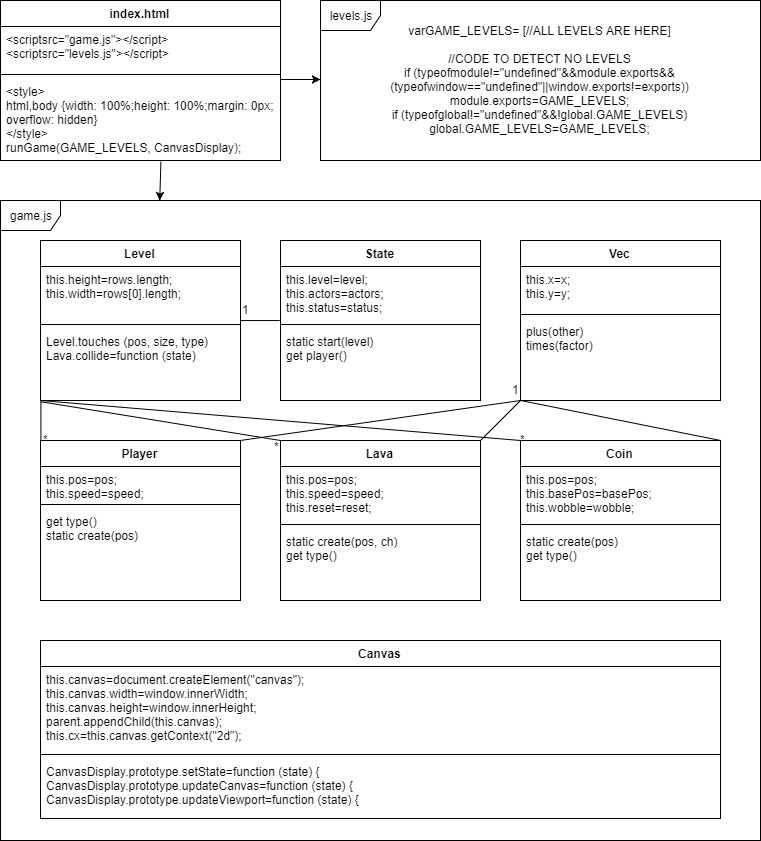
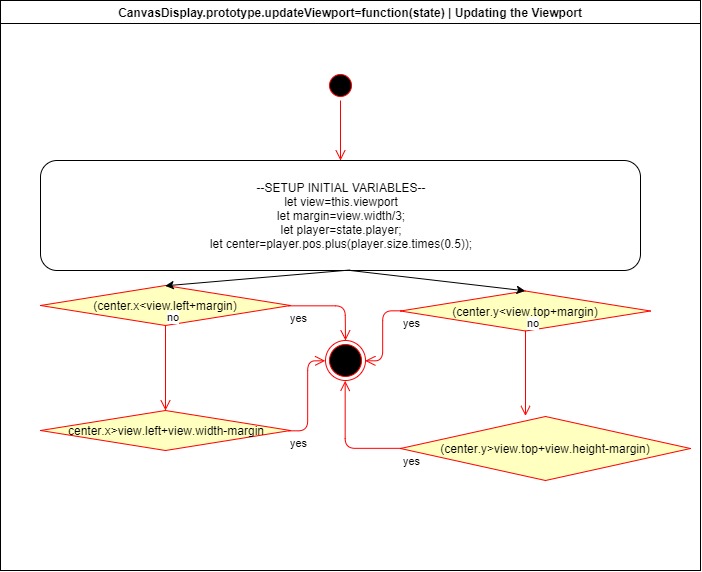
Iteration 2

UML Diagram 1: Class Diagram of Current Code



UML Diagram 2:



Iteration Work Plan:

* The Goal of The Iteration
  + 4-way movement driven by the arrow keys
  + Fits on the screen of a standard and common mobile device
  + Uses coloured blocks as images
  + Plays in less than 180 seconds
  + Not be offensive!
  + Sound

Also, I wanted to fix the camera jitter bug that was introduced in Iteration 1 as well as a new line splitting problem that was discovered when playing the game with a high zoom setting.

