Runner's Rampage

„ROCKET LAWNCHAIR!“ (Metal Slug 2)

Inhaltsverzeichnis

- Inhaltsverzeichnis
- Das Team
- Kurzbeschreibung
- Walkthrough-Video
- Das Spielprinzip
- Download
- Screenshots
- Erste Ideen & Konzept
- Umsetzung
  - Erstellung von Assets
  - Multiplayer
  - Erstes Level
  - Zweites Level
- Devblog
- Herausforderungen
  - Online-Multiplayer

    - Unity Collaborate & Plastic SCM
    - Movement, Physics & Kollisionen
    - Projektumfang/Scope
- Ziele
- Zeitaufwand/Stundenrechnung
- Lessons Learned
- Tools

Das Team

Teammitglied	Matrikelnummer	Aufgabenbereich
Yannic Kühn	2382131	Konzept, Coding
Tobias Rawald	2380225	Konzept, Coding, Art Design, 3D Modeling
Justus Boos	2388000	Konzept, Coding, Multiplayer
Tom Kalberg	2382485	Konzept, Level-, Art & Sound-Design, Dokumentation

---

Kurzbeschreibung

Runner's Rampage ist ein Partyspiel, bei dem sich 2-4 Spieler in einem Arena-ähnlichen Level mit Raketenwerfern bekämpfen. Last Man Standing - der letzte Überlebende gewinnt!

Walkthrough-Video

Das Spielprinzip

Nach dem Spielstart hat der Spieler im Hauptmenü/Startscreen die Möglichkeit, seinen Spielernamen zu ändern, einen öffentlichen oder privaten Raum zu erstellen, bereits erstellte Räume zu suchen oder einem Raum beizutreten. Sobald man sich in einem Raum befindet, hat man die Auswahl zwischen vier Charakteren: Blau, Grün, Schwarz und Rot. Wenn alle Spieler ihre Charakter ausgewählt haben, wird das Spiel gestartet und alle Spieler werden in dem Arenalevel gespawnt.

Nun gilt es, schnell zu reagieren! In der Arena sind mehrere Item-Pickups verteilt, hinter denen sich ein Raketenwerfer verbirgt. Mit Hilfe des Raketenwerfers kann man andere Spieler schnell und effektiv aus dem Level befördern. Die Spieler haben außerdem die Möglichkeit, sich mit Hilfe von Jumps, Dashes und Slides aus dem Staub zu machen und den Raketen auszuweichen. Wer den Raketenwerfer einsammelt, hat daher nur theoretisch die Oberhand - denn auch dieser spawnt nach einer gewissen Zeit erneut. Jeder Spieler hat außerdem drei Lebenspunkte in Form von Herzen - man verliert einen Lebenspunkt, wenn man aus der Arena fällt. Ein chaotisches Katz- und Maus-Spiel, bei dem nur der letzte Überlebende gewinnt!

Download

Ein erster Alpha-Build des Spiels lässt sich unter folgendem Link herunterladen: KLICK

Screenshots

Erste Ideen & Konzept

Beim ersten Brainstorming kristallisierte sich schnell heraus, dass wir alle gerne ein Spiel mit **Online-Multiplayer** kreieren möchten, da wir zum einen bisher nur wenig Erfahrung mit diesem Thema und den dazugehörigen Komponenten wie etwa Netcode, Serverinfrastruktur oder API sammeln konnten, zum anderen macht die aktuelle COVID-19-Situation es nicht gerade einfach, ein Spiel mit lokalem Multiplayer effektiv gemeinsam zu testen. Außerdem sind Offline-Multiplayerspiele eine aussterbene Art - heutzutage ein Multiplayerspiel ohne Onlinekomponente zu veröffentlichen, ist weder zeitgemäß noch besonders klug, gerade wenn man sein Produkt am Ende vielleicht sogar einmal vermarkten möchte.

