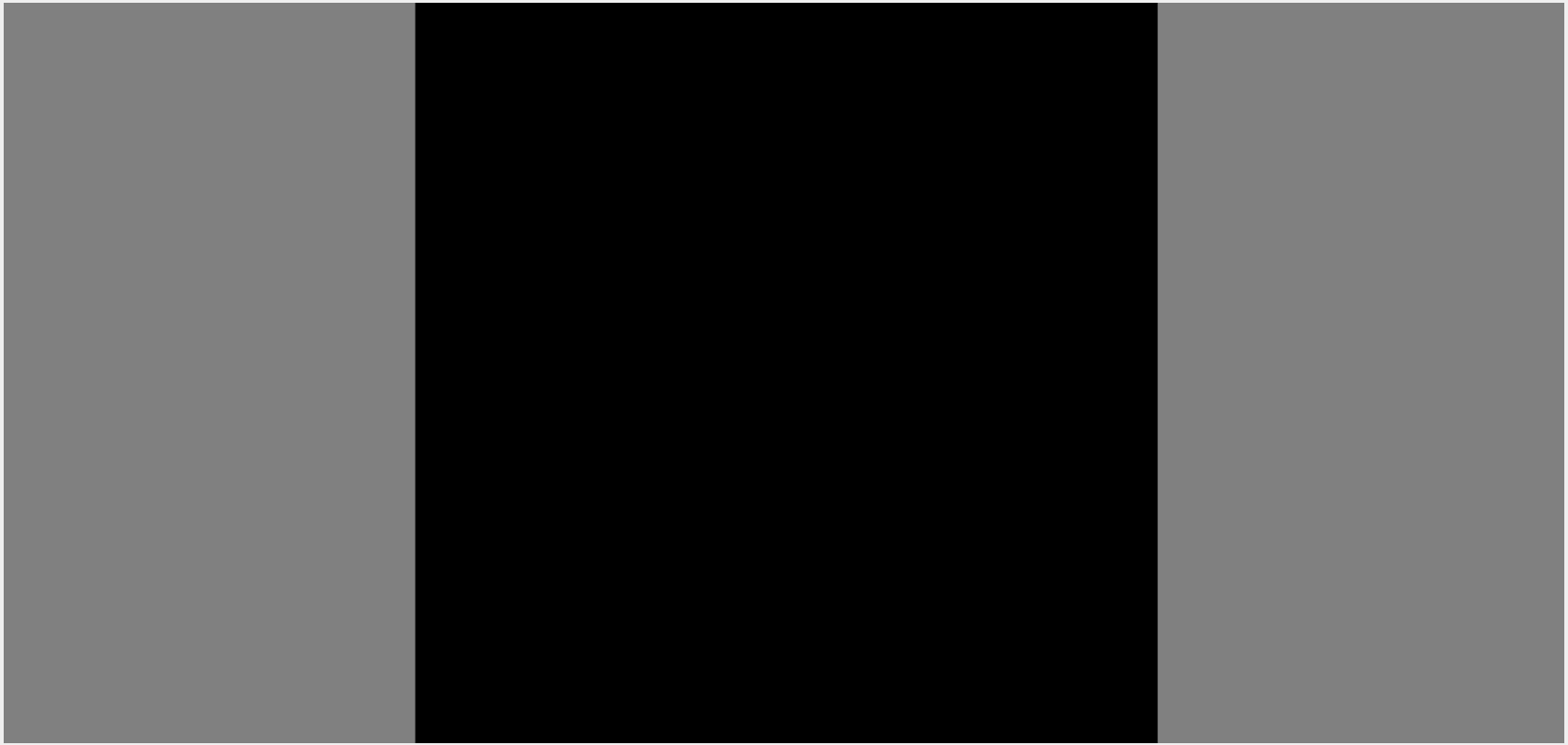


-or-

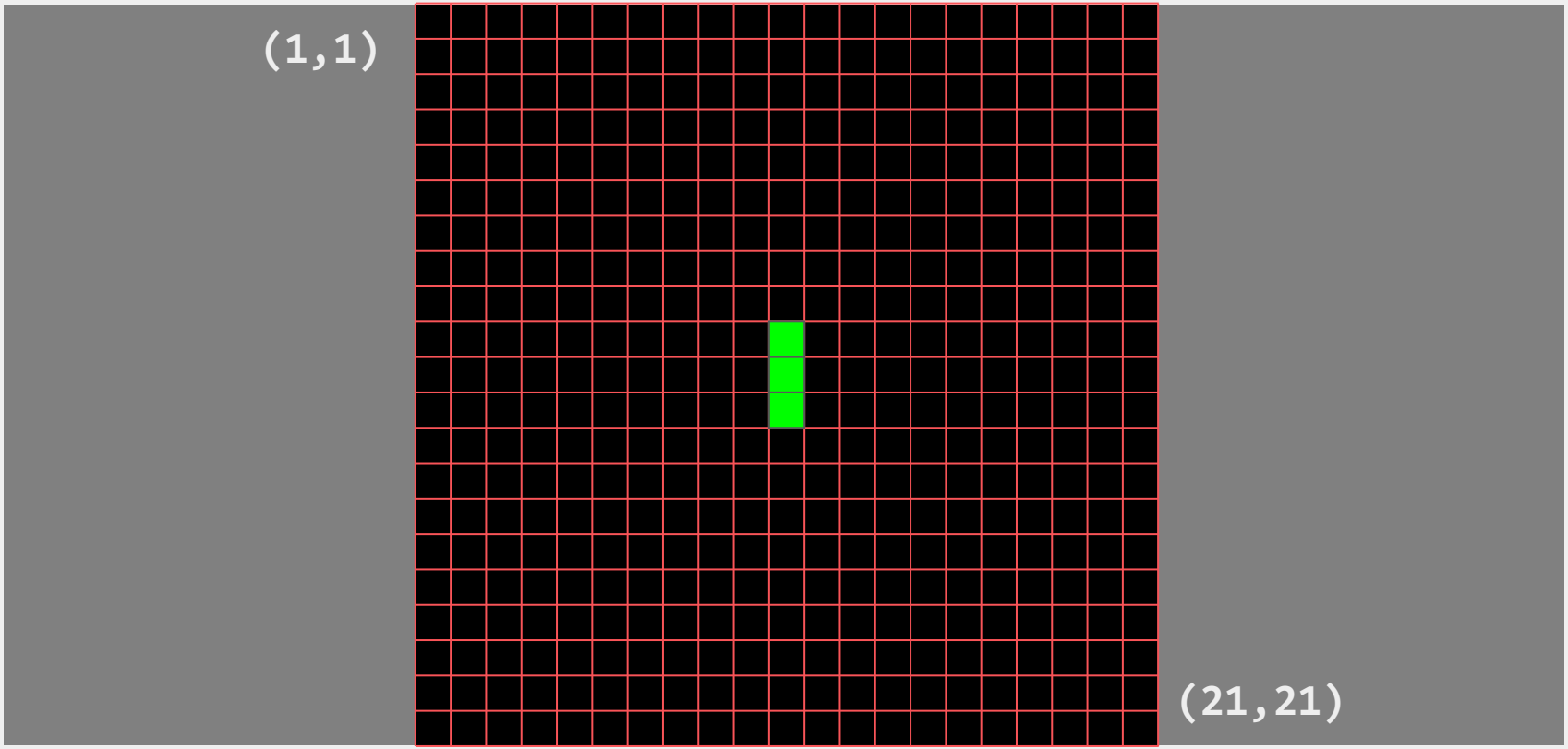
- Install the Live Server extension in VSCode and start it after navigating to **`index.html`**



Starter Game Grid



Starter Game Grid



Step 0:

🔧 Game Setup 🔧

```
git checkout w1-starter
```

Step 0: Game setup

`game.js`

`index.html`

```
git checkout w1-starter
```

Tasks:

1. Create the game loop that runs 5 times per second

Hint: Look at the **`setInterval`** MDN documentation

2. Connect `game.js` to `index.html`

Hint: Syntax for adding a Javascript file to HTML is `<script src="scriptName.js">`

```
git checkout w1-starter
```

Step 0: Game setup

Tasks:

1. Create the game loop that runs 5 times per second

game.js

```
const SNAKE_SPEED = 5;

...

setInterval(main, 1000/SNAKE_SPEED);

const update = () => {
    console.log("Updating");
}
```

2. Connect `game.js` to `index.html`

```
git checkout w1-starter
```

Step 0: Game setup

Tasks:

1. Create the game loop that runs 5 times per second

2. Connect `game.js` to `index.html`

game.js

```
const SNAKE_SPEED = 5;

...

setInterval(main, 1000/SNAKE_SPEED);

const update = () => {
    console.log("Updating");
}
```

index.html

```
<script src="game.js" defer></script>
```

Step 1:

🐍 Create the Snake 🐍

```
git reset --hard
git checkout w1-step1
```

Step 1: Snake

snake.js

game.js

```
git reset --hard
git checkout w1-step1
```

Tasks:

