Dokumentation “Mathe Fischer”

Von

Marc Beyer

Kurzbeschreibung des Spiels

Im Spiel muss der Spieler Matheaufgabe lösen und Fische mit den korrekten Antworten fangen. Fängt er einen falschen Fisch, verliert er eins von drei Leben. Verliert er Alle, ist das Spiel vorbei.

Fängt er einen richtigen Fisch muss er möglichst viele andere Fische fangen um Bonuspunkte zu bekommen. Das macht er so lange bis er den Grund des Meeres erreicht hat, dort findet er zwei Kisten mit Bonus-Fischen, die beim öffnen das Spiel beenden.

Herausforderungen bei der Umsetzung des Projekts

Die wohl größte Herausforderung bei der Umsetzung des Spiels war es mit den Einschränkungen, die uns vorgegeben waren, zu arbeiten. Die Umsetzung mit Hilfe einer Engine / eines Frameworks wie Unity oder PhaserJS wäre natürlich viel leichter, da diese Tools auf die Spieleentwicklung ausgelegt sind und dem Entwickler stark unter die Arme greifen. Aber auch das Verbot von standard Web APIs wie WebGL und der Graph API, welche die Performance im Web stark verbessern würden war eine Herausforderung.

Um mir die Entwicklung leichter zu machen habe ich Angefangen eine sehr generelle Basis zu bauen, die stark an Unity angelegt ist. Ich habe eine Main-Game Loop (Die Update-Funktion wie in Unity) und habe alle Elemente, die visuell dargestellt werden sollen von der GameObject Klasse erben lassen, die sich um Positionierung, Rendering und Collision kümmert. Alle GameObjects werden vom GameManager kontrolliert und per Frame aufgerufen.
Ich habe einen SceneManger gebaut, der sich um das Laden und Löschen von Szenen kümmert. Einen InputHandler, der es vereinfacht die Eingaben des Users zu nutzen und eine Library Klasse, die viele nützliche Funktionen implementiert.

Ein großes Problem war die Performance. JavaScript läuft nur auf einem Thread und durch die oben genannten Einschränkungen, funktioniert die Graphische Darstellung mit DOM Elementen.
Das Erstellen und Render dieser nimmt Zeit in Anspruch und deshalb habe ich das Spiel so optimiert, dass GameObjects, die den Bildschirm verlassen (Fische und Luftblasen), nicht zerstört werden und immer wieder neue GameObjects erzeugt werden, sondern die alten recycelt werden um die Performance zu verbessern.

Viel Arbeit ist auch in die Bewegung des Angelhaken geflossen. Er soll eine Art des Widerstands im Wasser simulieren und wird nur indirekt vom Spieler gesteuert. Es hat mehrere Iterationen gebraucht, damit sich die Steuerung gut anfühlt und die Bewegungen gut aussahen.
Die Angelschnur ist ein Div, das mit CSS rotiert wird. Es ist ein Child des Angelhakens und sein Mittelpunkt ist an der Oberseite des Harkens.

Iterationen, Veränderungen und ein Pivot

Für mich ist Spieleentwicklung ein sehr iterativer Prozess. Zuerst muss die "*Core Gameplay Mechanik*" feststehen und Spaß machen, dann kann man darauf aufbauen und hoffentlich ein gutes Spiel bauen. So habe ich das auch bei diesem Spiel gemacht und habe mich stark auf das Angeln und die Fische konzentriert und immer wieder getestet und angepasst.
Am Anfang des Projekts war ein Shop und eine Oberwelt mit eines Stadt und NPCs geplant und ich habe angefangen, diese Features ein zu bauen. Ich habe aber schnell gemerkt, das es nicht viel zu Spiel beiträgt mit einem Charakter "in der Gegend" rum zu laufen und deswegen habe ich diesen ganzen Teil wieder aus dem Spiel entfernt.

Um dem Spieler einen Anreiz zu geben, das Spiel zu spielen, habe ich Leben und einen High-score eingebaut. Jetzt hat der Spieler ein Ziel und Konsequenzen wenn er etwas falsch macht.

Grafik und Sound

Ich habe mich bei diesem Spiel für Pixelart entschieden, da das eine relative zeiteffizient und ein überschaubarer Aufwand war. Durch die Änderungen am Spiel, habe ich aber viele Art-Assets wieder verworfen. Sound habe ich komplett aus dem Spiel gestrichen, da der Aufwand zu hoch ist.

Test des Spiels

Ich habe das Spiel für Firefox, Chrome Edge(Chromium based) und Opera(Chromium based). Getestet. Es gan ein paar probleme mit dem Fullscreen auf Firefox, da beim wechsel in den Fullscreen das Resize-Event nicht getriggert wird. Außerdem beim Image-rendering, da Firefox noch nicht "pixelated" unterstützt und es zeitweise einen Bug mit "crisp-edges" gab. Außerdem habe ich das Spiel auf meinem Webserver getestet und keine Fehler gefunden.

Tipps die das Testen erleichtern

In der "fishingController.js":
"maxDepth" auf einen kleineren Wert stellen (z.B auf 5), macht den Weg zu Grund des Meeres kürzer.
Im constructor *(hook, score = 0, hearts = 3, curDepth = 0, name = "FishingController")* kann man "hearts" auf einen höheren Wert stellen um mehr leben zu haben. Außerdem kann man hier die Start-Tiefe festlegen (curDepth) und den Start-Score (score).

Quellen

Nachschlagewerke und JavaScript-, CSS-, HTML-Dokumentationen

MDN Dokumentation (https://developer.mozilla.org/en-US/docs/Web),
W3Schools (https://www.w3schools.com/)

Browser Support

CanIUse (https://caniuse.com/)

Assets

Arvo Font (Copyright (c) 2010-2013, Anton Koovit (anton@korkork.com), with Reserved Font Name 'Arvo')