Web Programming

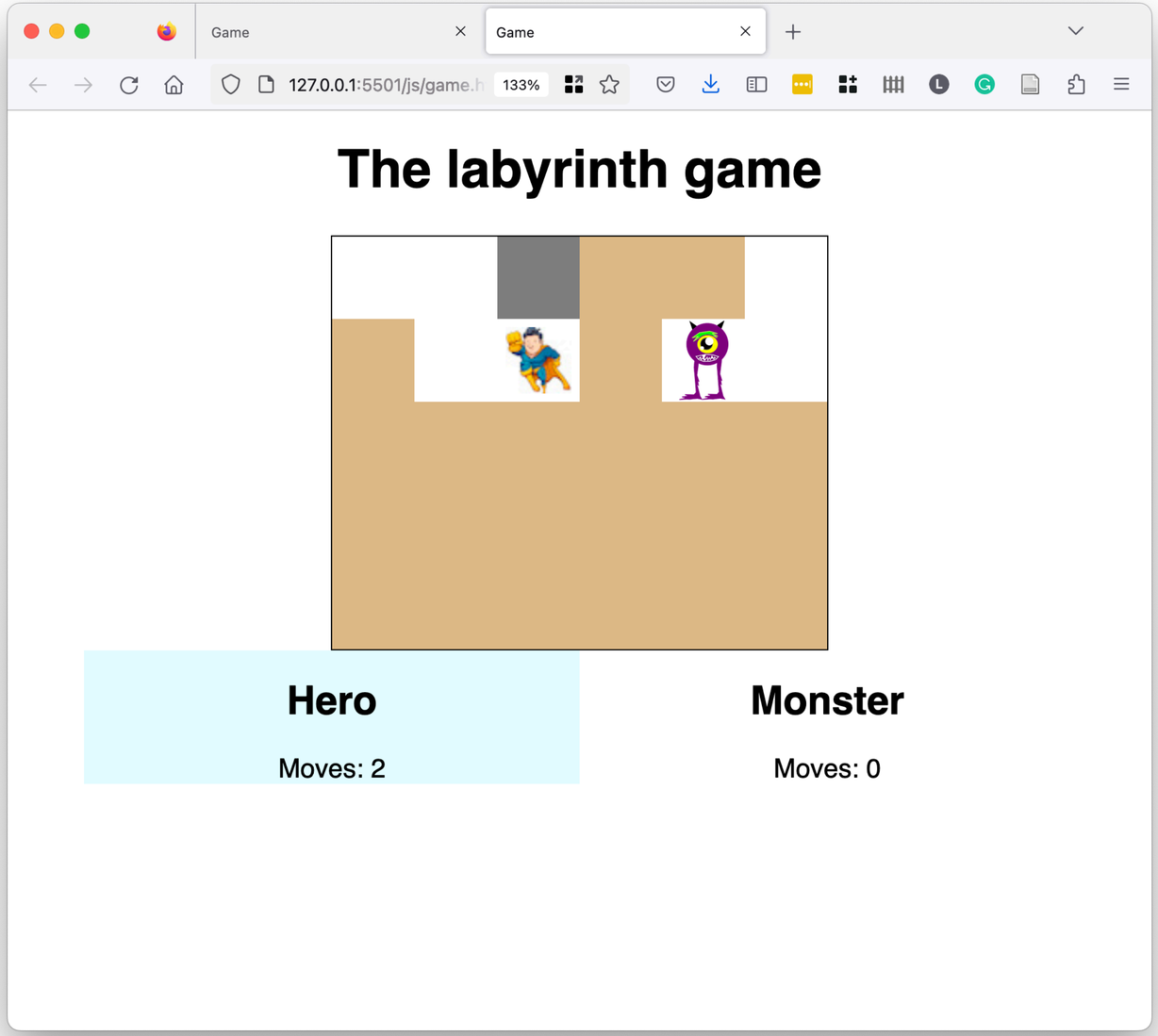
Vue.js example and CLI

Leander Jehl | University of Stavanger

Example

Labyrinth game

- Two players take turns
  - but hero always moves twice
- Flexible labyrinth layout
  - Fields: free, wall, exit, pit
  - Fields are hidden first



Labyrinth Game - Step 1

Layout (HTML and CSS)

- Create a static HTML page

index.html

```
<body>
    <h1>The labyrinth game</h1>
    <main>
        <div>
        <div id="board">
            <div class="field">
                <img class="hero" src="hero.jpg" alt="hero">
                <img class="monster" src="monster.png">
            </div> ...

        </div>
    </div>
    <div id="stats">
        <div id="herostats" class="turn">
            <h2>Hero</h2>
            <div>Moves: <span class="movescount">0</span> </div>
        </div>
        <div id="monsterstats">
            <h2>Monster</h2>
            <div>Moves: <span class="movescount">0</span> </div>
        </div>
    </div>
    </main>
    <div id="winner">
        <h2 class="monster">The monster won!</h2>
        <h2 class="hero">The hero won!</h2>
    </div>
</body>
```

style.css

```
#board {
    margin: auto;
    display: flex;
    flex-wrap: wrap;
    width: 300px;
    border: 1px solid black;
}
.field {
    width: 50px;
    height: 50px;
    display: flex;
    align-items: center;
    justify-content: center;
}

.field.hidden {
    background-color: burlywood;
}
.wall {
    background-color: grey;
}
.pit {
    background-color: black;
}
.exit {
    background-color: yellow;
}
.field img {
    display: none;
}
.field img.monster {
    height: 48px;
}
```