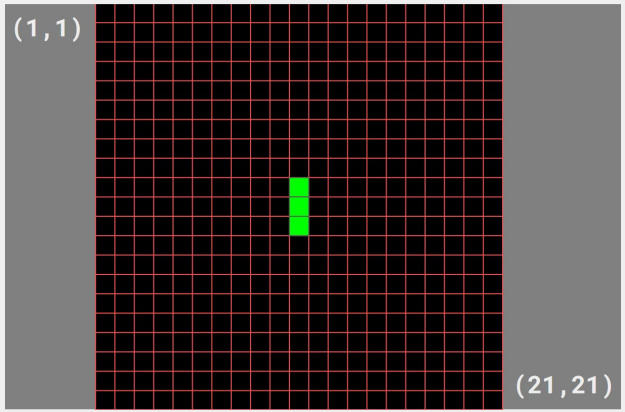
1. Create the snake body
2. Create a function to move the snake
3. Update the snake in the game loop
4. Connect snake file to HTML

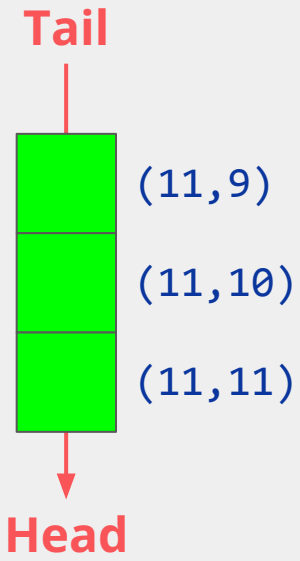
```
git checkout w1-step1
```

Step 1: Snake

1. Create the snake in `snake.js`

Hint: Recall the grid and consider this diagram →





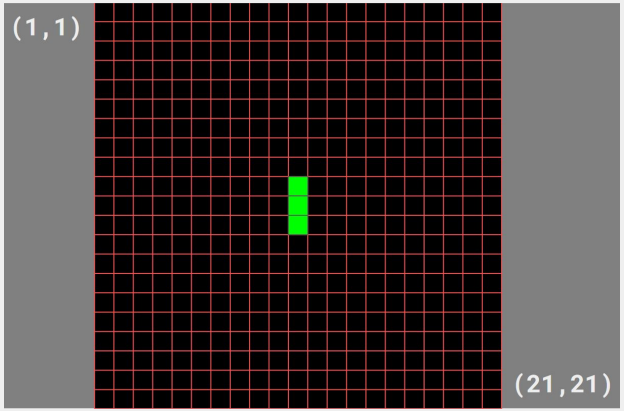
How should we represent the snake in JavaScript?

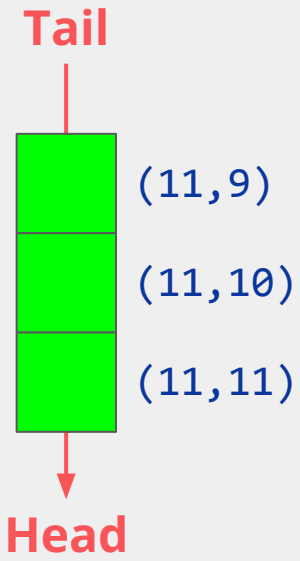
```
git checkout w1-step1
```

Step 1: Snake

1. Create the snake in `snake.js`

Hint: Recall the grid and consider this diagram →





snake.js

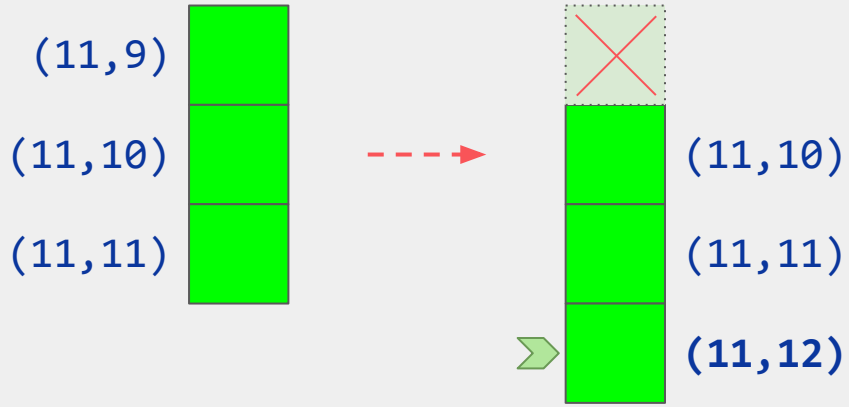
```
const snakeBody = [
  { x: 11, y: 11 },
  { x: 11, y: 10 },
  { x: 11, y: 9 },
];
```

```
git checkout w1-step1
```

