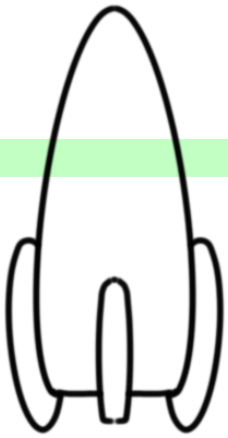
Not Rocket Science



Kreativkonzeption

Sommersemester 2021

Hochschule Furtwangen

vorgelegt von:

Lucia Rothweiler, 263812, MKB4

Alida Kohler, 263819, MKB4

Inhalt

01 Pitch

Ziel des Kurses ist es einen Prototypen einer Anwendung zu konzipieren und zu entwickeln. Folgendes Thema war dabei zu berücksichtigen:

Formen der Kommunikation zwischen Menschen aller Generationen im Kontext der räumlichen Distanzierung

Neben einem aussagekräftigen Konzept sollten wir auch mittels HTML, CSS und TypeScript einen Prototypen coden. Die Herausforderung bestand hierbei darin, dass mehrere Nutzer gleichzeitig Zugriff auf die Anwendung haben und miteinander interagieren können sollten. Daher war es erforderlich, einen Server in den Prototypen zu integrieren.

02 Ideenfindung

Durch die relativ offene Aufgabenstellung hatten wir die Freiheit, ein Thema auszuwählen, das uns beiden Spaß macht. Die Wahl fiel schnell auf ein Spiel, da uns beide die Entwicklung von ebendiesen sehr interessiert. Da es um Kommunikation zwischen Menschen geht, war klar, dass es ein Multiplayer Spiel werden sollte.

Uns gefiel auch der in den Vorlesungen vorgestellte Ansatz, Elemente innerhalb des HTML mit den Bewegungssensoren des Handys zu steuern. Daher wollten wir diesen in unsere Anwendung ebenfalls integrieren.

Nach einigem Überlegen kamen wir auf das Thema Weltall. Wir entschieden uns für ein Spiel, in dem zwei Spieler gemeinsam eine Rakete durch den Weltraum steuern müssen. Dabei stoßen die beiden Astronauten auf Gefahren und Hindernisse, die die Rakete beschädigen können.

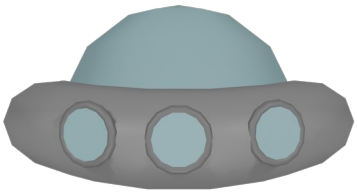
Ein Spieler steuert die Rakete, mit Hilfe der Bewegungssensoren des Handys nach links und rechts. Während der andere Spieler durch Tippen auf den Bildschirm UFOs abschießt.

03 Entwurfsphase

03.01 Allgemein

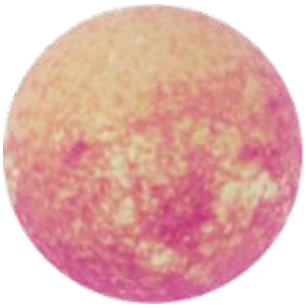
Für das Design des Spiels standen verschiedene Artstyles im Raum. Unsere Wahl fiel auf einen minimalistisch-realistischen Stil. Die Assets wurden in Blender modelliert, gerendert und als PNG in das HTML eingefügt.

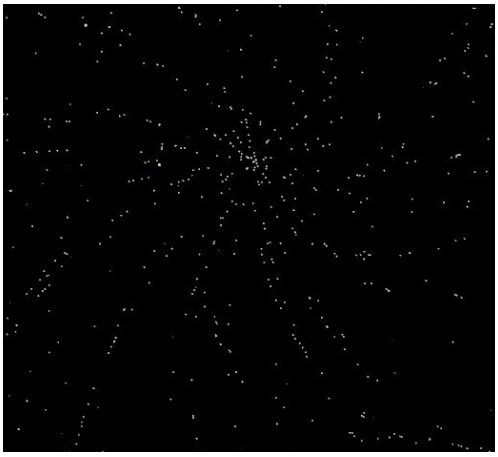
Die Rakete wurde in drei Zuständen modelliert, um den Spielern zu zeigen, ob und wie stark diese beschädigt ist.



In einem ähnlichen Stil modellierten wir ein kleines UFO. Dieses basiert auf der stereotypischen Darstellung.

Die Planeten stellen mit ihren kräftigen, bunten Farben einen Kontrast zu der eher monochromen Farbpalette der übrigen Assets dar.





Als Hintergrund entschieden wir uns für eine einfache, schwarz-weiße Galaxie, die sich nahtlos aneinander fügt. Diese wird als einziges Asset nicht mit TypeScript, sondern mit CSS animiert.

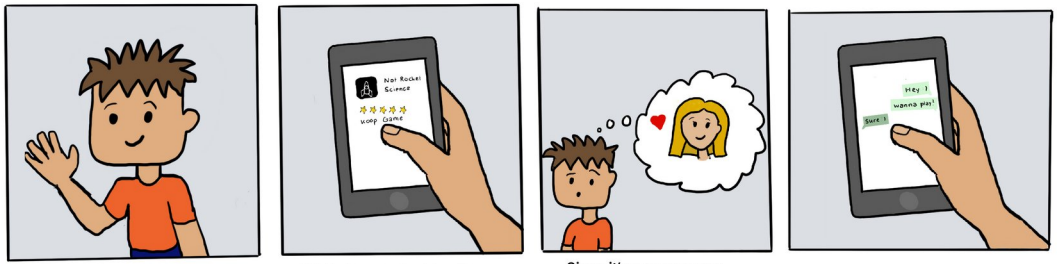
Für die Überschriften entschieden wir uns für die modern-futuristisch anmutende Schrift Arual. Diese haben wir von der Seite „dafont“

Lorem Ipsum Dolor sit amet

Beim normalen Fließtext verwenden wir die Schrift „Roboto Thin“

Lorem Ipsum Dolor sit amet

03.02 Storyboard

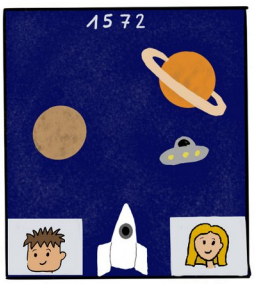


This is Tim. He likes to play games on his phone.