Labyrinth Game - Step 1

Layout (HTML and CSS)

- Create a static HTML page
- Create classes for effects

```
.field img {
    display: none;
}
.hero img.hero {
    display: block;
}
.field.hidden {
    background-color: burlywood;
}
```

Move hero by moving the **hero** class

Show a field by removing the **hidden** class

Example #1

examples/js/labyrinth/htmlcss

index.html

```
<body>
    <h1>The labyrinth game</h1>
    <main>
        <div>
        <div id="board">
            <div class="field">
                <img class="hero" src="hero.jpg" alt="hero">
                <img class="monster" src="monster.png">
            </div> ...

        </div>
    </div>
    <div id="stats">
        <div id="herostats" class="turn">
            <h2>Hero</h2>
            <div>Moves: <span class="movescount">0</span> </div>
        </div>
        <div id="monsterstats">
            <h2>Monster</h2>
            <div>Moves: <span class="movescount">0</span> </div>
        </div>
    </div>
    </main>
    <div id="winner">
        <h2 class="monster">The monster won!</h2>
        <h2 class="hero">The hero won!</h2>
    </div>
</body>
```

style.css

```
#board {
    margin: auto;
    display: flex;
    flex-wrap: wrap;
    width: 300px;
    border: 1px solid black;
}
.field {
    width: 50px;
    height: 50px;
    display: flex;
    align-items: center;
    justify-content: center;
}

.field.hidden {
    background-color: burlywood;
}
.wall {
    background-color: grey;
}
.pit {
    background-color: black;
}
.exit {
    background-color: yellow;
}
.field img {
    display: none;
}
.field img.monster {
    height: 48px;
}
```

Labyrinth Game - Step 2

Game logic

- Model logic as JS classes
- Model logic without thinking about HTML

  `class Game`

  - who's turn, moves, winner

  `class Board`

  - player positions and moving

  `class Field`

  - Field contains: wall?, exit?, pit?, hidden?, hero?, monster?

Labyrinth Game - Step 2

Game logic

`class Game`

- who's turn
- how many moves left
- is game ended
- who has won

```
class Game {
    constructor(){
        this.players=['hero','monster'];
        this.turn=0;
        this.heromoves = 2;
        this.monstermoves = 0;
        this.winner = "";
        this.ended = false;
           }
    player(){ //return player }
    move(){ //register move }
    eat(){ //monster wins }
    pit(){ //player looses }
    exit(){ //player wins }
    win(winner){ //set winner }
```

Labyrinth Game - Step 2

Game logic

`class Game`

- who's turn
- how many moves left
- is game ended
- who has won

```
class Game {
    constructor(){
        this.players=['hero','monster'];
        this.turn=0;
        this.heromoves = 2;
        this.monstermoves = 0;
        this.winner = "";
        this.ended = false;
           }
    player(){ //return player }
    move(){ //register move }
    eat(){ //monster wins }
    pit(){ //player looses }
    exit(){ //player wins }
    win(winner){ //set winner }
```

Labyrinth Game - Step 2

Game logic

`class Field`

- type (wall, exit, pit, free)
- hero?
- monster?
- hidden?

Check for errors

```
class Field{
    constructor(type){
        let types = ['wall', 'free', 'exit', 'pit'];
        if (types.indexOf(type) == -1){
            throw “cannot create field with wrong type”;
        }
        this.type = type // 'wall','free', 'exit', or 'pit'
        this.hidden = true;
        this.hero = false;
        this.monster = false;
    }
    show(){
        this.hidden = false;
    }
    set(player){
        this[player]=true;
        this.hidden = false;
    }
    unset(player){
        this[player]=false;
    }
}
```

Labyrinth Game - Step 2

Game logic

board layouts

- Field type

`Math.random()` returns a random number between 0 and 1 in [0,1)

```
const board1 =
    ['free','free','wall','wall','wall','free',
     'wall','free','free','free','wall','free',
     'wall','free','wall','free','free','free',
     'wall','free','wall','wall','wall','exit',
     'wall','free','pit','free','wall','free'];
const board2 =  ...
const board3 =  …

function getRandomBoard(){
    // randomly pick 0, 1 or 2
    let r = Math.floor(Math.random()*3)
    // use r to pick a board
    let boards =[board1,board2,board3]
    return boards[r];
}
function generatefields(){ }
```