Nach dem die Genre-Ideen, die von jedem aus der Gruppe in die Runde geworfen wurden, in mehreren (teilweise hitzigen) Diskussion abgewogen wurden, kamen wir letztendlich auf einen ersten gemeinsamen Nenner: Das Spiel sollte am Ende ein **Jump n' Run Partyspiel** werden, bei dem mehrere Spieler um die Wette laufen und sich währenddessen mit **Items bzw. Powerups** bekämpfen. Wer zuerst das Ziel erreicht oder als letzter Spieler überlebt, gewinnt die Runde. Bei der Konzeptionierung haben wir uns hierbei von Spieletiteln wie z.B. **Mario Kart**, **Mashed** und vor allem **Speedrunners** inspirieren lassen.

Bei den Überlegungen bezüglich des Game Designs stand zuerst die Idee im Raum, das Spiel auf einer 2D-Basis zu erstellen. Somit könne man sich beim Design einer der drei Achsen gänzlich sparen und wäre bei der Erstellung von Assets auf Sprites statt Polygone angewiesen. Unity hat hierfür sogar einen eigenen 2D-Modus, der das Programmieren auf zweidimensionaler Ebene vereinfacht. Diese Idee haben wir jedoch letzendlich verworfen, da wir unsere bereits gesammelten Erfahrungen mit Unity 3D und der Erstellung und Animation von 3D-Assets mit Blender anwenden möchten. Die Entscheidung fiel also auf 3D-Grafik mit Low-Poly Assets, um den Gesamtaufwand im Rahmen zu halten und Teile der Assets selber erstellen zu können, ohne zu stark auf Assetpakete angewiesen zu sein.

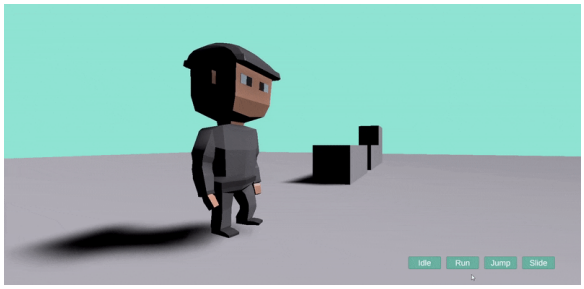
Moonraker
Hothead
Gil
Unic
FP
Unic
Moonraker

Umsetzung

Das Projekt bzw. Spiel wurde mit Hilfe von Unity erstellt - eine genaue Auflistung aller benutzten Tools und Assets findet sich weiter unten. Um den Leser nicht mit einer kompletten Schritt-für-Schritt Dokumentation unserer Arbeit an dem Projekt zu langweiligen, werden im folgenden einige Teilbereiche der Umsetzung, die wir für nennenswert halten, dargestellt bzw. erläutert.

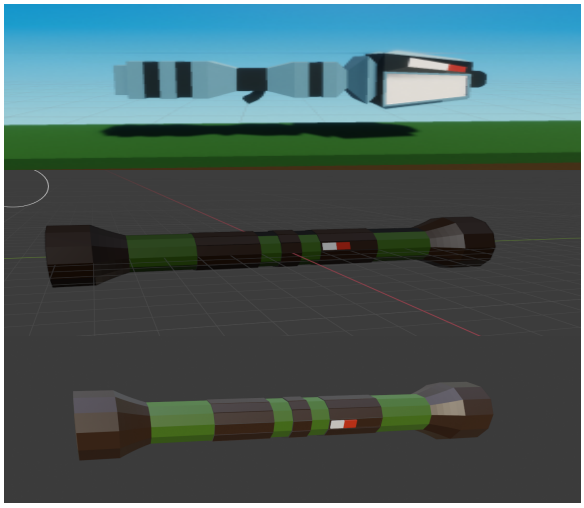
Erstellung von Assets

Wie in unserem Konzept bereits erläutert, war es uns wichtig, unser Spiel auf 3D-Basis zu erstellen, um die Möglichkeit zu haben, selber mit Blender Assets zu erstellen. Es wurden daher u.a. das **Playermodel**, der **Raketenwerfer inkl. Rakete**, die **Herzen für die Lebensanzeige** und **Teile der Leveldesigns** selber in Blender erstellt. Für das Playermodel wurden außerdem die benötigen Animation für das Laufen, Springen, Dashen und Sliden angefertigt.