While he was looking for new games he found „Not Rocket Science“

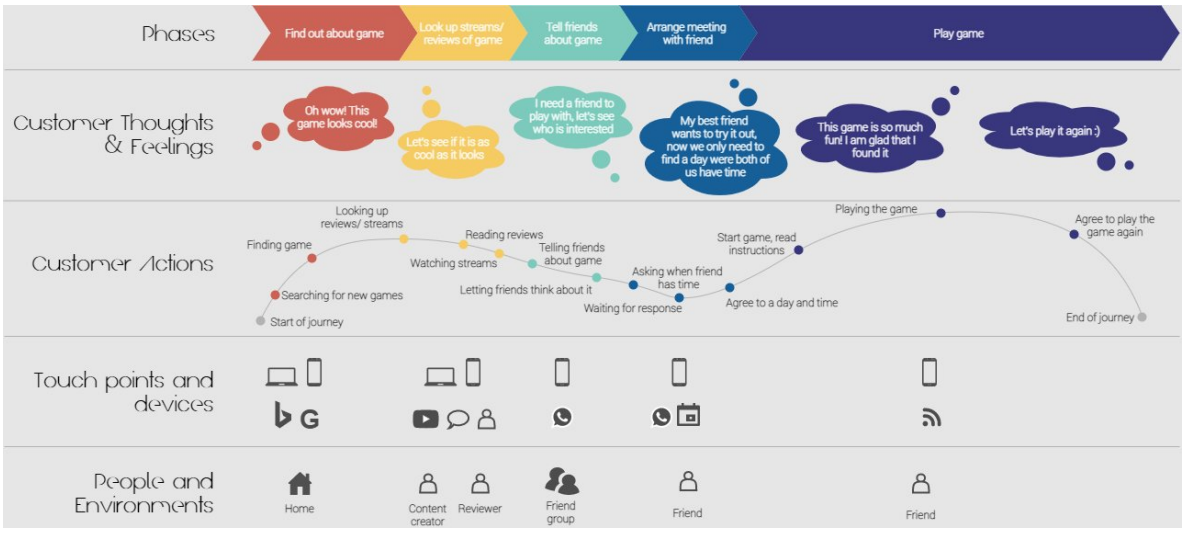
Since it's a coop game he wants to play it with his friend Emma.

He asks her if she wants to play with him.



They play together and try to beat the highscore.

