Canva Digital Design FrontEnd Challenge - Snake!

This challenge is divided into three steps.
Overall, this should take somewhere between 2-3 hours, depending on how smoothly you progress.
Take a little time to read this PDF before starting!

There is a provided `index.html` which will load `src/game.candidate.js` as a script with `type="module"`. What happens beyond that is up to you, but each module should be exposed in a testable way, and implementing the public API definitions required as specified in these instructions.

For each part there is a code comment where the code for that part of work should start. Not all of your code needs to be in the commented Part sections, just the main functions being used/modified.

If you find yourself running low on time, we would appreciate at least a description of how you would implement the outstanding tasks (in comment form, in the relevant source files).

When submitting, please make sure to remove `node_modules` before zipping!

Requirements

Before starting, please make sure the following are setup on your system.

- Node >= 9
- NPM, or Yarn
- Chrome - to make sure it works in Chrome!

You may use any feature available in the latest version of Chrome, and we will be using ***Chrome*** and ***Jest*** for assessing & testing. Please make sure the provided tests are working, and that `index.html` runs correctly in Chrome. We're using `babel-jest` for testing with modules/ES6.

You ***MAY NOT***

- Use any packages on NPM (or any other package manager)
- Use any CSS processors
- Use any JS packages / frameworks / libraries
- Use any JS/CSS packaging/bundling, eg. `webpack`

Common APIs and provided files

There are two data types provided in `js/data/data_types.js`.
These are definitions for Tiles, and Direction values. These should be used in your code.

In the zip, you are provided with following general files;

- `instructions.md` & `.pdf`
- `package.json`
- `images/**` - used as reference for rendering section
- `src/index.html`
- `tests/**` - some test spec files and helpers

A `package.json` is provided, which has some basic setup for `jest`. As always with package.json, there are scripts provided, one for running the test, and another for running with a development server.

Along with all the files for each section, you may also add `tests/candidate.test.js` and write your own tests should you like. You won't be assessed on your tests, but it may help you finish the problem.

Part 1. Game UI

Change `src/game.candidate.js` (starting @ *line 133*) to have the following.

1. Arrow key movement of the Snake. Re-use method `move(direction)`.
2. Pausing when space key is used (`togglePaused`).
3. Change to use `requestAnimationFrame` instead of `setTimeout` for running.

Part 2. Snake Game

Implement the functionality for `src/snake.candidate.js`.

A Snake game has four tiles, as defined by `Tiles` in `src/js/data/data_types.js`.

- `Empty` is a free area/tile.
- `Wall` is wall around the edge of the board.
- `Snake` is a tile which has part of the snake in that area.
- `Fruit` which can be eaten by the snake to grow.

A Snake game has the following public API to implement:

- `constructor(SnakeConfigParams)` - constructor with simple config object.
  - `{ rows: number, columns: number, nextFruitFn: Function }`
  - `rows` - number of rows to make board with
  - `columns` - number of columns to make board with
  - `nextFruitFn` - *[optional]* a function which will return the `Position` to put next fruit when generating a new fruit.
- `getBoard()` - get the 2D board, `number[][]`.
- `getTile(Position)` - get the tile value (`number`) at `Position {x, y}`
- `setDirection(Direction)` - set current movement direction.
- `tick() => TickReturnObject` - progress the snake forward one spot and return the result.

If no `nextFruitFn` is provided (`undefined`), then the snake game should put fruit in a random position not inside the snake body or inside a wall.

If a function is provided, it should be called with no parameters, and will return the position where the next fruit tile should be placed. This completely replaces any random fruit generation - when a fruit is eaten, a new one should be generated getting a new position from this function.

Each time `tick()` is called, the snake should move forward one tile, or turn and move if direction has been changed. One of four things may have happened ..

- The snake ran into a Wall - game over.
- The snake run into itself - game over.
- The snake went into an Empty area.
    - Snake head moves onto empty tile, and body pieces all move.
- The snake went onto a Fruit, and ate the fruit
    - Snake should grow forward onto that tile. Tail will remain in same tile.
    - Remove current `Fruit` from board.
    - Make another `Fruit` onto board.

The `TickReturn` object should look like:

```
// TickReturn {}
{
  gameOver: boolean, // did this tick/move finish the game?
  eating: boolean, // did the snake eat this tick/move?
  changes: TickChange[] // list of board changes that happened this turn.
}
```

```
// TickChange {}
{
  position: Position, // { x, y } position
  tileValue: number, // new Tiles value for this position
}
```

Other notes/requirements - your Snake game should ..

- Use `0,0` as the top/left location.
    - `NOTE` - when doing 2D access, the `row` will be first.
    - eg. if we want to see spot `x,y`, then `snake.getBoard()[y][x]` should work.
- All snake boards should have a ring of walls on the edge positions.
    - ie. The first row of any board should be all `Wall` tiles.
    - The left/right edges are also all `Wall` tiles.
- Snake config rows & columns should be integers >= 5, and throw an error if not. `5x5` will leave a `3x3` playing area, and anything smaller doesn't work!
- starting snake is 1 tile, in the center, with direction/moving right.
    - Notation here is `Y` columns x `Z` rows
    - for a 21x21 board, this would make the snake start tile be `{ x: 10, y: 10 }`
    - for a 20x20 board, the snake would also start at `{ x: 10, y: 10 }`
    - for a 15x8 board, the snake will start at `{ x: 7, y: 4 }`
- remove `defaultBoard` since it's no longer needed.

Part 3. Renderer

Task 3 is to update what our game looks like by writing a new renderer. Again, you may use, and are encouraged to use, any features of ***Chrome***.

The old renderer is using tables, and looks hard to maintain! The old renderer isn't even using our common data types. Make a new renderer which is clear and easy to understand.

Your renderer should aim to match `screenshot.png` in the images folder

Using `src/js/renderers/renderer.table.js` & `screenshot.png` as a reference,

- Edit `renderer.candidate.js` & `game.candidate.css`.
- Change `game.candidate.js` to use `candidateRenderer` ~ *line 160* (it's already being imported)

Things to note:

- There is a tile sized margin around the page.
- The tiles are twice as large than before.
- There are two pixels between snake segments, and between snake & fruit.
- Walls are connected, & the walls are 1/3 of the size of the tile.
- You don't need to worry about changing the 'Game Over' message.
- The rendered tiles are centered in their tile "area".