Labyrinth Game - Step 2

Game logic

`class Board`

- type (wall, exit, pit, free)
- hero?
- monster?
- hidden?

```
class Board{
    constructor(){
        this.heroXY = [0,0];    // x,y coordinates of hero
        this.monsterXY = [5,0]; // x,y coordinates of monster
        this.xmax=5;            // x ranges from 0 to 5
        this.ymax=4;            // y ranges from 0 to 4
        this.fields = []; // one Field for each position on the board
        this.game = new Game(); // Game instance
    }
    playerXY(){ // get hero or moster position }
    start(){ // generate fields set players to start positions }
    hasField(x,y){ // is (x,y) inside board? }
    getField(x,y){ // get correct Field }
    setplayer(x,y,player){ // move player, check exit&pit }
    trymove(x,y,player){ // check for turn&wall }
    moveright(player){ // x+1 }
    moveleft(player){ // x-1 }
    moveup(player){ // y-1 }
    movedown(player){ // y+1 }
}
```

Labyrinth Game - Step 2

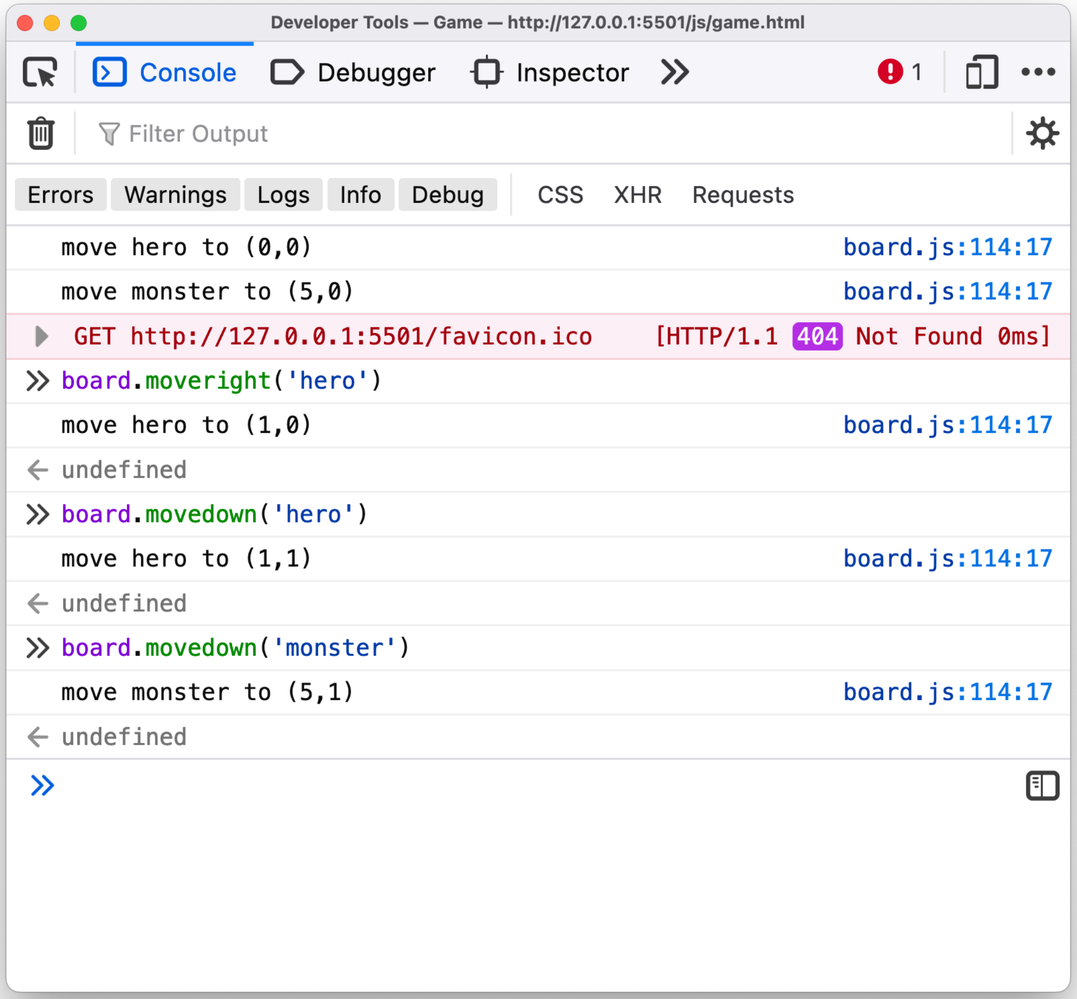
Game logic

- Create global object

```
let board = new Board();
```

- Test in console

```
console.log(`move ${player} to (${x},${y})`);
```



Labyrinth Game - Step 3

Event listeners

Can also be done after dynamic display!

