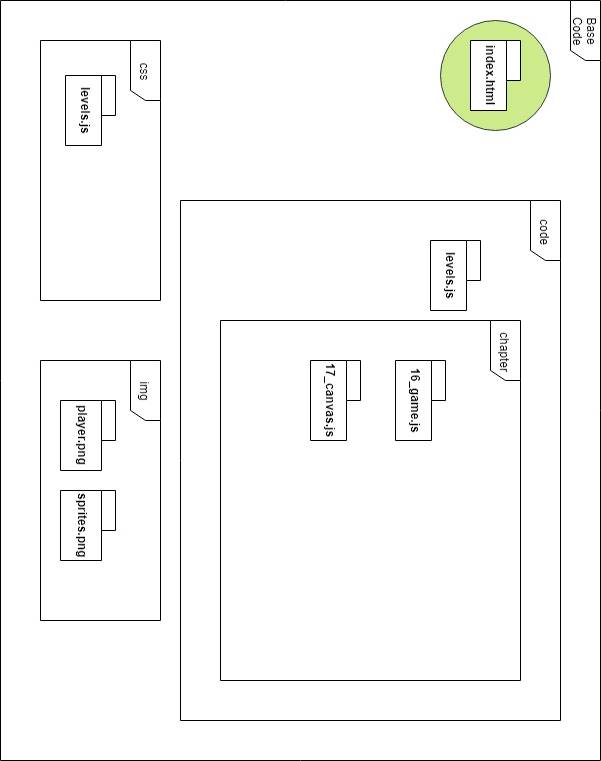
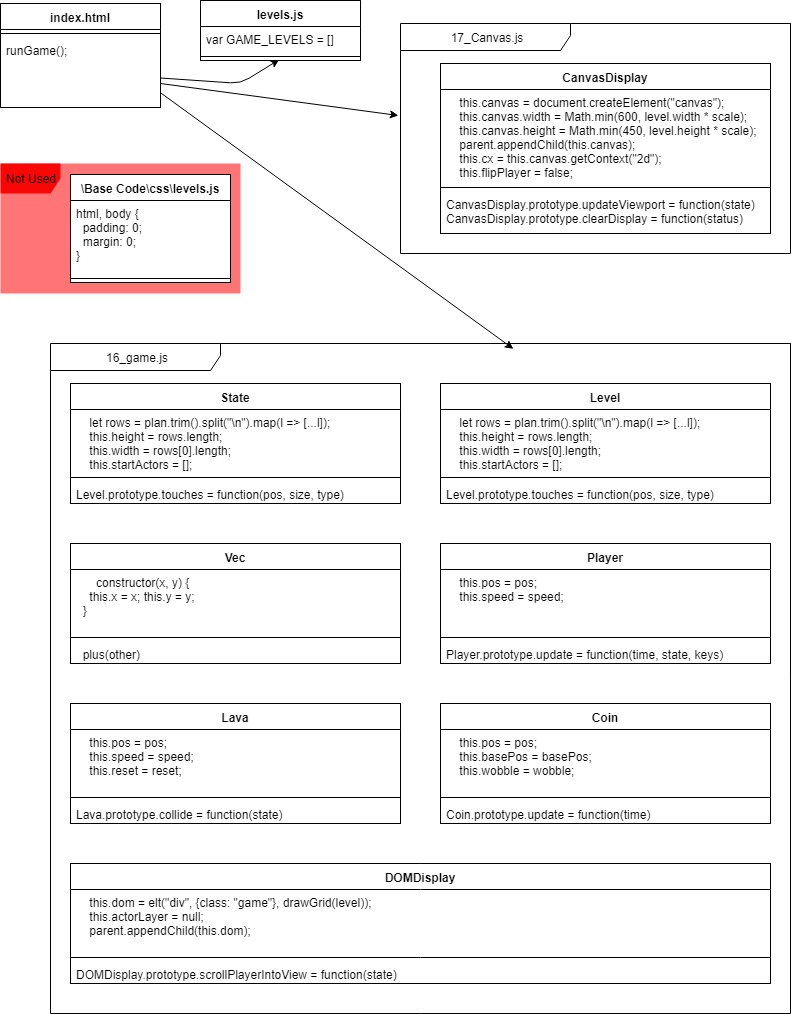
Iteration: 1