Step 1: Snake

2. Create a function to update (move) the snake

Hint:



Hint: Recall the function syntax from yesterday:

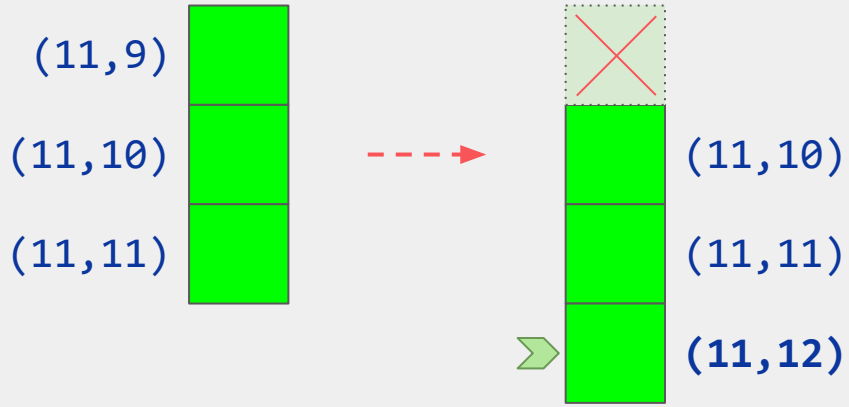
```
const functionName = () => {
   ...
}
```

`git checkout w1-step1`

Step 1: Snake

2. Create a function to update (move) the snake

Hint:



snake.js

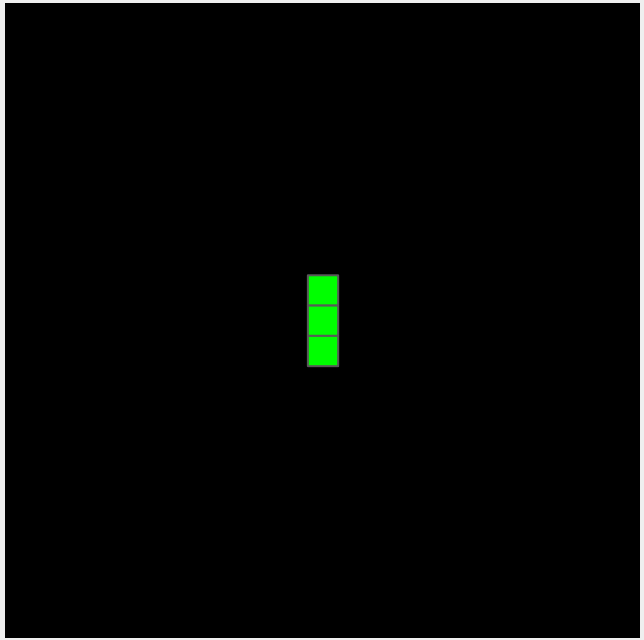
```
const updateSnake = () => {
  // Remove tail segment
  snakeBody.pop();

  // Add new head segment
  const newHead = { ...snakeBody[0] };

  newHead.x += 0;
  newHead.y += 1;

  snakeBody.unshift(newHead);
};
```

```
git checkout w1-step1
```

Step 1: Snake

3. Call the snake updater function in the game loop in `game.js`

4. Connect the snake script to the HTML file



```
git checkout w1-step1
```

Step 1: Snake

3. Call the snake updater function in the game loop in `game.js`

game.js

```
const update = () => {
  console.log('Updating');
  updateSnake();
};
```

4. Connect the snake script to the HTML file

```
git checkout w1-step1
```

Step 1: Snake

3. Call the snake updater function in the game loop in `game.js`

game.js

```
const update = () => {
  console.log('Updating');
  updateSnake();
};
```

4. Connect the snake script to the HTML file

index.html

```
<script src="snake.js" defer></script>
```

Step 2:

⌨️ Respond to Inputs ⌨️

```
git reset --hard
git checkout w1-step2
```

Step 2: Inputs

input.js

game.js

```
git reset --hard
git checkout w1-step2
```

Tasks:

1. Add a keyboard input event listener
2. Move snake based on inputs
3. Connect input file to HTML
4. Make sure snake can’t backtrack

```
git checkout w1-step2
```