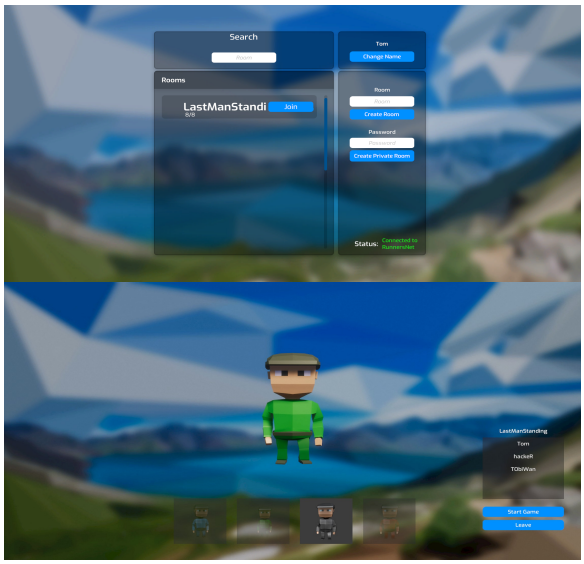
Devblo

- [x] Projekt in Unity erstellt und mit Collaborate für alle Gruppenmitglieder verfügbar gemacht
- [x] Basic Movement mit RigidBody funktioniert
- [x] Experimente RigidBody vs. CharacterController
- [x] Rudimentäres Testlevel zum Testing des Movements erstellt
- [x] Recherche und Tests mit verschiedenen Spiral Meshes
- [x] Erstes Low-Poly Charaktermodell in Blender erstellt und nach Unity exportiert
- [x] Erste Animationen (laufen, springen, rutschen) in Blender erstellt und nach Unity exportiert
- [x] MLAPI (Unity Multiplaye API) in Unity Projekt integriert
- [x] Dedicated Server Lösung getestet und wegen zu großem Aufwand wieder verworfen
- [x] Photon Realtime Networking implementiert
- [x] Erste Multiplayer-Tests mit mehreren Spielern
- [x] Multiplayer-Lobby mit mehreren Räumen implementiert



Multiplayer

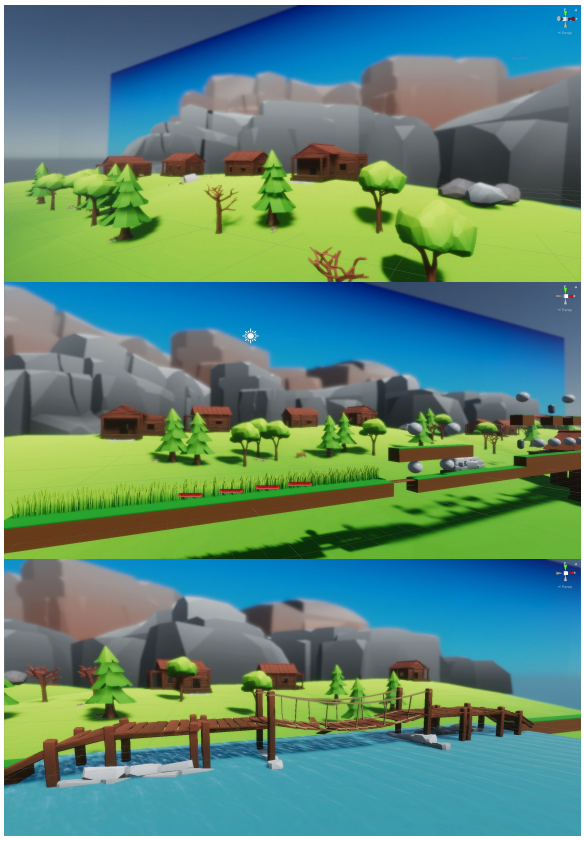
Mit **Photon Unity Networking (PUN)** war es uns möglich, unser Spiel von Grund auf inkl. der gewünschten Netzwerkfunktionen zu programmieren. Mit Hilfe von PUN lassen sich in unserem Spiel Räume erstellen, denen man anschließend beitreten kann. Alle Spieler in einem Raum spielen dann im Netzwerk zusammen gegeneinander - dabei werden alle Aktionen wie Bewegung, Kollision, Schaden, usw. über das Netzwerk synchronisiert, sodass jeder Spieler "das gleiche sieht".