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

Coding, Testing]

* + Fix Camera Jitter Bug – A
  + Fix Line Splitting Bug - B
  + 4-way movement driven by the arrow keys - C
  + Fits on the screen of a standard and common mobile device - D
  + Uses coloured blocks as images - E
  + Plays in less than 180 seconds - F
  + Not be offensive! - G
  + Sound - H
* A Time Estimate for Each Task [30 Minute Blocks]
  + A – 60 Minutes
  + B – 60 Minutes
  + C – 30 Minutes
  + D – 30 Minutes
  + E – 30 Minutes
  + F – 30 Minutes
  + G – 30 Minutes
  + H – 30 Minutes TOTAL = 5 HOURS
* The Planned ‘Product’ Of Each Task
  + A – Screen doesn’t snap
  + B – Lines aren’t Split
  + C – Player Can Move
  + D – Works on Mobile
  + E – Uses Coloured Images as Blocks
  + F – Plays in less than 180 Seconds
  + G – It’s not Offensive
  + H – Sound Working
* A Record of The Actual Time Each Task Took
  + A = 1:33:00
  + B = 1:30:00
  + C = 0:10:00
  + D = 1:10:00
  + E = 0:05:00
  + F = 3:19:00
  + G = 0:01:00
  + H = 2:50:00

**PLANNING A COMPLEX ALGORITHM**

**DESIGN THE ROUTINE**

CHECK PREREQUISITES

Define the problem

*Screen is jittery, need to re-write “updateViewport” code*

Information the routine will hide

Parts of the screen will be hidden when the player zooms in far enough

Inputs to the routine

*state*

Outputs from the routine

viewport

Pre-conditions

*Game is running*

Post-conditions

Screen centers on player, or on whole level, or as far as it can reach

Name the Routine

*UpdateViewport*

Decide how to test the routine

Check to see if the screen is working

Research functionality available in standard libraries

*Need to use a clamping function*

Think about error handling

Not needed

Think about efficiency

Needs to be very efficient because this method is called each frame

Research algorithms & data types

*Need to use a clamping function*

**WRITE PSEUDOCODE**

If the screen is big enough to fit the whole game into, then do so.

Otherwise the camera should follow the player’s center, the camera should be zoomable.

If the player moves far enough to the side or top of the level, leave the camera there.

**CODE THE ROUTINE**

1. CanvasDisplay.prototype.updateViewport = function (state) {
2. let view = this.viewport
3. let player = state.player;
4. let center = player.pos.plus(player.size.times(0.5));
5. let scaledDownCanvasWidth = this.canvas.width / scale;
6. let scaledDownCanvasHeight = this.canvas.height / scale;
7. let levelWidth = state.level.width;
8. let levelHeight = state.level.height;
9. //Calculate the width and height of the viewport
10. view.width = Math.min(levelWidth, scaledDownCanvasWidth)
11. view.height = Math.min(levelHeight, scaledDownCanvasHeight)
12. //Make sure to do this V AFTER calculating the width and height of the viewport
13. //Calculate the positions of the camera that would center on the player, as well
14. //The maximum positions that the camera could be at
15. playerXMinusView = center.x - (view.width / 2)
16. lastPossibleCameraX = levelWidth - view.width
17. playerYMinusView = center.y - (view.height / 2)
18. lastPossibleCameraY = levelHeight - view.height
19. //sets the position of the camera
20. view.left = clamp(playerXMinusView, 0, lastPossibleCameraX)
21. view.top = clamp(playerYMinusView, 0, lastPossibleCameraY)
22. //Try to do the first one, if its too small do the second one, if its too big do the third one
23. function clamp(desired, min, max) {
24. if (desired < min) {
25. return min
26. }
27. else if (desired > max) {
28. return max
29. }
30. else return desired
31. }
32. };

**CHECK THE CODE**

Working!!!

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

The camera should follow the player as they move through the level without stuttering, also I want to add in blocks instead of images and add in sound, there should be a 180 second counter that restarts the game if the player takes too long

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:

Spent too much time on this iteration. Need to do more research into how the program works, need to read more eloquent JavaScript. Want to go through the code again to gain a greater understanding of it.

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

Work on jumping bug (jitters on jump) that was introduced in this iteration. Go through the code and comment everything to gain an understanding of how it works