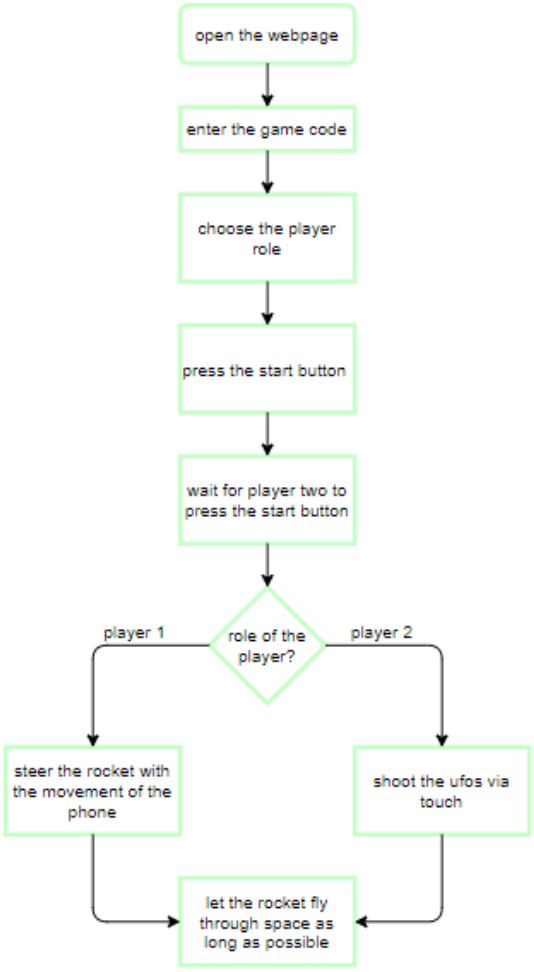
03.03 Customer Journey



03.04 Spielablauf als Flowchart

Flowchart Player 1

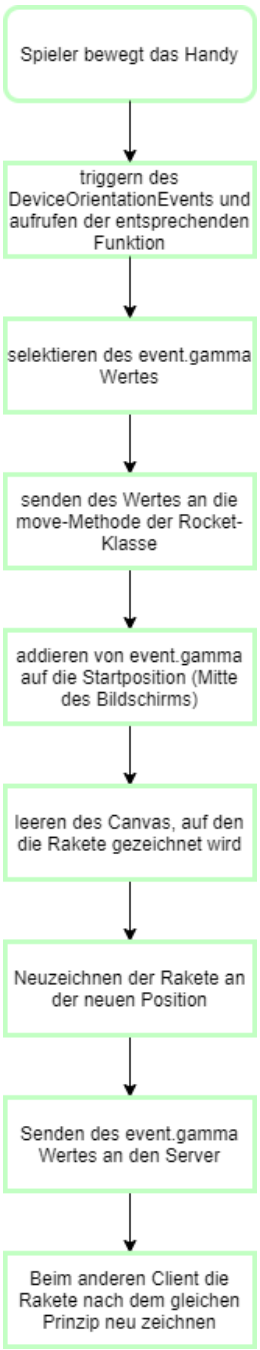
Flowchart Player 2



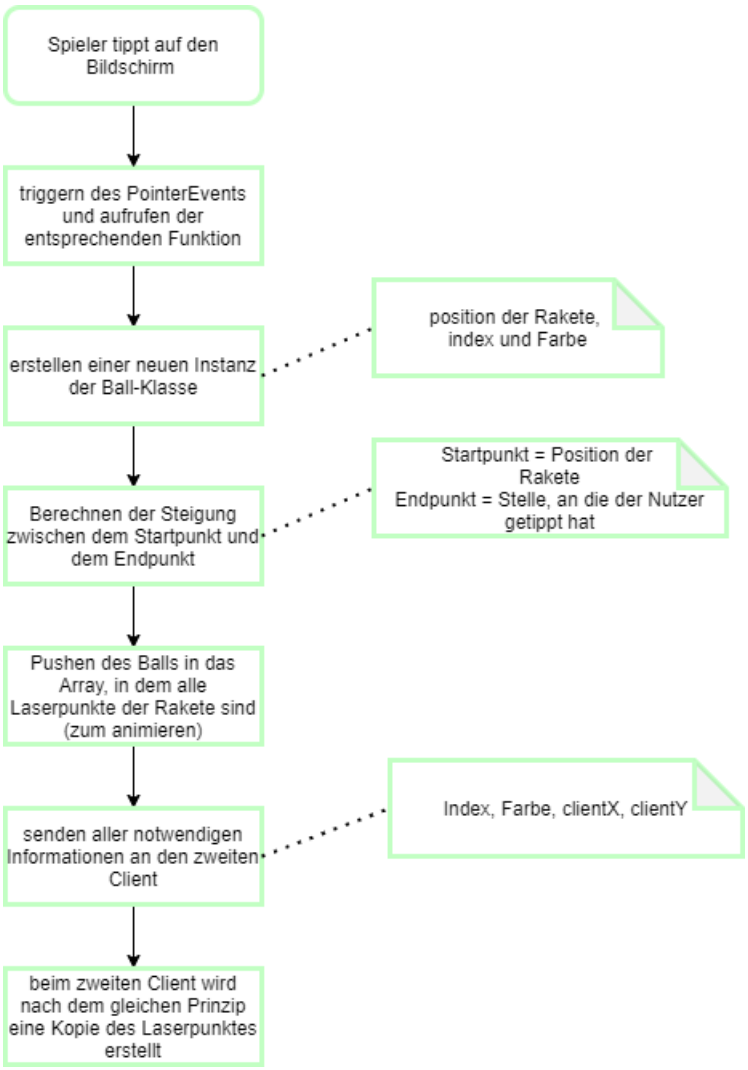
03.05 Startscreens

03.06 Flowcharts für den Code der Grundfunktionen

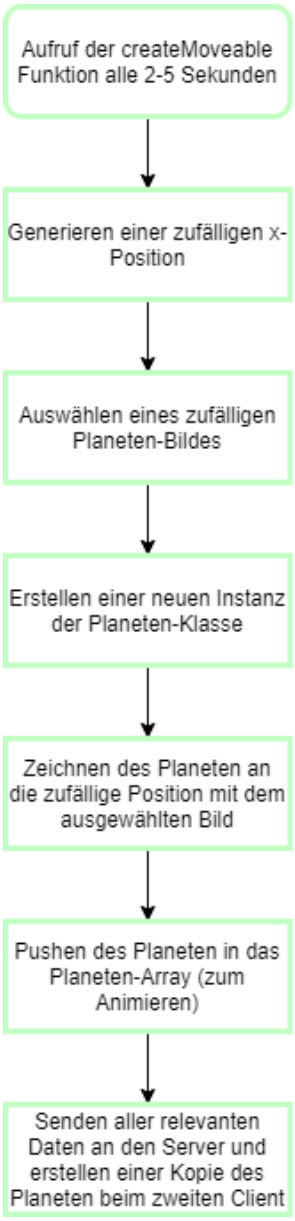
Flowchart Bewegungs-steuerung

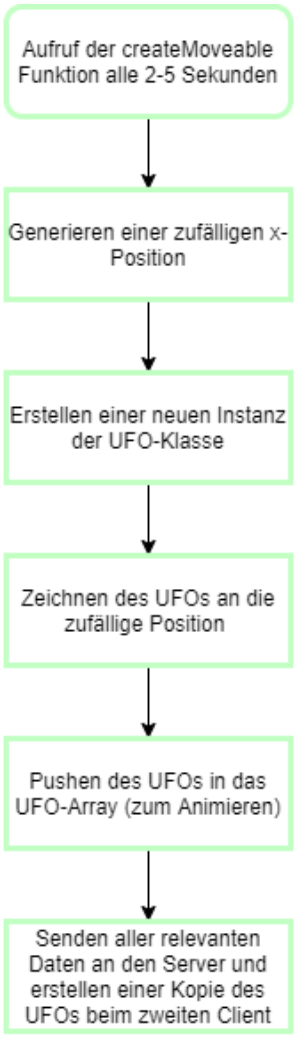


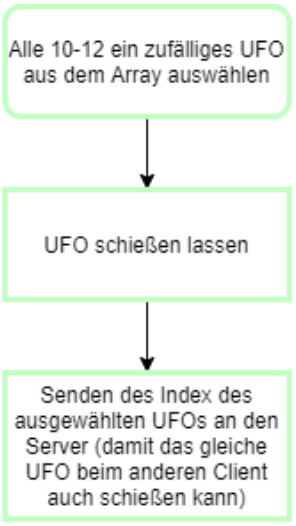
Flowchart Touchevent



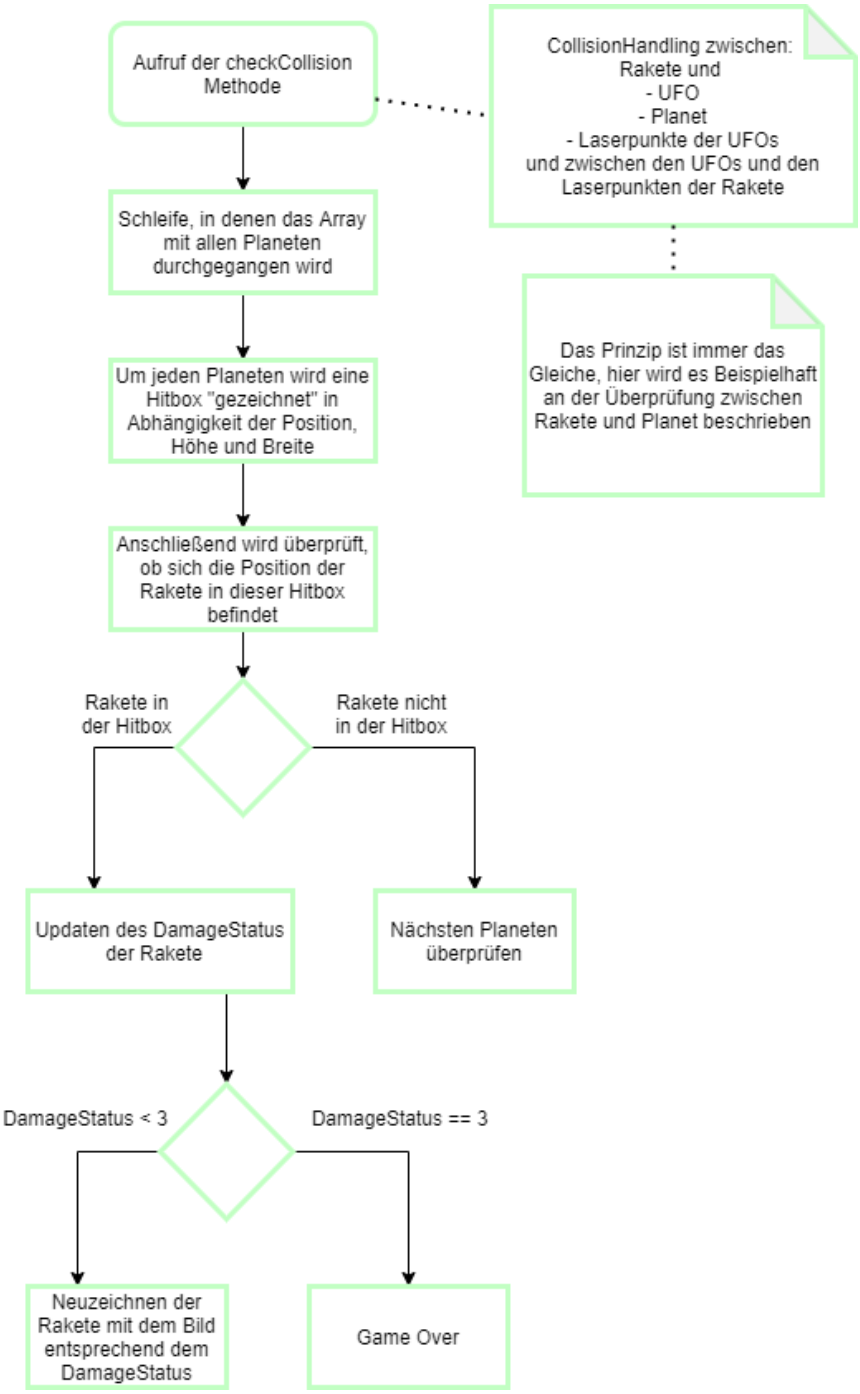
Flowcharts Generierung von Planeten, UFOs und UFO-Laserpunkt







Flowchart Collisionhandling



03.07 Klassendiagramm

ball
+ positionX: number + positionY: number + distance: number = 0 + angle: number = 0 + speed: number = 2 + velocityX: number + velocityY: number + didDamage: boolean = false + index: number + color: string
constructor: (_positionX: number, _positionY: number, _index: number, _color: string) getElevation (_endX: number, _endY: number) move() draw()

ufo
+ positionX: number + positionY: number + sizeX: number + sizeY: number + damage: number = 0 + image: HTMLImageElement + didDamage: boolean + index: number
constructor(_x: number, _y: number, _sizeX: number, _sizeY: number, _image: HTMLImageElement, _index: number) shoot(_directionX?: number, _directionY?: number) draw(_ctx: CanvasRenderingContext2D) move (_add: number) checkCollision()

planet
+ posX: number + posY: number + image: HTMLImageElement + size: number + didDamage: boolean = false + index: number + type: string
constructor (_x: number, _y: number, _image: HTMLImageElement, _size: number, _index: number, _type: string) move(_add: number) draw(_ctx: CanvasRenderingContext2D)

rocket
+ startPosX: number + startPosY: number + newPos: number + image: HTMLImageElement + imageDamageOne: HTMLImageElement + imageDamageTwo: HTMLImageElement + sizeX: number: 50 + sizeY: number = 100
constructor (_startPosX: number, _startPosY: number, _image: HTMLImageElement, _imageDamageOne: HTMLImageElement, _imageDamageTwo: HTMLImageElement) move() drawRocket() damageUpdate() checkCollision()