Step 2: Respond to inputs

1. Create keyboard input event listeners for controlling the snake in **`input.js`**

KeyboardEvent: key property

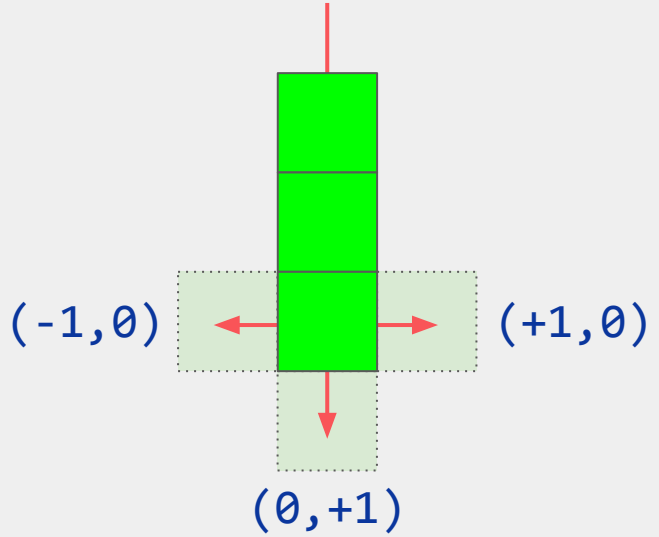
The `KeyboardEvent` interface's `key` read-only property returns the value of the key pressed by the user, taking into consideration the state of modifier keys such as `Shift` as well as the keyboard locale and layout.

Value

A string.

mdn web docs_

Hint: Consider where the new snake head will be given **arrow key** up/down/left/right inputs



```
git checkout w1-step2
```

Step 2: Respond to inputs

1. Create keyboard input event listeners for controlling the snake in **`input.js`**

KeyboardEvent: key property

The `KeyboardEvent` interface's `key` read-only property returns the value of the key pressed by the user, taking into consideration the state of modifier keys such as `Shift` as well as the keyboard locale and layout.

Value

A string.

mdn web docs_

`input.js`

```
let inputDirection = { x: 0, y: 1 };

window.addEventListener('keydown', (event) => {
  if (event.key === 'ArrowUp') {
    inputDirection = { x: 0, y: -1 };
  } else if (event.key === 'ArrowDown') {
    inputDirection = { x: 0, y: 1 };
  } else if (event.key === 'ArrowRight') {
    inputDirection = { x: 1, y: 0 };
  } else if (event.key === 'ArrowLeft') {
    inputDirection = { x: -1, y: 0 };
  }
});

const getInputDirection = () => {
  return inputDirection;
};
```

```
git checkout w1-step2
```

Step 2: Respond to inputs

2. Change the snake's position based on the user inputs in **`snake.js`**

3. Connect the input script to the HTML file

```
git checkout w1-step2
```

Step 2: Respond to inputs

2. Change the snake's position based on the user inputs in **`snake.js`**

3. Connect the input script to the HTML file

snake.js

```
const updateSnake = () => {
  ...

  const newHead = { ...snakeBody[0] };
  const snakeDirection = getInputDirection();

  newHead.x += snakeDirection.x;
  newHead.y += snakeDirection.y;

  snakeBody.unshift(newHead);
}
```

```
git checkout w1-step2
```

Step 2: Respond to inputs

2. Change the snake's position based on the user inputs in **`snake.js`**

snake.js

```
const updateSnake = () => {
  ...

  const newHead = { ...snakeBody[0] };
  const snakeDirection = getInputDirection();

  newHead.x += snakeDirection.x;
  newHead.y += snakeDirection.y;

  snakeBody.unshift(newHead);
}
```

3. Connect the input script to the HTML file

index.html

```
<script src="input.js" defer></script>
```

```
git checkout w1-step2
```

Step 2: Respond to inputs

4. Ensure user can't turn back on itself (snake can't crash into itself)

Modify **`input.js`**

```
git checkout w1-step2
```

Step 2: Respond to inputs

4. Ensure user can't turn back on itself (snake can't crash into itself)

Modify **`input.js`**

input.js

```
let inputDirection = { x: 0, y: 1 };

window.addEventListener('keydown', (event) => {
  if (event.key === 'ArrowUp' && inputDirection.x !== 0) {
    inputDirection = { x: 0, y: -1 };
  } else if (event.key === 'ArrowDown' && inputDirection.x !== 0) {
    inputDirection = { x: 0, y: 1 };
  } else if (event.key === 'ArrowRight' && inputDirection.y !== 0) {
    inputDirection = { x: 1, y: 0 };
  } else if (event.key === 'ArrowLeft' && inputDirection.y !== 0) {
    inputDirection = { x: -1, y: 0 };
  }
});

const getInputDirection = () => {
  return inputDirection;
};
```