Erstes Level

- [x] Animation over Network implementiert
- [x] Erstes UI-Design für Hauptmenü und Lobby
- [x] Charakterauswahl implementiert
- [x] Mehrere Multiplayer-Funktionstest
- [x] Recherche nach passenden Assetpacks und anschließendes Testing
- [x] Implementierung von ausgesuchten Assetpacks
- [x] Konzeptionierung, Desig und Erstellen des ersten Levels
- [x] Umstieg von CharacterController auf RigidBody
- [x] Erweiterung des RigidBody Movement-Scripts, Implementierun eines "Dashes"
- [x] Weitere Low-Poly Charaktermodelle in Blender erstellt und nach Unity exportiert
- [x] Erste Low-Poly Itemmodelle (z.B. Rocket Launcher) in Blender erstellt und nach Unity exportiert
- [x] Weitere Experimente mit RigidBody Movement (Double Jump, etc.)
- [x] ItemHandler implementiert
- [x] Rocket Launcher modelliert und nach Unit exportiert

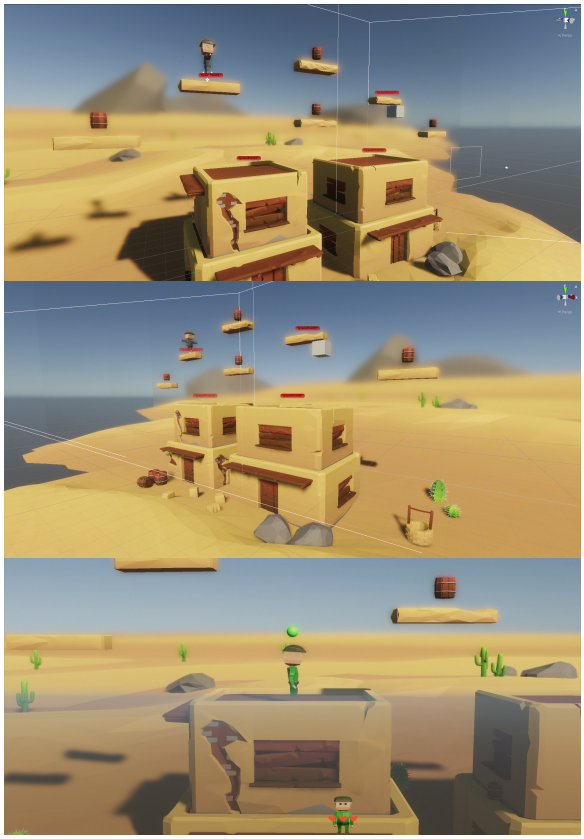
Als das Spiel noch als Runner-Partyspieler konzipiert war, haben wir ein erstes Level entworfen. Hierfür wurden zunächst Levelbausteine in Sand- und Grasfarben mit verschiedenen Höhen und Breiten erstellt, aus denen anschließend Großteile des "Laufpfades" gebaut wurden. Für die Dekoration, speziell im Hintergrund, haben wir uns zum Teil mit Objekten aus einem Asset Pack beholfen (Auflistung aller Tools siehe unten). Um den Hintergrund des Levels nicht zu präsent wirken zu lassen und den Spieler auf das Spielgeschehen zu konzentieren, haben wir außerdem mehrere "Blur Layer" implementiert und vor den Hintergrund gelegt, damit dieser eine unscharfe Optik erhält. Die Unschärfe der Layer lässt sich über Schieberegler justieren (siehe Video).



Zweites Level

Nachdem die Entscheidung getroffen wurde, den Umfang des Projektes etwas einzudämmen, um am Ende zumindest einen spielbaren Prototypen vorliegen zu haben, haben wir uns bewusst von dem Runner-Part, bzw. das Um-die-Wette-laufen, unseres Spiels entfernt und dieses als Partyspiel in Arena-ähnlichen Levels neu konzipiert. Für das neue Konzept wurde deshalb ein Arena-Level entworfen und erstellt - dieses Mal im Wüsten-Look und um einiges überschaubarer.