04 Prototypen

Ausgehend von den oben genannten Arbeitspaketen/ Flowcharts haben wir verschiedene Prototypen erstellt. Ziel war es, mit jedem Prototyp eine Kernfunktionalität zu coden, um diese am Ende zusammenfügen zu können.

04.01 Bewegungssteuerung

Für den ersten Prototypen widmeten wir uns der Bewegungssteuerung. Von Relevanz war für uns die Rotation um die y-Achse, also die Bewegung nach rechts und links. Diesen Wert erhalten wir durch die Eigenschaft „gamma" des DeviceOrientationEvents. Auf dem Canvas befindet sich lediglich ein grünes Quadrat, welches in Abhängigkeit von Höhe und Breite des Bildschirms platziert wird. Bei Bewegung wird event.gamma zu der Startposition addiert.

04.02 Laserpunkte und PointerEvents

Nach unserem ersten Konzept verfügte die Rakete über ein Kanonenrohr, aus welchem die Laserpunkte abgefeuert wurden. Dieses wurde durch die Bewegung vom Handy ausgerichtet,

anschließend konnten per Touch Laserpunkte abgefeuert werden. Gegen Ende des Projektes stellten wir jedoch fest, dass das Kanonenrohr weder für das Design noch für die Funktionalität einen Mehrwert für unsere Anwendung brachte.

Der wichtigste Teil dieses Prototyps war die Berechnung der Flugbahn des Laserpunktes. Gegeben waren zwei Punkte: Das Ende des Kanonenrohrs in Abhängigkeit der Rotation und die Stelle, an die der Nutzer getippt hat. Letztere erhielten wir durch die Eigenschaften „clientX" und „clientY" des PointerEvents. Durch die Berechnung erhalten wir einen x- und einen y-Wert, den wir dann für die Animation alle 40 Millisekunden auf die Position addieren.

04.03 Generieren von Objekten und Animieren von Bildern auf Canvas

Zu Beginn des Projektes war unklar, ob und wie wir Bilder auf dem HTML-Canvas darstellen können. Nach kurzer Recherche fanden wir heraus, dass wir das Bild als img im HTML einbinden müssen. Anschließend wird das Bild in der TS-Datei selektiert und mit der Methode „drawImage" des CanvasRenderingContext2D in der entsprechenden Größe und Skalierung auf den Canvas gemalt.

Im zweiten Schritt entwickelten wir mit setInterval einen einfachen Algorithmus, der alle 2-5 Sekunden einen Planeten an einer zufälligen x-Position generiert.

04.04 Objektorientierung

Nach den ersten Grundfunktionen erstellten wir Klassen für die Rakete, Planeten, UFOs, Laserpunkte und das Kanonenrohr. Für den Prototypen vier haben wir uns zu viel vorgenommen und wir stießen auf einige Probleme. Dinge, die in den vorherigen Prototypen funktionierten, hatten auf einmal Bugs. Wir entschlossen uns dazu, diesen Prototypen nicht fertigzustellen und stattdessen die Prototypen fünf und sechs zu entwickeln, in denen wir den Code kleinschrittiger schrieben. Am Ende hatten wir fünf funktionsfähige Klassen mit allen notwendigen Eigenschaften und Methoden (siehe Klassendiagramm).

04.05 Collisionhandling

Nachdem der Code nun in Klassen getrennt war, konnten wir uns mit dem Collisionhandling der einzelnen Objekte befassen. Es gibt zwei Klassen, in denen überprüft werden muss, ob es eine Kollision gab: die der Rakete und die der UFOs.

Die Rakete kann mit Planeten, UFOs und den Laserpunkten der UFOs kollidieren, die UFOs mit den Laserpunkten der Rakete. Für jede Überprüfung wurde um eins der Objekte eine Hitbox „gezeichnet" und anschließend überprüft, ob sich die Position des anderen Objekts in dieser Hitbox befindet.

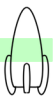
04.06 Server

Unseren Server erstellten wir mit Hilfe von WebSockets, da sich die Standard-Variante mit http/ https für unsere Anwendung nicht eignete. Für unser Projekt ist es entscheidend, dass die Updates innerhalb weniger Millisekunden von einem Client zum anderen kommen. Um die Programmierung von Websockets zu testen, fertigten wir einen kleinen Prototypen mit zwei Clients an. Client 1 basierte auf unserem ersten Prototypen mit der grünen Box. Nur wurde diesmal nicht nur die Box neu gezeichnet, sondern der event.gamma-Wert auch an den Server übermittelt. Ein zweiter Client empfing die neue Position und zeichnete die grüne Box ebenfalls neu, er spiegelte die Bewegungen des ersten Clients wider.

04.07 Kombination aller Prototypen

Nachdem wir erste Erfahrungen mit dem Server gesammelt hatten, machten wir uns daran, die Prototypen zu kombinieren. Wie bereits oben beschrieben, haben die beiden Clients unterschiedliche Aufgaben, daher unterscheiden sich auch die jeweiligen Scripte. Hier konnten wir zum ersten Mal sehen, ob die Schnittstellen für die beiden Clients so funktionieren wie geplant.