- Add event listener

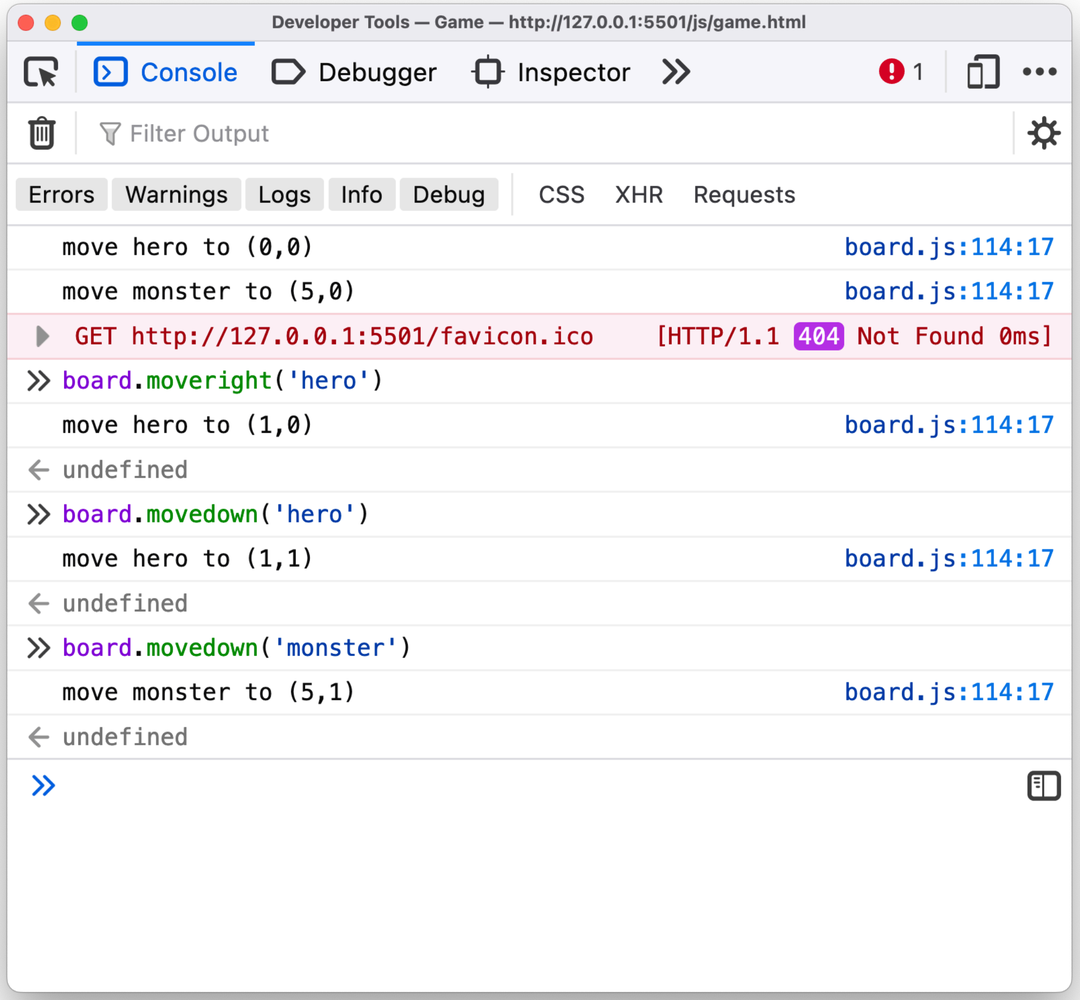
```
window.onload = function() {
    document.body.addEventListener("keyup", keyhandle);
}
```

- Handle correct keys

```
function keyhandle(event) {
    switch (event.keyCode) {
        case 37: // left
            board.moveleft('hero');
            break;
        case 65: // left
            board.moveleft('monster');
            break;
        ...
        default: // any other key
            // do nothing
            console.log(event.keyCode);
            break;
    }
}
```

Example #2

examples/js/labyrinth/jsonly



Labyrinth Game - Step 4 (pure JS)

Display

- Write turn and moves to Stats

```
if (this.turn == 0)
            document.getElementById("herostats").classList.add('turn');
document.querySelector("#herostats .movescount").textContent = this.heromoves;
```

- Apply classes to fields
- add object to Field

```
class Field{
    constructor(type, element){
        this.type = type
        this.hidden = true;
        this.hero = false;
        this.monster = false;
        this.element = element;
        this.settype();
        this.element.classList.add('hidden');

    }
    settype(){ }
}
```

Labyrinth Game - Step 4 (pure JS)