- [x] Rocket Launcher Funktio (aufnehmen und schießen) implementiert
- [x] Weitere Tests und Experimente mit Movement, Drag und Rigidbody
- [x] Experimente mit Procedural Generation (des Levels)
- [x] Projekt notgedrungen vo Unity Collab auf Plastic SCM migriert
- [x] Konzept und Scope des Projekts verschoben und verkleinert
- [x] Neues Level erstellt (Desert)
- [x] Implementierung von GroupCamera und PlayerMarker
- [x] Etliche Anpassungen am UI vorgenommen
- [x] Erste Tests mit FEEL (Unity Addon für Feedback, Sound, usw.)
- [x] Reset Button (für Levelneustart) implementiert)
- [x] Rocketlauncher Finetuning
- [x] Movement Finetuning
- [x] Erste Tests zu Implementierung von Sound
- [x] Herz (für Lebenspunkte) in Blender modelliert und nach Unity exportiert
- [x] UI Update bei Verlust von Herzen implementiert
- [x] Cinemachine (FEEL) gefixed



- [x] Item Animation implementiert
- [x] Erste Soundeffekte implementiert
- [ ] Weitere Soundeffekte un Musik implementiert
- [ ] GameOver-State und -Screen implementiert
- [ ] SFX für Rocketlauncher und Explosion implementiert
- [ ] Rocketlauncher Schaden implementiert
- [ ] Rocketlauncher Knockback implementier

Herausforderungen

Wie bei den meisten Projektarbeiten kommt am Ende doch vieles anders, als man denkt - auch unser Projekt blieb von unerwarteten Herausforderungen oder Hürden nicht verschont. Im Folgenden werden diese weiter ausgeführt um u.a. ein deutlicheres Bild unserer Arbeit an dem Projekt zu schaffen und zumindest teilweise unser Endergebnis zu begründen.

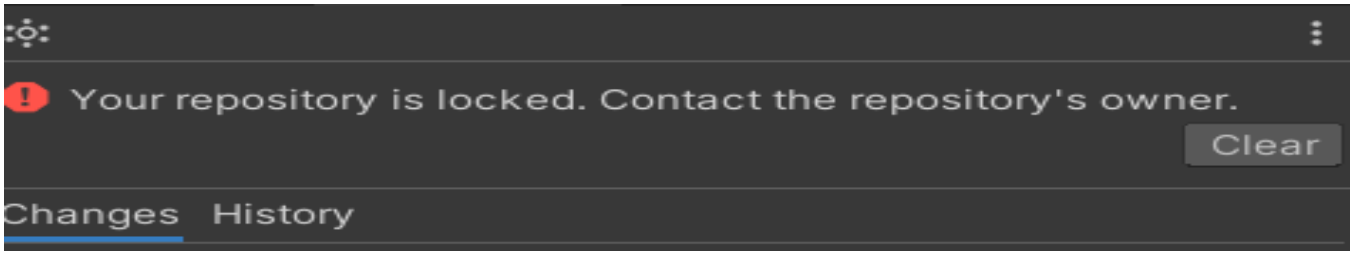
Online-Multiplayer

Dass sich die geplante Online-Komponente in unserem Projekt als Herausforderung entpuppen würde, war aufgrund der mangelnden Erfahrung zwar zu erwarten, jedoch war das Ausmaß dann doch umfangreicher als gedacht. Auch beim Multiplayer führen viele Wege zum Ziel und bei den ersten Online-Recherchen stößt man nicht direkt auf "**die eine Lösung**". Daher hatten wir es zuerst mit einer Dedicated Server Lösung versucht, die sich aber bald als zu komplex und umfangreich für unser Projekt entpuppte. Wie oben bereits erläutert, war es am Ende **PUN (Photon Unity Networking)**, das bei uns nun zum Einsatz kommt. Mit PUN war es uns auch mit verhältnismäßig wenig Erfahrung möglich, ein Grundgerüst für den Online-Multiplayer zu schaffen. So hatten wir z.B. recht schnell die Funktion implementiert, Räume zu erstellen, sich gegenseitig zu finden und einem Raum beizutreten. Die eigentliche Herausforderung der Online-Komponente war jedoch, das eigentliche Gameplay über das Netzwerk z synchronisieren. So kam es ständig zu Verzögerungen (Lags), Animationsrucklern oder Synchronisationsprobleme, bei denen z.B. einige Spieler nicht exakt das Gleiche im Spiel sahen wie andere. All diese Probleme mussten beim Implementieren von Gameplay Features, Logiken und Funktionen beachtet und ständig neu angefasst werden, was im Großen und Ganzen einfach **sehr viel Zeit i Anspruch** nahm.