Client 1 generiert alle 2-5 Sekunden einen neuen Planeten und registriert die Bewegungen der Rakete. Sowohl die neue Position der Rakete als auch alle relevanten Informationen für den Planeten werden an den Server geschickt.



Client 2 generiert alle 2-5 Sekunden ein neues UFO, lässt alle 10-12 Sekunden ein UFO schießen und kreiert einen neuen Laserpunkt der Rakete, wenn der Spieler auf den Bildschirm tippt. Auch hier werden alle relevanten Informationen an den Server gesendet.

05 Ergebnisse und Fazit

Im Laufe des Semesters entwickelten wir einen Prototypen, der die Grundfunktionen unserer Idee abbildet. Wir hatten die Gelegenheit, unsere Fähigkeiten sowohl in der Konzeption, als auch in der Programmierung zu erweitern und zu vertiefen. Gerade bei der Serverseitigen Programmierung konnten wir viel lernen. Für uns war es eins der ersten größeren Projekte, das auch über einen längeren Zeitraum ging Wir haben uns das Semester über weitgehend selbst organisiert, hier konnten wir auch viel zum Thema Zeitplanung und -managment lernen. Einige der gelernten Fähigkeiten konnten wir auch direkt im Projektstudium anwenden.

06 Ausblick

Als Erweiterungen für unsere Anwendung hatten wir folgende Überlegungen:

- Punktesystem → Highscore
- Freischaltbare, verschiedene Designs
- Join Random Game Funktion
- Einzelspielermodus
- Verschiedene Schwierigkeitsgrade
- Voicechat
- Spielmodus mit drei Spielern
- Collectables (Sternschnuppen → reparieren die Rakete / bringen Extrapunkte)

07 Entwürfe, Prototypen

Github-Repository: https://github.com/KohlerAl/Lucida

Github-Repository (alle Prototypen) auf Github Pages:
https://kohleral.github.io/Lucida/start.html

Link zum finalen Prototypen:

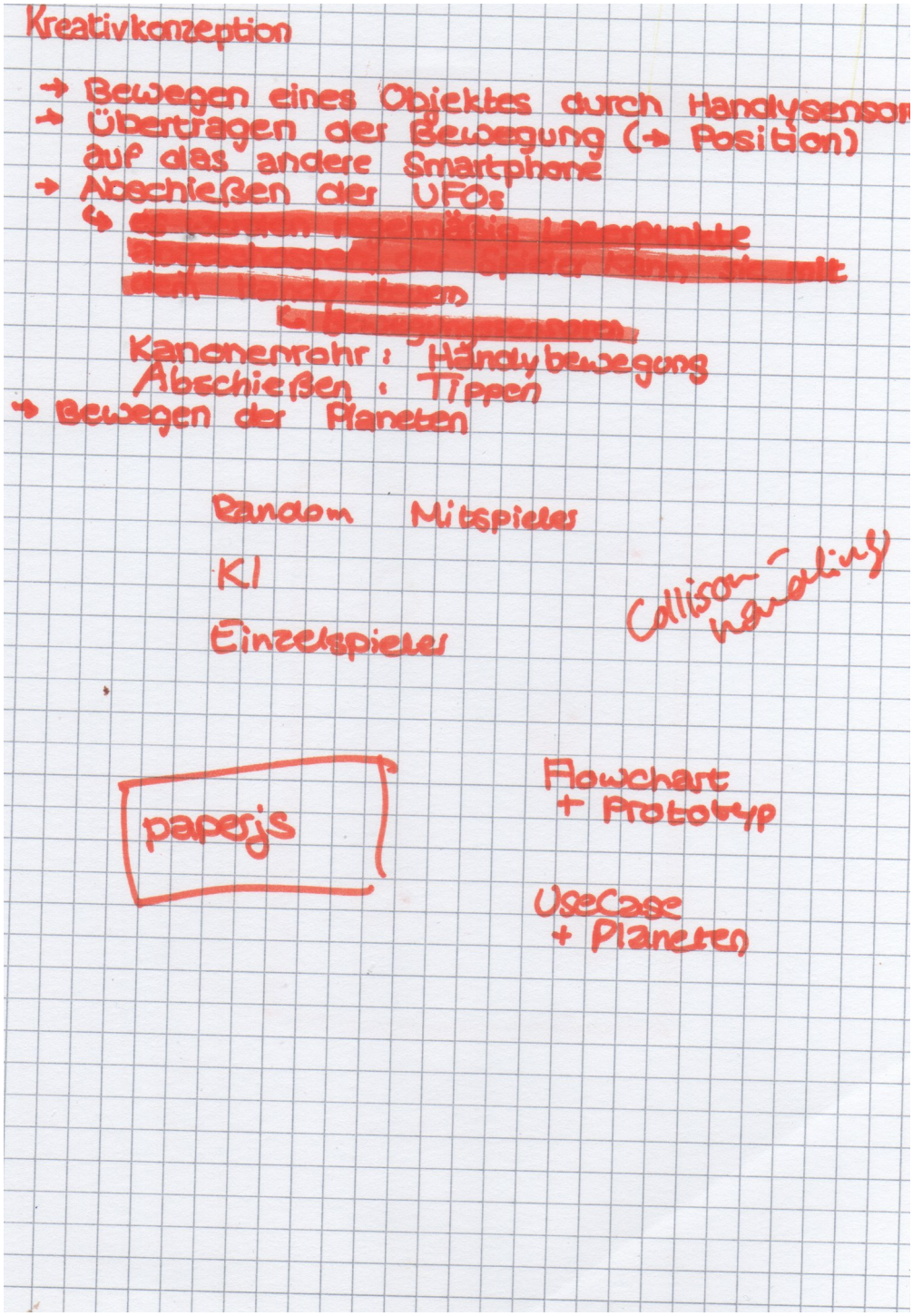
! Hinweis : Am besten funktioniert die Anwendung, wenn beide Prototypen-Seiten in einem neuen Tab aufgerufen werden. Erst wenn beide Seiten geladen sind, auf Start drücken !