Display

- add object to Field

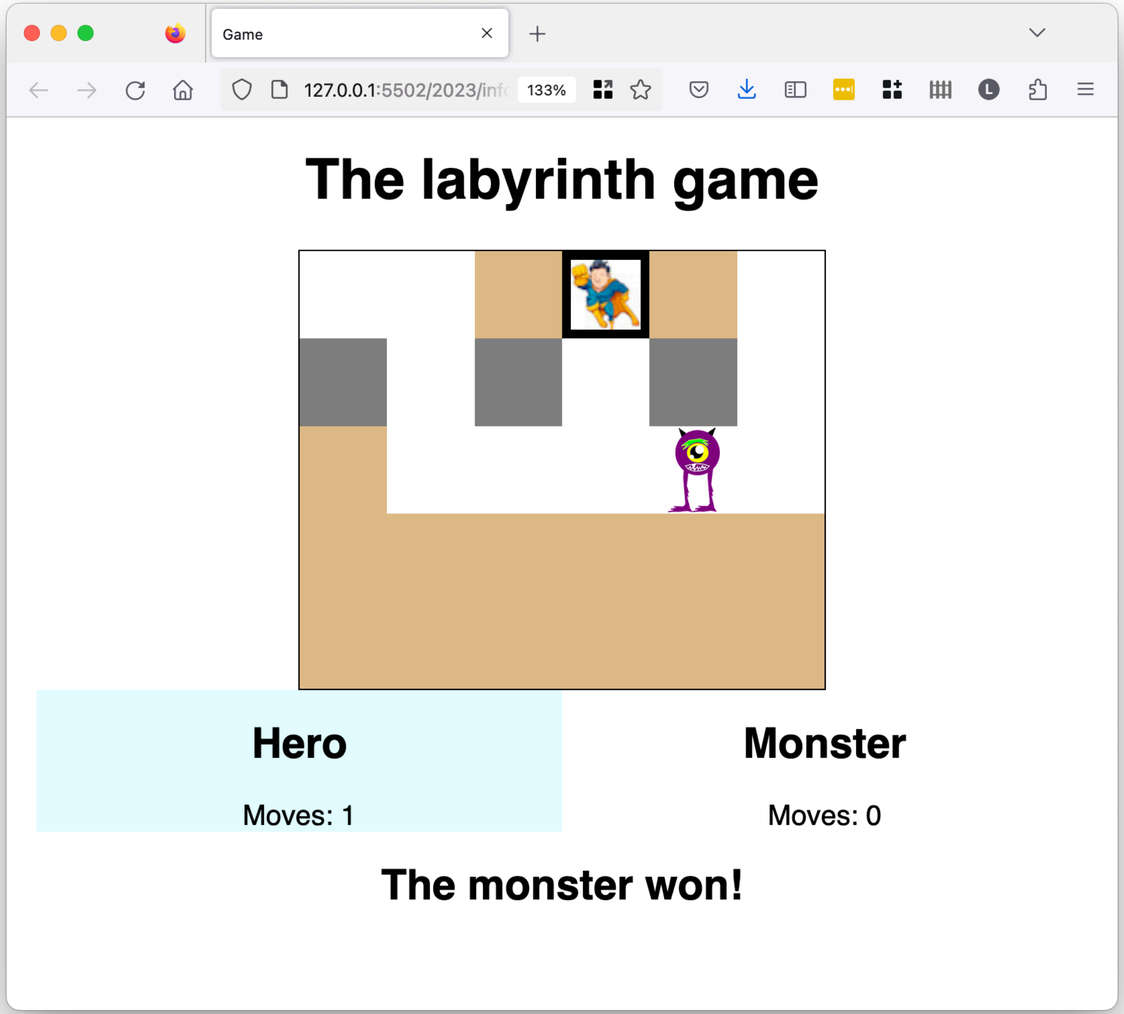
```
function generatefields(){
    let fields = [];
    let layout = getRandomBoard();

    let fieldelements = document.getElementsByClassName("field");
    if (fieldelements.length != layout.length){
        throw `Layout does not fit to page:
                ${fieldelements.length} field in html and
                ${layout.length} fields in layout.`
    }

    for (let i=0; i<layout.length; i++){
        fields.push(new Field(layout[i], fieldelements[i]));
    }
    return fields;
}
```

Example #3

examples/js/labyrinth/js



Labyrinth Game - Step 4 (Vue)

Display

- Create Board in data
- Handle keyup in methods

```
let app = Vue.createApp({
    data: function(){
        return {
            board: new Board()
        }
    },
    methods:{
        handle: function(event){
            switch (event.keyCode) {
                case 37: // left
                    this.board.moveleft('hero');
                    break;
...
```