Step 3:

🍎 Add the Food 🍎

```
git reset --hard
git checkout w1-step3
```

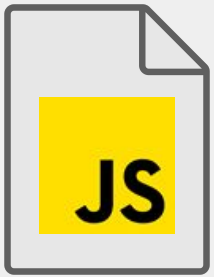
So Far

```
git reset --hard
git checkout w1-step3
```

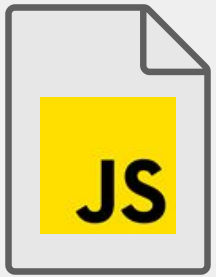
- Game Setup
  - game.js
  - index.html
- Create the Snake
  - snake.js
- Responding to Inputs
  - input.js
  - snake.js
  - index.html

Current Code

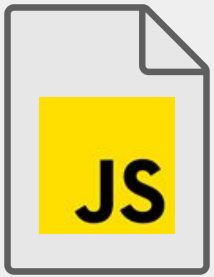
```
git reset --hard
git checkout w1-step3
```



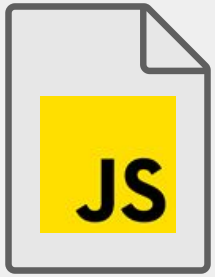
game.js



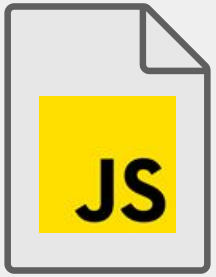
snakeUtils.js



snake.js



food.js



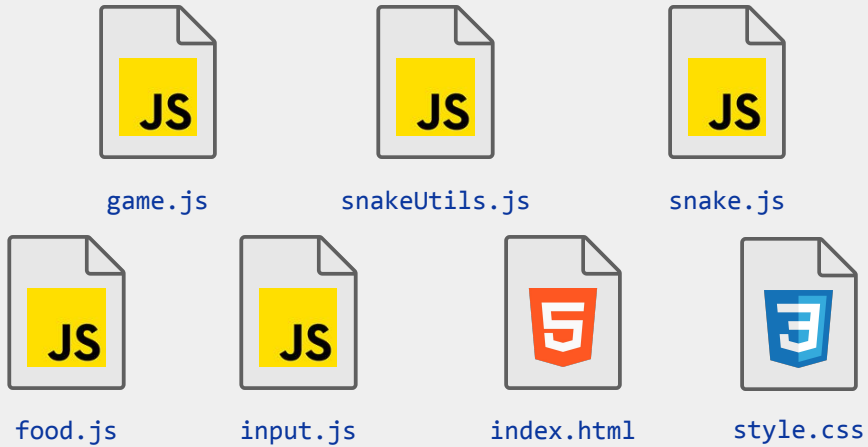
input.js

index.html

style.css

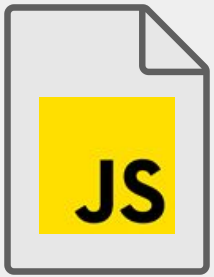
First... Why do we separate the files?

- **Modularity** - Separation of concerns
- **Avoids giant files** that are hard to navigate
- Small, clearly named, **focused files** are easier to navigate within a codebase
- Easier to add **new features** in the future

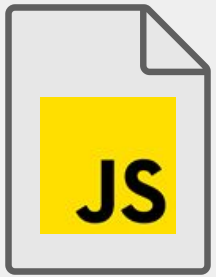


Current Code

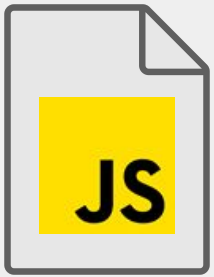
```
git reset --hard
git checkout w1-step3
```



game.js

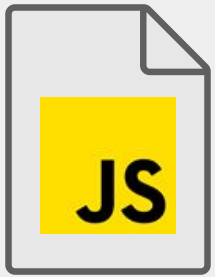


snakeUtils.js

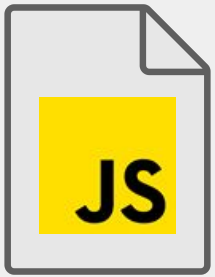


snake.js

Add to HTML!



food.js



input.js

index.html

style.css

To Do

```
git reset --hard
git checkout w1-step3
```

- Goal
  - Add food for the snake to eat
- That leaves us with the tasks:
  - Create the food
  - Create a function to update the food
  - Update the food

food.js

game.js

Don’t forget about snakeUtils.js!!!