Unity Collaborate & Plastic SCM

Um gemeinsam an einem Unity-Projekt zu arbeiten, bedarf es im besten Fall einer **Versionskontrolllösung**, die Code von mehrere Personen zusammenführt (Merging) und verwaltet. Eine der bekanntesten Lösungen ist **Git**. Für unser Projekt hatten wir uns jedoch für **Unity Collaborate** entschieden, da diese Lösung zum einen noch direkter und komfortabler in den Unity Editor integrier ist und wir zum anderen als Studenten von Unity kostenlos mehrere Gigabyte an Cloudspeicher zur Verfügung gestellt bekamen, der direkt von Unity Collaborate genutzt werden konnte. Alles in allem eine sehr inuitive und leicht zu bediende "Rundum-

sorglos-Lösung", die uns für viele Stunden in unserer Projektarbeit begleitete und sauber funktionierte. Da Collaborate jedoch Ende 2021/Anfang 2022 von Unity abgeschaltet wurde, waren wir plötzlich mittendrin gezwungen, unser Projekt auf die neue Lösung von Unity zu migrieren: **Plastic SCM**. Im Gegensatz zu Collaborate entpuppte sich Plastic SCM schnell als mächtiges, aber auch leider sehr komplexes Werkzeug. Aufgrund der mangelnden Erfahrung und zu dem Zeitpunkt auch sehr dürftigen Dokumentation war unser Migrationsprozess von etlichen Problem geplagt, u.a. mussten wir unser Projekt auf eine neuere Unity-Version anheben, da nur bestimmte LTS-Versionen (long term support) für eine Migration auf Plastic SCM kompatibel waren. Das Anheben der Unity-Version brachte dann, wie zu erwarten, weitere Probleme mit sich, z.B. mit unseren Assetpaketen und den Shadern. Mit viel Mühe haben wir es dann zwar geschafft, unser Projekt in der neuen Version sauber zu migrieren und anschließend die Fehler ausmerzen, jedoch war das Ganz ein zeitintensiver und nervenaufreibender Prozess, den wir niemandem wünschen wollen.



Movement, Physics & Kollisionen

Da es in unserem Spiel zum Großteil ums Laufen und Springen geht, müssen sich diese Dinge beim Spielen natürlich flüssig und "gut" anfühlen. Das ist jedoch leichter gesagt als getan. Rudimentäre Bewegungsabläufe wie Laufen, Springen, Rutschen oder Dashen in unsere Programmierlogik zu implementieren war zunächst keine wirklich große Herausforderung. Knifflig wird es erst dann, wenn man sich Gedanken darüber macht, ob sich all diese Dinge auch richtig anfühlen und gut aufeinander abgestimmt sind. Dazu gehören Parameter wie z.B. Trägheit, Beschleunigung, Bremsen und vor allem die Gravitation. Jeder, der schon einmal ein wirklich gutes Jump 'n' Run, wie z.B. ein Spiel aus der Super Mario Reihe, gespielt hat, wird bei seinem eigenen Spiel schnell merken, dass sich alle Bewegungen unsauber, zu träge oder schlichtweg "falsch" anfühlen. Und nach unserer Erfahrung dauert es wirklich sehr lange, bis man mit dem Finetuning an einem Punkt angekommen ist, an dem man mit seinem Movement auch nur halbwegs zufrieden ist. Die nächste Hürde, die sich uns dann wortwörtlich in den Weg stellte, waren die Kollisionen. Da unsere Level zum Großteil aus mehreren Plattformen bestehen, auf denen sich der Spieler bewegt bzw. die er durch Sprünge erreichen kann, muss auch die Kollision der Spielerfigur mit den Objekten sauber sein. Wenn ein Spieler z.B. auf eine Plattform springen möchte, den Sprung jedoch zu kurz setzt und mit der Kante der Plattform zusammenstößt, muss die Spielerfigur entsprechend gestoppt werden - auch das muss implementiert werden, denn ohne eine vernünftige Kollision wird die Spielerfigur nicht gestoppt, sondern rutscht mit der gleichen Geschwindigkeit über die Plattform hinweg. All diese Mechaniken zu implementieren mag dem einen oder anderen auf den ersten Blick vielleicht als einfach erscheinen, jedoch ist genau das Gegenteil der Fall. Da der Kern unseres Spiels eben aus genau diesen Mechaniken besteht, müssen sich diese so gut, flüssig, passend und schnittig anfühlen wie möglich - und das dauert.