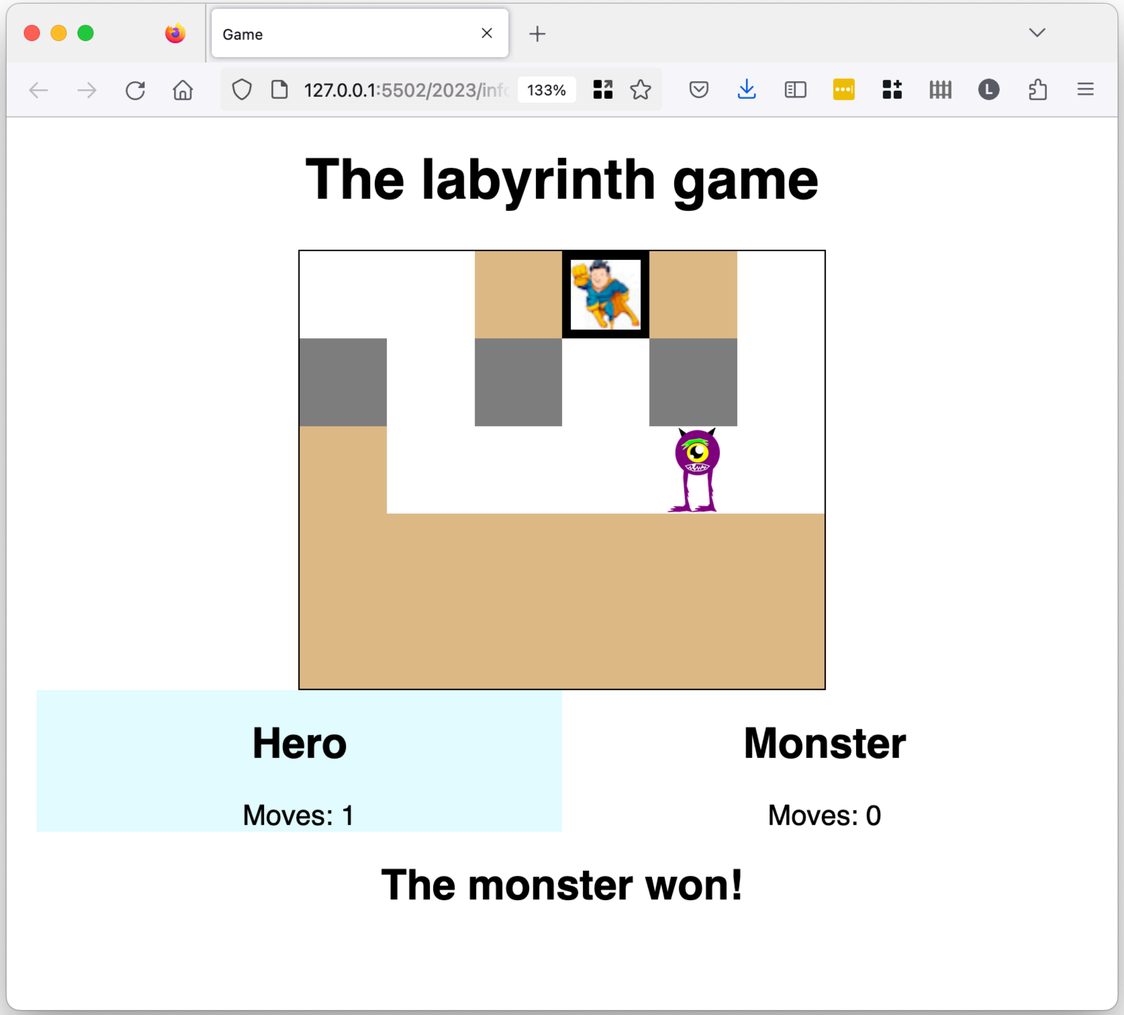
- Listen to keyup

tabindex="0" enables key events on not input elements.

```
<main v-on:keyup="handle" tabindex="0">
```

Example #4

examples/js/labyrinth/vue



Not curriculum!
This is how you develop in the real world!

CLI and single file components

- CLI is a tool to set up a new vuejs project.
  - Uses webpack
  - Web pack avoids including different files in index.html
- Single file components allow to have
  - nicely highlighted templates
  - JavaScript component definition
  - CSS scoped to this component
  - **All in one file**

Not curriculum!
This is how you develop in the real world!

CLI and Single file components

- Requirements:
  - Install **`node.js`** and **`npm`** https://nodejs.org/en/download/

```
~: node -v
v12.10.0
~: npm -v
6.11.3
```

For examples to work, use node version <=16

  - Install vue cli

```
~: npm install @vue/cli
```

may have to run as root

  - Create new project

```
~: vue create my-test-project
```

  - Choose default tools

```
? Please pick a preset:
  Default ([Vue 2] babel, eslint)
❯ Default (Vue 3 Preview) ([Vue 3] babel, esl
```

VSCode: Install Extension Vetur

Not curriculum!
This is how you develop in the real world!

Vue CLI setup

VSCode: Install Extension Vetur

- Folder structure

```
my-test-project
 > node_modules     // JS libraries and dependencies, e.g. vue
 > public           // Static files, contains index.html
 > src              // All your code is here
 babel.config.js  // Babel configuration
 package.lock.json // Dependency versions (for npm)
 package.json       // npm configuration
 README.md
 vue.config.js      // add this file
```

- src folder

```
src
 > assets           // More static assets, e.g. images
 > components       // Your components
 App.vue            // Main component
 main.js            // create and mount app
```

Not curriculum!
This is how you develop in the real world!

