Verlauf der technischen Umsetzung:

Gestartet haben wir unsere Umsetzung in der Technik damit, dass wir uns nochmal unser Klassendiagramm angeschauten und daraus die Oberklassen gebaut haben. So begannen wir damit, erstmal DisplayObject und InteractiveObject zu erstellen und zu testen. Bei den Tests bemerkten wir schnell, dass wir einige wichtige Variablen und Methoden in unserem ersten Klassendiagramm vergessen hatten und ergänzten diese nachträglich. So arbeiteten wir uns von oben nach unten durch unser Klassendiagramm durch, bis wir Szenen mit bewegten Objekten und Buttons erstellen konnten. Bevor wir allerdings diesen Teil fertigstellen konnten, kamen noch zwei größere Probleme auf.

Problem Globale Variablen:

Das erste war, die Bilder, Schriften oder Sounds, welche in der p5setup.js geladen wurden den anderen Dateien zur Verfügung zu stellen. Als erstes lösten wir dies über ein ENUM Objekt, welches wir an window anhängten, um es überall aufrufen zu können. Später kam dann der Tipp, dass wir dies auch über import und export lösen konnten, was bei uns aber zu zahlreichen Bugs geführt hat. Im Nachhinein könnten diese Bugs durch einen anderen verzweigteren Bug mit der Libarie p5sound zu tun gehabt haben. Auf Grund dieses Bugs, welchen wir erst kurz vor Ende mit Hilfe lösen konnten, verwendeten wir aber bis zum Schluss unser ENUM Objekt. Dieses erwies sich generell als sehr praktisch, da wir dort auch andere Konstanten, wie die ID der verschiedenen Hitboxformen speichern und unter einem Namen zugängig machen konnten. So wurde unser Code deutlich lesbarer, da nirgendwo eine Zahl übergeben werden musste, welche keine Koordinate oder Zeit war.

Problem Methoden aus anderen Objekten aufrufen:

Das zweite große Problem auf welches wir stießen war, wie wir eine Funktion von einem Übergeordneten Objekt von einem beliebig weit Untergeordneten Objekt aufrufen können. Wir stießen dabei auf einen simplen EventDispatcher, welchen wir an unser window anhängen konnten. Dieser sammelt praktisch Funktionen aus verschiedenen Objekten und speichert diese unter einem bestimmten Namen ab. Nun kann man aus egal welchem Objekt dem EventDispatcher Objekt mitteilen, dass dieser eine gewisse Funktion in einem anderen Objekt ausführen soll. Diese Lösung war auf alle Fälle die beste, welche wir gefunden haben. Die Alternative wäre gewesen, auf das X. Überobjekt zu zugreifen, was allerdings die Wiederverwendbarkeit der Klassen beeinträchtigt hätte, da diese dann immer als X. Unterobjekt des entsprechenden Überobjektes instanziiert werden müssten. Theoretisch hätten wir auch die jeweilige Funktion durch die Unterobjekte übergeben können, was aber wiederum zu noch mehr Verwirrung geführt hätte.

Szenenverwaltung:

Als wir diese zwei großen Probleme gelöst hatten, konnten wir endlich mit der eigentlichen Umsetzung beginnen. Als erstes hatten wir den Gedanken, dass jedes Spiel eine eigene Unterklasse von Game wird und die einzelnen Level dann als Szenen an diese angehängt werden. Dies erwies sich aber als unnötig, da sich unsere Level nicht so stark von einander unterschieden, um für diese wirklich einzelne Klassen zu erstellen. Der Code in den Szenen hätte sich wahrscheinlich relativ schnell gedoppelt, was wir unbedingt mit unserer View Hierachy und mit der Objektorientierung vermeiden wollten. Nach dieser Erkenntnis, kam uns aber die Idee, dass wir diese Dopplungen vermeiden können, wenn wir nur ein Game Objekt als oberstes Objekt erstellen und an dieses dann die einzelnen Simulationen und Screens als Szenen anhängen. So entstand letztendlich unsere Szenenverwaltung und jede Simulation/jeder Screen wurde als eine Unterklasse von Szene geschrieben.

Untersimulationen:

Als die Szenen Struktur nun klar war, konnte es wirklich mit unseren Untersimulationen losgehen. Als erstes programmierten wir die Untersimulation Händewaschen, da uns diese am einfachsten erschien und wir die obengenannten Konzepte erstmal an einem echten Teil unserer Simulation testen wollten. Nach dem unsere Struktur den ersten Test gut überstanden hat, setzen wir eine ~~das Grundgerüst dieser Szene stand, setzten wir die~~ ersten Versionen der Karte und des Startscreens um. Schließlich begannen wir als letztes mit People Bouncy und überarbeiteten währenddessen auch nochmal die anderen Szenen. Durch unser modulares Arbeiten konnten wir für People Bouncy viele Objekte aus Händewaschen übernehmen, wie z.B. den Mentorvirus oder den Weiterbutton. Weiterhin konnten wir dadurch auch gut an unterschiedlichen Klassen gleichzeitig arbeiten. Als letztes erstellten wir dann die Webseite und alles was um den Canvas zu sehen sein sollte. Ein kleines Problem blieb ungelöst und zwar entsteht beim überblenden zu oder nach den Videos ein kurzes flackern.