Projektumfang/Scope

Im Laufe unserer Projektarbeit kristallisierte sich immer mehr heraus, dass wir uns mit dem Runner-Aspekt unseres Spiels vielleicht etwas zu viel vorgenommen hatten. Ein Spiel, bei dem man zusätzlich zum gegenseitigen Bekämpfen auch noch um die Wette läuft, braucht entsprechend lange Laufstrecken und damit auch umfangreichere Level. Schon bei unserem ersten Level ist uns klar geworden, dass dieses für ein gutes Spielerlebnis noch um ein vielfaches länger sein müsste - dabei war der Aufwand bis dahin schon nicht zu knapp gewesen. Die Herausforderung war also an dieser Stelle, gemeinsam die Entscheidung zu treffen, bei unserem Projekt sinngemäß die Reißleine zu ziehen. Nach vielen Diskussionen wurde letztendlich die Entscheidung getroffen, den Umfang bzw. "Scope" des Projekts zu verkleinern. Die Runner-Komponente wurde entfernt und es wurde ein zweites, neues Level gebaut, das eher einer Arena gleicht. Das Ziel des Spiels war es also nicht mehr, als erster im Ziel, sondern in der Arena der letzte Überlebende zu sein. Auch bei den Items bzw. Powerups haben wir uns von vielen Ideen erstmal nur auf den Raketenwerfer beschränkt, da uns wichtiger ist, ein Item sauber und gut zu implementieren, als mit vielen Ideen zu jonglieren, die alle angefangen, aber nicht beendet werden.

Ziele

In der folgenden Tabelle sind unsere ursprünglich geplanten Ziele in Form von Spielelementen aufgelistet und in "Must Have" bzw. "Nice To Have" unterteilt sowie mit der jeweiligen Priorität quantifiziert. Außerdem wurden der Status farblich markiert, sodass direkt ersichtlich ist, welche Ziele bisher erreicht wurden und welche nicht.

Prio (0-10)	Must Have	Nice To Have
8	FERTIG Ein vollständiges Level	IN ARBEIT Zwei oder mehr Level
6	FERTIG Items bzw. Powerups	FEHLT Fähigkeiten
9	FERTIG Kamera	
6	FEHLT Highscore-System	
5	IN ARBEIT Sounds & Musik	
10	FERTIG 2-4 Spieler	ABGELEHNT > 4 Spieler
6	IN ARBEIT Hauptmenü / Multiplayer-Lobby	
6	FERTIG Start-Screen	
6	IN ARBEIT Game-Over-Screen	
6	FEHLT Einstellungen bzw. Settings	
9	FERTIG Movement: Laufen	FERTIG Movement: Slide bzw. Dash
9	FERTIG Movement: Jump	ABGELEHNT Movement: Wall Jump
9	ABGELEHNT Movement: Double Jump	

9	FERTIG Animationen	
5	FEHLT Controller-Implementierung	
7	IN ARBEIT Gameplay-Effekte	FEHLT Environment-Effekte

Zeitaufwand/Stundenrechnung

ⓘ Wird aktuell in einer separaten Excelliste gepflegt und zu einem späteren Zeitpunkt in das Wiki übertragen.

Lessons Learned

ⓘ Wird gegen Ende der Projektarbeit ausformuliert.

Tools

Im Folgenden noch eine vollständige Auflistung der verwendeten Tools, Unity-Plugins und Assets:

- Unity Version 2020.3.8f1 mit DX11 und HDRP (High Definition Render Pipeline)
  - Photon Unity Networking (PUN)
  - FEEL (ehemals MMFeedbacks)
  - Lowpoly Style Ultra Pack (CH Assets)
- Unity Collab
- Plastic SCM
- Blender
- Audacity
- Discord
- ShareX
- Handbrake
- Instagiffer
- Microsoft Excel