Single file components

- Components can now be specified in .vue files:

```
<template>
    <!-- The template for your component -->
    <form>
        <input type="text" v-model="song">
        ...
    </form>
</template>

<script>
    // define your component in JavaScript
</script>

<style scoped>
    // CSS queries applied only to this component
</style>
```

Not curriculum!
This is how you develop in the real world!

ES6 import and export

- Using CLI, components are not defined globally,

```
// globally defined component:
app.component(“song-form",{ });
```

- Instead the definition of a component is exported

SongForm.vue

```
// export component configuration
export default {
    template: ...
    methods: ...
};
```

Only one default export per file.

App.vue

```
// import component
import songForm from './components/SongForm'

export default {
    template: ...,
    // use songForm in this component
    components: {
        songForm,
    }
};
```

Not curriculum!
This is how you develop in the real world!

Example #6

examples/js/vue3/playlist-CLI

```
../playlist: npm run serve
```

Starts a development server, serving your app.

```
<template>
  <div id="app">
    <song-form></song-form>

    <ul id="playlist">
      <song-list-item
        v-for="(song, index) in playlist"
        v-bind:song="song"
        v-bind:index="index"
        v-bind:key="index"
      ></song-list-item>
    </ul>
  </div>
</template>
```

```
<script>
import gState from './data.js'

import songForm from './components/SongForm'
import songListItem from './components/SongListItem'

export default {
  name: 'App',
  data: function(){
    return {
      playlist: gState.playlist,
    }
  },
  components: {
    songForm,
    songListItem
  }
}
</script>
```

Not curriculum!
This is how you develop in the real world!

Submitting

- Add config file **`vue.config.js`**

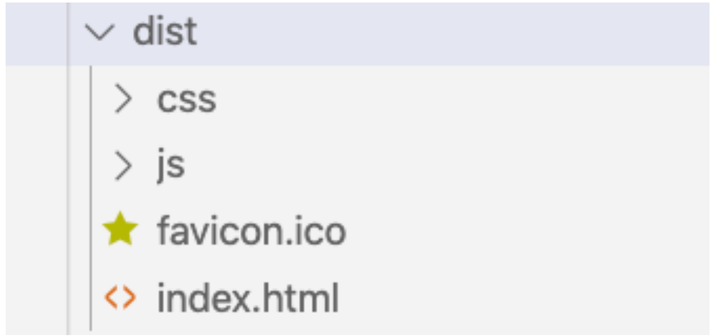
```
// vue.config.js
module.exports = {
    // change to relative path
    publicPath: './'
}
```

- Run: **`npm run build`**

```
../playlist: npm run build
```

You can submit like this.
In grading/approving, we will not consider your code, only functionality.

- Submit files from **`dist`** folder



Not curriculum!
This is how you develop in the real world!

Exercise #4, #4b

github.com/dat310-2023/info/tree/master/
exercises/js/vue_cli

Routing and vuex

Not curriculum!
This is how you develop in the real world!

- You can choose a setup including vuex and routing

```
~: vue create my-test-project2
```

```
? Please pick a preset:
  default (babel, eslint)
❯ Manually select features
```

```
? Check the features needed for your project:
 ◉ Babel
 ○ TypeScript
 ○ Progressive Web App (PWA) Support
 ◉ Router
❯◉ Vuex
 ○ CSS Pre-processors
 ◉ Linter / Formatter
 ○ Unit Testing
 ○ E2E Testing
```

Select Vuex and Router using <space>

Not curriculum!
This is how you develop in the real world!

Vue CLI setup

- Folder structure with router and vuex (store)
  - src folder

```
src
 > assets            // More static assets, e.g. images
 > components        // Your components
 > router
   index.js          // Define you routes here
 > store
   index.js          // Add state mutations getters,… here
 > views             // Usually holds components that are used for routing
 App.vue             // Main component
 main.js             // Dependency versions (for npm)
```

Not curriculum!
This is how you develop in the real world!

Example #7

examples/js/vue3/grades-router-vuex-CLI

```
../playlist: npm run serve
```

Starts a development server, serving your app.

App.vue

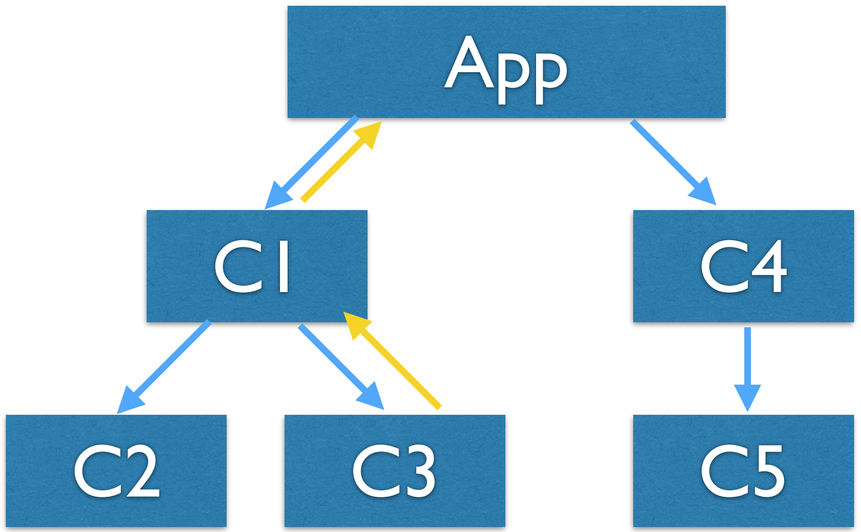
```
<template>
  <div id="app">
    <router-view/>
  </div>
</template>
```