UML Diagram 1: Package Diagram of Base Code



UML Diagram 2: Class Diagram of Base Code



Iteration Work Plan:

* The Goal of The Iteration

As outlined in the “Game Outline” document, this iteration will focus on:

* + Setup of file and folder structure
  + Copy files over from “Base Code”
  + An altered avatar
  + A goal (target position, traverse a path, collect ‘stuff’, arrange/move ‘stuff’
  + Opposition (ONE OF: barriers, obstacles, dangerous objects, moving objects, following objects, traps)

Also I will be focusing on cleaning up the code structure and making the code easier to read, as well as altering the game so that it covers the entire screen when playing the game.

* The Planned Tasks in Sequence [Planning, Analysis, Design,

Coding, Testing]

* + Setup Folders and Copy Files – A
  + Code Refactor – B
  + Change Game Screen Size – C
  + Alter Avatar – D
  + Check that the Goal and Opposition segments of the game work – E
* A Time Estimate for Each Task [30 Minute Blocks]
  + A – 30 Minutes √
  + B – 60 Minutes √
  + C – 60 Minutes √
  + D – 30 Minutes √
  + E – 30 Minutes √
* The Planned ‘Product’ Of Each Task
  + File Structure Setup
  + Code is Easier to Read and in Separate Folders
  + Game Fills Entire Screen
  + Avatar is Altered
  + Everything Works
* A Record of The Actual Time Each Task Took

(3 Hours 34 Minutes)

**PLANNING A COMPLEX ALGORITHM**

**DESIGN THE ROUTINE**

CHECK PREREQUISITES

Define the problem

*Resize the game canvas so that it stretches to fit the window size. E.g big screen = big view of the level.*

*Small screen = small view of the level, the game canvas should always stretch to fit the entire screen.*

Information the routine will hide

nothing

Inputs to the routine

*State*

Outputs from the routine

Canvas Scales to Fit Screen

Pre-conditions

*canvas exists*

Post-conditions

stretching

Name the Routine

*updateCanvas*

Decide how to test the routine

see if the canvas stretches to fit the screen when the game is running

Research functionality available in standard libraries

*need to use the window object*

Think about error handling

using math.min to ensure we actually see something

Think about efficiency

Research algorithms & data types

*need to use CSS for some extra stuff*

**WRITE PSEUDOCODE**

canvas width = window width

canvas height = window height

**CODE THE ROUTINE**

CanvasDisplay.prototype.setState = function (state) {  
 this.updateCanvas(state);  
};  
CanvasDisplay.prototype.updateCanvas = function (state) {  
 this.canvas.width = window.innerWidth;  
 this.canvas.height = window.innerHeight;  
 this.viewport = {  
 left: 0,  
 top: 0,  
 width: Math.min(state.level.rows[0].length, this.canvas.width / scale),  
 height: Math.min(state.level.rows.length, this.canvas.height / scale)  
 };

<style>

html,

body {

width: 100%;

height: 100%;

margin: 0px;

overflow: hidden

}

</style>

var CanvasDisplay = class CanvasDisplay {  
 constructor(parent, level) {  
 this.canvas = document.createElement("canvas");  
 this.canvas.width = window.innerWidth;  
 this.canvas.height = window.innerHeight;  
 parent.appendChild(this.canvas);  
 this.cx = this.canvas.getContext("2d");  
  
 this.flipPlayer = false;  
  
 this.viewport = {  
 left: 0,  
 top: 0,  
 width: Math.min(level.rows[0].length, this.canvas.width / scale),  
 height: Math.min(level.rows.length, this.canvas.height / scale)  
 };  
 }  
  