https://kohleral.github.io/Lucida/html/prototype10_one.html

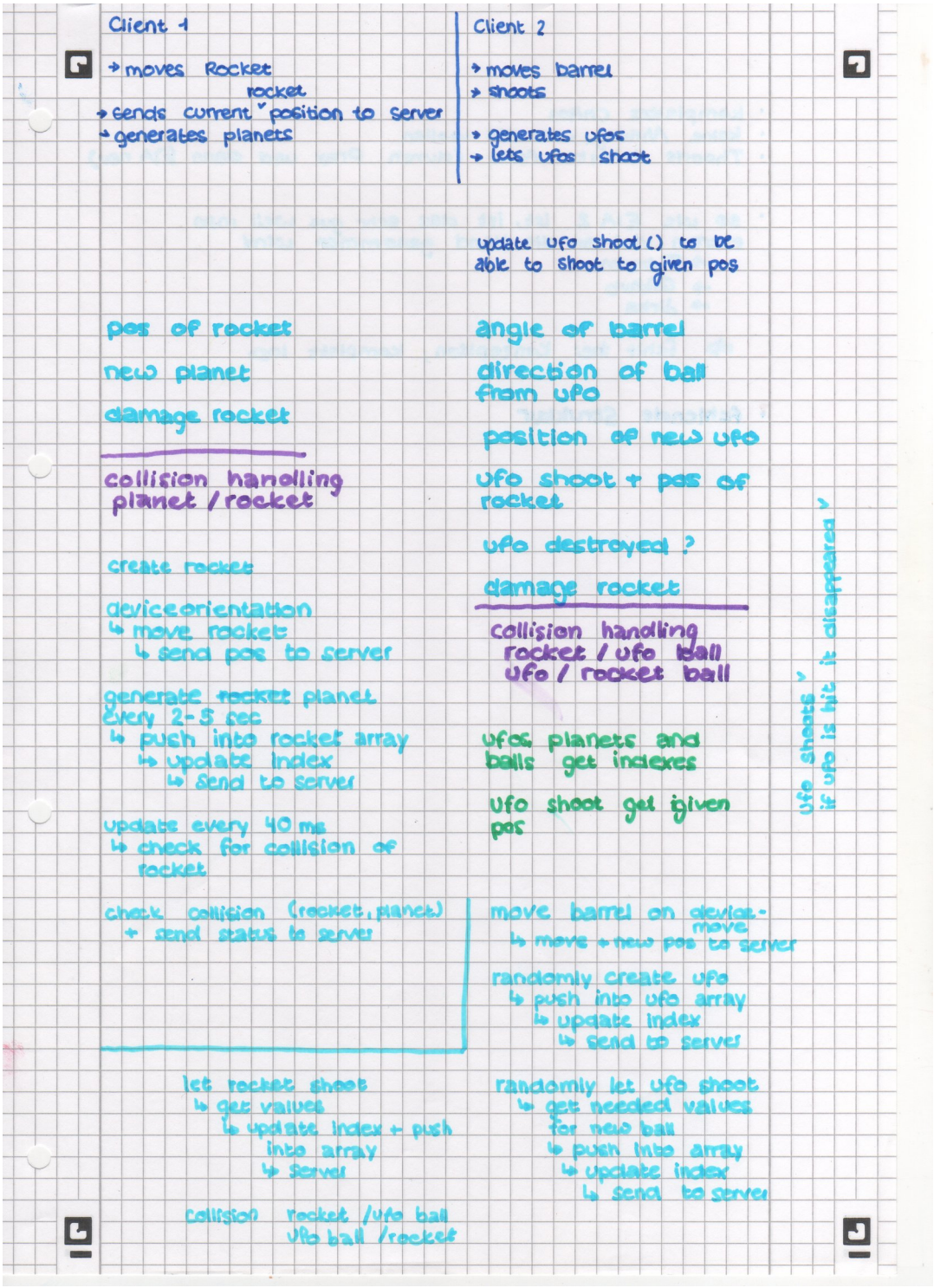
https://kohleral.github.io/Lucida/html/prototype10_two.html