router/index.js

```
const routes = [
  {
    path: '/',
    name: 'Home',
    component: Home
  },
  {
    path: '/student/:student_no',
    name: 'Student',
    props: true,
    component: Student
  },
  {
    path: '/course/:course_id',
    name: 'Course',
    props: true,
    component: Course
  }
]
```

State management

- If multiple components access the same state, it needs to be passed down using props and changed using events.
  - State shared by C3 and C5 must be located in App.
  - If shared state is changed in C3, change is propagated using events and props

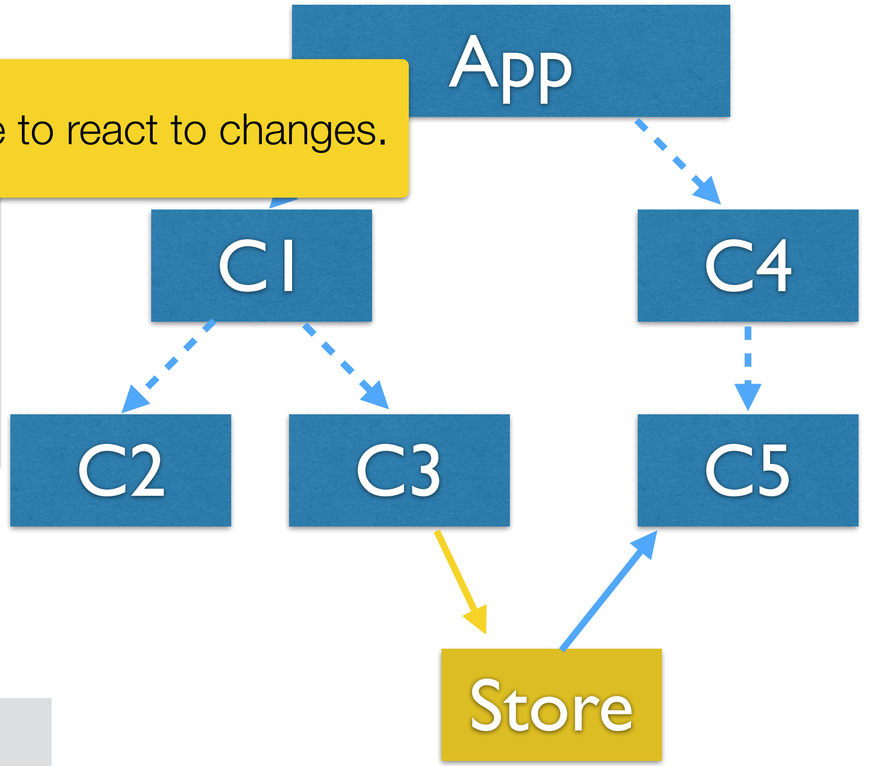


A different pattern: External store

- Outside of your app, define a store:

```
class DataStore{
    constructor(data){
        this.data = data;
    }
    getter(){}
    setter(){}
}
let store = Vue.reactive(new DataStore(data));
```

Vue.reactive( ) allows Vue to react to changes.



- Retrieve data from store,
  e.g. on component creation

```
  data() {
       return store.data;
   }
```

(read the docs)

Example #2

examples/js/vue3/global-state-fruits/index.html

form.js

```
let fruitFormC = { ...
    methods: {
        add: function(){
            store.addFruit(this.name, _);
        }
    }
}
```

Update global state instead of emitting event.

list.js

```
let favoriteC= {
    computed:{
        fruits: function(){
            return store.fruits;
        },
        // using getters inside computed
         // properties works
        favorites: function(){
            return store.favoriteFruits();
        },
    }
}
```

Use getters in computed

data.js

```
class Store{
    constructor(){
        this.fruits = [
            { name: "Apple",   favorite: true },
            { name: "Banana",  favorite: true },
            { name: "Pear",    favorite: true },
            { name: "Grapes",  favorite: false },
            { name: "Oranges", favorite: false },
            { name: "Kiwi",    favorite: false }
        ]
    }
    //getter
    favoriteFruits(){
        return this.fruits.filter(
            (fruit) => fruit.favorite);
    }
    //setter
    addFruit(name, isFavorite){
        this.fruits.push({name: name, favorite:
isFavorite});
    }
}

let store = Vue.reactive(new Store())
```