 clear() {  
 this.canvas.remove();  
 }  
}

**CHECK THE CODE**

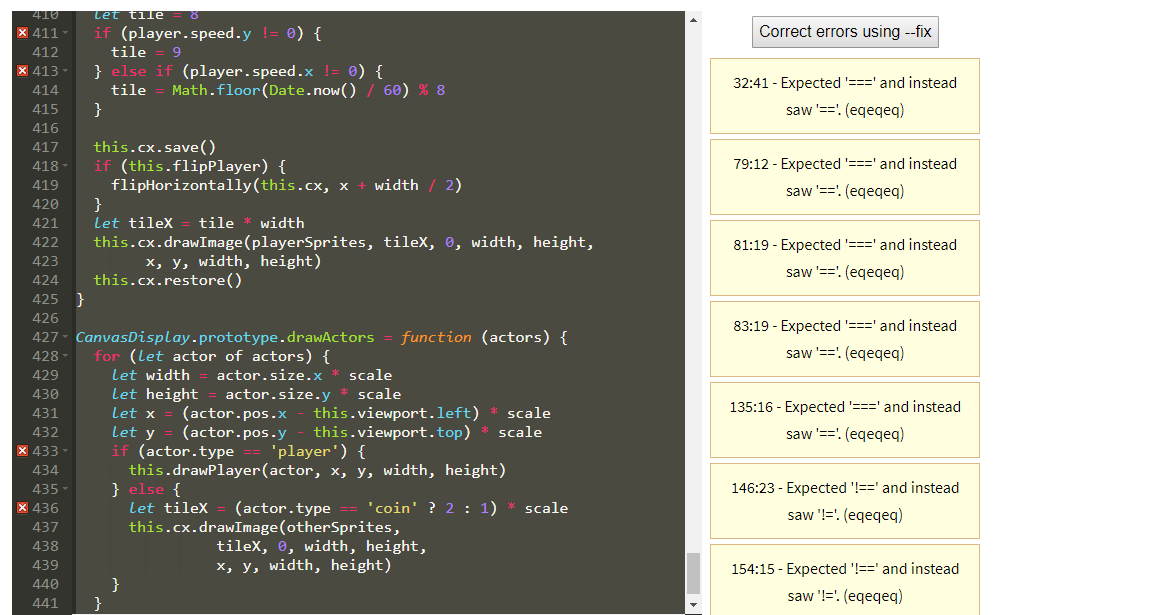
1. Mentally check for errors
2. Step through in Debugger
3. Test the code
4. Remove errors in the code
5. Clean up

Looks Good!

A plan for how the program feature you are working on will work [UML dynamic diagram, story-boards, wireframe, pseudocode]:

POST CODE COMPLETION>>>>>

A report showing nil style defects in your code according to JavaScript Standard Style https://standardjs.com/index.html:



Mistakes were made! A description and analysis of the mistakes made in the iteration:

Everything went smoothly except for a bug i encountered, it will be fixed as part of Iteration 2.

Lessons were learned? A plan for doing ONE thing differently in the next iteration:

Fix the camera jitter bug

DOM VS CANVAS

<https://www.kirupa.com/html5/dom_vs_canvas.htm>

<https://paulbakaus.com/2010/07/19/why-canvas-is-not-an-obvious-choice-for-web-games/>

canvas pros :

* could manipulate pixel and apply filter effect, so easy for image processing;
* very efficient for small size but hundreds of elements in the game
* many libraries for game could be found using canvas, such as box2dweb, and could make awesome games such as angry bird

cons:

* it's stateless, so you have to record the states of the elements in the canvas, and handle the hit test by yourself.
* low efficient for very large size but with one a few elements in the game
* great ability, great responsibility. the freedom to draw, brings in you have to charge of all the drawing staff. Fortunately, there are many libraries there, such as cocos2d-html5, IvanK.

DOM pros:

* rendering by the browser, so less error-prone;

cons:

* could do simple animation with CSS only, that makes the game not fluent;
* no good for manipulating hundreds of DOM elements;

===DOM=== ===CANVAS===

