Architecture

July 5, 2022

Contents

1 Group 7

- Max Ornik

- Marcel Sziener
- Skyler Mayfield

2 Program Architecture

2.1 `index.html`

Contains the game canvas and links to the script files; this is what the browser loads and renders.

2.2 `anim-tester.html`

Auxiliary web page used for loading test code separately.

2.3 `js/main.js`

Contains the main program loop. This is where the html canvas gets initialised and where the assets/player/levels are prepared and initialised. The `movePlayer` function contains the logic for handling key events and keeping the player from leaving the boundaries of the map.

2.4 `js/level.js`

Contains the game levels, which are written as separate functions. These functions are enumerated so that they can be sequentially started in the main function. The levels consist of commands for placing `world_objects` and logic for placing enemies in the world.

2.5 `js/config.js`

Intended to be used for holding global variables that should be available from anywhere within the program, such that they can be easily changed in

this single location to change the game's behavior.

2.6 `js/anim-tester.js`

A file used to separately test logic for input handling and a proof of concept for animating the sprites. As the animation consists of some number of frames placed next to each other in the sprite file, animating them means showing only a single frame at a time and clearing the previous frame. It works by cropping the visible area of the sprite to the size of a single animation frame, and then moving this cropped area to the next animation frame in order. If there are six frames, this can be achieved by enumerating the frames and displaying them in a loop using `mod 6` to access the correct frame.

2.7 `js/world/character.js`

The character is a special class that contains information about the animations. The sprites have different sets of animations, which need to be played under certain situations. The player, for instance, uses a breathing animation as the idle animation, a walking animation while moving, and a slashing animation when attacking. All of this is connected with logic such that the current state is always known and can be used to play the correct animation or determine how much health the player has left. The player and the enemies both inherit the functionality of their parent class `Character` and add additional functionality specific to each case.

2.8 `js/world/pokemon_fight.js`

This file contains the logic of the pokemon style fight system. This works by starting a new `world` object specifically for the fight scene; the state of the main game world is not touched other than to remove the monster if the player wins. Drawing the fight scene works analogously to drawing a level, but the input is restricted to the options needed to make the fight work.

2.9 `js/world/world.js`

This file contains the logic for holding the state of a world. All of the objects drawn in the world belong to the world, allowing the programmer to create levels simply by making a world object and drawing in that world. The drawn objects have properties, and the world object is responsible for checking and acting on the properties; the player cannot walk through walls for instance, and when the player encounters a monster, a fight begins.

2.10 `js/world/world_objects.js`

The class hierarchy for world objects is set up so that inheritance makes adding objects with specific properties very simple. Adding a new walkable surface for instance only involves adding a child class of `Floor` and specifying which asset map contains the sprites and the index of the sprites which should be drawn.

2.11 `js/core/move.js`

This file contains movement logic that translates simple values into percentages for drawing on the responsive canvas.

2.12 `js/core/events.js`

This file contains key handling logic. The event handler can be instantiated as an object, making using it very simple.

2.13 `js/core/draw.js`

This file sets up the canvas so that it responds to the window size.

2.14 `js/core/component.js`

In order to make drawing the sprites a reality, we needed a data structure to represent an object that can be drawn on the canvas. Therefore, this class

is very general; it can be used to draw JavaScript shapes, text, or sprites from the asset pack. These objects need to be scaled with the window so that the proportions don't change.

2.15 `js/core/asset.js`

This file sets up an interface for mapping the sprite sheets from the asset pack to objeccts that can be indexed and instantiated by name. This means that we had to examine the sprite sheets and figure out the grid size, which is 16x16 pixels. This grid can then be indexed, by saying that in a certain sprite sheet there is a sprite component in column 2 row 4 that looks like a tuft of grass. This information is then used when drawing, which makes writing levels very simple.

2.16 `js/core/animation.js`

This file contains the logic for sequentially displaying animation frames in different ways; some animations are intended to loop and some are intended to be played in order once. This requires separate functions.

2.17 A note about this document

Both the Open Document and Latex versions of the document were generated using the free and open source tool `pandoc`.