Step 3: Food

food.js

game.js

```
git reset --hard
git checkout w1-step3
```

Tasks:

1. Create the food
2. Create a function to update the food
3. Update the food

Step 3: Food

food.js

game.js

```
git reset --hard
git checkout w1-step3
```

Tasks:

1. Create the food
    a. Initialize food
2. Create a function to update the food
    a. When do we update food?
    b. What else do we need to do when updating food?
3. Update the food
    a. Where does it need updated?
    b. How can we update the food?

Implementing Step 3

Step 3: Food

food.js

game.js

```
git reset --hard
git checkout w1-step3
```

Tasks:

1. **Create the food**
   a. Initialize food
2. Create a function to update the food
   a. When do we update food?
   b. What else do we need to do when updating food?
3. Update the food
   a. Where does it need updated?
   b. How can we update the food?

food.js

```
let food = { x: 4, y: 16 };
```

Step 3: Food

food.js

game.js

```
git reset --hard
git checkout w1-step3
```

Tasks:

1. Create the food
   a. Initialize food
2. **Create a function to update the food**
   a. When do we update food?
   b. What else do we need to do when updating food?
3. Update the food
   a. Where does it need updated?
   b. How can we update the food?

food.js

```
let food = { x: 4, y: 16 };
```

food.js

```
const updateFood = () => {
  if (onSnake(food)) {
    growSnake();
    food = getNewFoodPosition();
  }
};
```

Step 3: Food

food.js

game.js

```
git reset --hard
git checkout w1-step3
```

Tasks:

1. Create the food
   a. Initialize food
2. Create a function to update the food
   a. When do we update food?
   b. What else do we need to do when updating food?
3. **Update the food**
   a. Where does it need updated?
   b. How can we update the food?

food.js

```
let food = { x: 4, y: 16 };
```

food.js

```
const updateFood = () => {
  if (onSnake(food)) {
    growSnake();
    food = getNewFoodPosition();
  }
};
```

game.js

```
const update = () => {
  console.log('Updating');
  updateSnake();
  // TODO: Update the food
};
```

```
const update = () => {
  console.log('Updating');
  updateSnake();
  updateFood();
};
```

Step 4:

😵 Add Game Over 😵

```
git reset --hard
git checkout w1-step4
```

To Do

```
git reset --hard
git checkout w1-step4
```

- Goal
  - Allow the game to end
- That leaves us with the tasks:
  - Create a function to check if game is over
  - Update whether game is lost
  - Add a ‘Game Over’ alert
  - Define the interval ID and clear interval

Don’t forget about snakeUtils.js!!!

game.js

Step 4: Game Over

game.js

```
git reset --hard
git checkout w1-step4
```

Tasks:

1. Create a function to check if game is over

2. Update whether game is lost

3. Add a ‘Game Over’ Alert

4. Define the interval ID and clear interval

Step 4: Game Over

game.js

```
git reset --hard
git checkout w1-step4
```

Tasks:

1. Create a function to check if game is over
    a. Remember snakeUtils.js!
    b. && is an AND ; || is an OR
2. Update whether game is lost
    a. How can we keep track of this?
3. Add a ‘Game Over’ Alert
    a. Syntax for alert: alert(...)
    b. When do we want to send this alert?
4. Define the interval ID and clear interval
    a. Check the MDN Docs for setInterval()!
    b. How can we clear the interval? Remember the docs!

Implementing Step 4

Step 4: Game Over

game.js

```
git reset --hard
git checkout w1-step4
```

Tasks:

```
const checkGameOver = () => {
  return snakeOutOfBounds() || snakeIntersectSelf();
};
```

1. **Create a function to check if game is over**
    a. Remember snakeUtils.js!
    b. && is an AND ; || is an OR
2. Update whether game is lost
    a. How can we keep track of this?
3. Add a 'Game Over' Alert
    a. Syntax for alert: alert(...)
    b. When do we want to send this alert?
4. Define the interval ID and clear interval
    a. Check the MDN Docs for setInterval()!
    b. How can we clear the interval? Remember the docs!