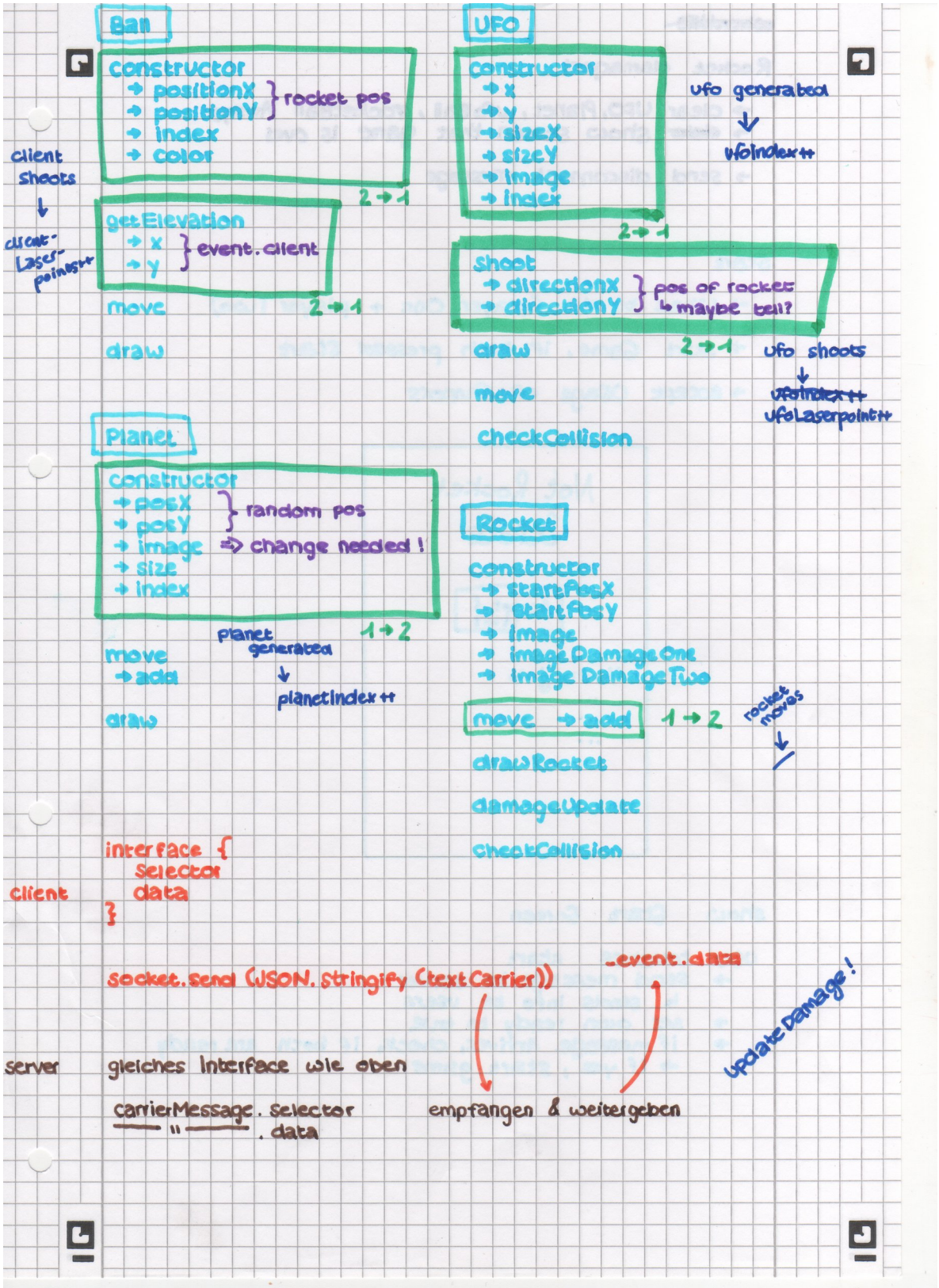
08 Anhang

Handschriftliche Notizen zur Idee und Arbeitspaketen zu Beginn des Semesters



Handschriftliche Notizen vor und während der Entwicklung des 10. Prototypen




Ban
constructor
→ positionX
→ positionY
} rocket pos
→ index
→ color
2 → 1
client shoots
↓
client-LaserPoints++
getElevation
→ x
→ y
} event.client
2 → 1
move
draw
UFO
constructor
→ x
→ y
→ sizeX
→ sizeY
→ image
→ index
2 → 1
ufo generated
↓
ufoIndex++
shoot
→ directionX
→ directionY
} pos of rocket
↳ maybe bell?
2 → 1
draw
move
checkCollision
ufo shoots
↓
ufoIndex++
ufoLaserpoint++
Planet
constructor
→ posX
→ posY
→ image
→ size
→ index
} random pos
⇒ change needed!
1 → 2
planet generated
↓
planetIndex++
move
→ add
draw
Rocket
constructor
→ startPosX
→ startPosY
→ image
→ imageDamageOne
→ imageDamageTwo
move → add
1 → 2
rocket moves
↓
drawRocket
damageUpdate
checkCollision
interface {
selector
data
}
client
socket.send (JSON.stringify (textCarrier))
-event.data
server
gleiches Interface wie oben
carrierMessage. selector
" . data
empfangen & weitergeben
Update Damage!

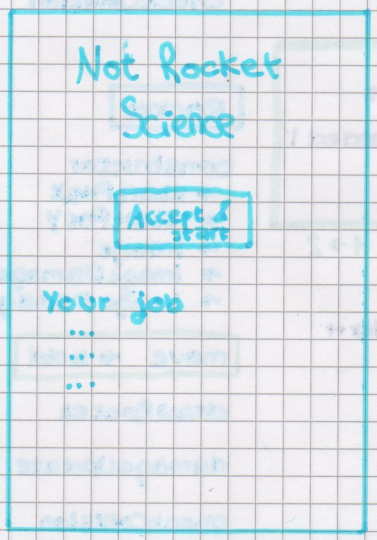
~~removeUFO~~

Rocket damaged

→ clear UFO, Planet, ufoBall, rocketBall Arrays
→ ~~delete~~ show screen that game is over
→ send disconnect message

Start

→ what to do (player One + player Two)
→ start Game, if both pressed Start
→ accept Usage of Buttons



show Start Screen

user touches start
→ send message to server
↳ sends info to users
→ set own ready to true
→ if message arrives, check, if both are ready
→ if yes, start game