Step 4: Game Over

game.js

```
git reset --hard
git checkout w1-step4
```

Tasks:

1. Create a function to check if game is over
   a. Remember snakeUtils.js!
   b. && is an AND ; || is an OR
2. **Update whether game is lost**
   a. How can we keep track of this?
3. Add a 'Game Over' Alert
   a. Syntax for alert: alert(...)
   b. When do we want to send this alert?
4. Define the interval ID and clear interval
   a. Check the MDN Docs for setInterval()!
   b. How can we clear the interval? Remember the docs!

```
const checkGameOver = () => {
  return snakeOutOfBounds() || snakeIntersectSelf();
};
```

```
let gameOver = false;
```

```
const update = () => {
  console.log('Updating');
  updateSnake();
  updateFood();
  // TODO: Update Game State
};
```

```
const update = () => {
  console.log("Updating");
  updateSnake();
  updateFood();
  isGameOver = checkGameOver();
};
```

Step 4: Game Over

game.js

```
git reset --hard
git checkout w1-step4
```

Tasks:

1. Create a function to check if game is over
   a. Remember snakeUtils.js!
   b. && is an AND ; || is an OR
2. Update whether game is lost
   a. How can we keep track of this?
3. **Add a 'Game Over' Alert**
   a. Syntax for alert: alert(...)
   b. When do we want to send this alert?
4. Define the interval ID and clear interval
   a. Check the MDN Docs for setInterval()!
   b. How can we clear the interval? Remember the docs!

```
const checkGameOver = () => {
  return snakeOutOfBounds() || snakeIntersectSelf();
};
```

```
let gameOver = false;
```

```
const update = () => {
  console.log('Updating');
  updateSnake();
  updateFood();
  // TODO: Update Game State
};
```

```
const update = () => {
  console.log("Updating");
  updateSnake();
  updateFood();
  isGameOver = checkGameOver();
};
```

```
const main = () => {
  update();
  draw();
  // TODO: Add Game Over Alert
};
```

```
const main = () => {
  update();
  draw();
  if (isGameOver) {
    alert("Game Over");
    clearInterval(gameLoop);
  }
};
```

Step 4: Game Over

game.js

```
git reset --hard
git checkout w1-step4
```

Tasks:

1. Create a function to check if game is over
   a. Remember snakeUtils.js!
   b. && is an AND ; || is an OR
2. Update whether game is lost
   a. How can we keep track of this?
3. Add a 'Game Over' Alert
   a. Syntax for alert: alert(...)
   b. When do we want to send this alert?
4. **Define the interval ID and clear interval**
   a. Check the MDN Docs for setInterval()!
   b. How can we clear the interval? Remember the docs!

```
const checkGameOver = () => {
  return snakeOutOfBounds() || snakeIntersectSelf();
};
```

```
let isGameOver = false;
```

```
const update = () => {
  console.log('Updating');
  updateSnake();
  updateFood();
  // TODO: Update Game State
};
```

```
const update = () => {
  console.log("Updating");
  updateSnake();
  updateFood();
  isGameOver = checkGameOver();
};
```

```
const main = () => {
  update();
  draw();
  // TODO: Add Game Over Alert
};
```

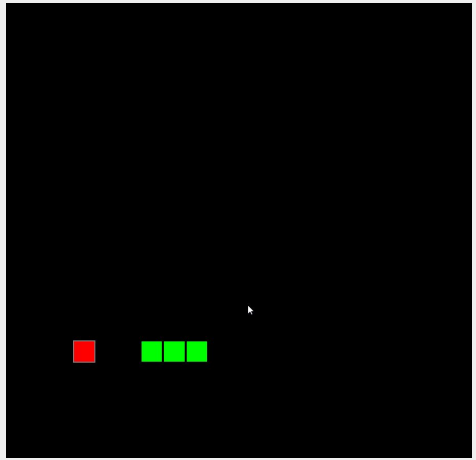
```
const main = () => {
  update();
  draw();
  if (isGameOver) {
    alert("Game Over");
    clearInterval(gameLoop);
  }
};
```

```
let gameLoop = setInterval(main, 1000/SNAKE_SPEED);
```

```
if (gameOver) {
  alert('Game Over');
  clearInterval(gameLoop);
}
```

Finished Game

```
git reset --hard
git checkout w1-complete
```



Challenge
Reset the Game

```
git reset --hard
git checkout w1-complete
```

Challenge: Restart the Game

```
git reset --hard
git checkout w1-complete
```

- **Goal**
  - Let the user restart the game whenever they want, particularly after game over
- **Tasks**:
  - Create an input event listener
    - **Hint:** Recall Step 2 to add an additional keyboard input
  - Reset the snake's position
  - Reset the input direction
  - Restart the game loop

`input.js` `game.js` `snake.js`

Hint: Create a function **`resetGame()`** in `game.js`

```
git reset --hard
git checkout w1-challenge
```

to see our solution

Lunch time!!

Please be back by 12:45 PM ET ;)