Dokumentation

Da wir entsprechend des Themes ein möglichst organisches Leveldesign wollten, brauchte es einen Editor mit so vielen Freiheiten wie möglich.
Der von Phaser zur Verfügung gestellte Tileset basierte Editor vereinfacht zwar vieles, ist aber auch stark limitiert und eignete sich nicht.
Das ganze Level zu hard coden wäre eine andere Möglichkeit gewesen, wäre auf Dauer aber sehr arbeitsintensiv.
Um das zu umgehen haben wir als Vorarbeit einen Level Editor erstellt, der momentan den größten Teil unseres Projekts ausmacht, leider auch so viel Zeit in Anspruch genommen hat, dass das Spiel nicht in allen geplanten Aspekten umgesetzt werden konnte.
Dafür ist aber ein solider Grundstein gelegt, der weiteres Leveldesign deutlich erleichtert und beschleunigt.

Programm System Erklärung

Generell

Um das Managen und Integrieren von neuen und vorhandenen Systemen und Objekten so einfach wie möglich zu gestalten, galt es vorhandene und erstellte Managementsysteme so generell und universell wie möglich zu gestalten.
Konkretes Beispiel ist die Automatisierung der “update” Events mit den in Phaser vorhandenen Systemen.

Accumulator

Da die Frequenz der Bearbeitungsschritte der Engine an die generierten Frames gebunden sind, sorgt dies für unterschiedliches Verhalten des Spiels auf verschiedenen Geräten.

Der Accumulator umgeht dies, indem er die Methode “fixedUpdate” in der Frequenz der eingestellten FPS (Frames Per Second) aufruft, indem er die Zeit zum letzten gerenderten Frame sammelt (eng. accumulated) und seine Methode aufruft, sobald die Frequenz, die er anstrebt, erreicht ist.

Der Accumulator kann in jedem Objekt manuell initialisiert werden oder automatisiert in allen Objekten, die einer Gruppe hinzugefügt werden.

Die “fixedUpdate” methode muss leider manuell in das Objekt hinzugefügt werden.

Game Objects

Die zwei Custom Klassen, worldObjSprite und worldObjImage bilden die Parent Basis Klassen für andere Game Objekt Klassen.
Die Basis Klassen sind abgeleitet von den Phaser internen Klassen Sprite und Image, dienen als Bündelung und besitzen schon die vom Accumulator benötigte “fixedUpdate” Methode.

PhyObj MovementObj, ConnectObj, sind absteigend voneinander abgleitet.

1.PhyObj: definiert ein Objekt, das von der Phaser Matter Physic-Engine beeinflusst wird und weist ihnen eine Kollisionskategorie und Maske als 32Bit Integer zu, die darüber entscheiden, mit welchen Objekten es kollidiert.
Wir haben die Verwendung von Physik Kollisionsgruppen vermieden, da diese dieselbe Funktionalität mit zusätzlichen Einschränkungen wie Kategorien + Maske bieten.
Somit kann jedes Objekt beliebig vielen Kollisionskategorien angehören und mit ihnen über die Maske Rechen effizient kollidieren.
Diese Klasse würde z.B. für alle Physik-aktivierten Objekte verwendet werden wie z.b. Gerümpel, welches mit dem Spieler und anderer AI kollidieren soll.

2.MvementObj: definiert daran anknüpfend Methoden zur erlauben und beschränken von Bewegung des Objekts, via generalisierten inputnahme Methoden und weiteren Funktionen. Diese sind für die Einbindung von Spieler oder AI Steuerung designt.

3.ConnectObj ist eine Parent-Klasse für Spieler und Boss Objekte, die die Bewegung entlang der Wände realisiert. Dies passiert über ein Physik Body Sensor Objekt, welches eine Überlappung mit Objekten wahr nimmt, die für eine solche Verbindung via der “CONNECTER” und “CONNECTABLE” Kollisionskategorie aktiviert sind.
Geplant ist, dass hier auch die Bewegungseinschränkung des Objektes zur Wand sowie die das Finden von sogenannten Fußstützen an der Wand stattfindet.
Ersteres hat sich jedoch komplizierter als erwartet herausgestellt, wobei die unregelmäßigen strukturellen Entscheidungen in Phaser die bei weitem größte Hürde sind.

Scenes

Für Scenes existieren SceneMainMenu und die eigentliche Spiel Szene SceneMainGame.
SceneMainMenu soll später das Hauptmenü darstellen, wird im Moment aber nur zum Laden und initialisieren von Grundsystemen verwendet und wechselt dann automatisch in die Spiel-Szene.
Ein Arbeitsziel, mit der Haupt-Spiel Szene wäre generelle Funktionen auf ihren Parent zu übergeben, sodass der Parent eine Basis für alle spielbaren Szenen bildet.

Debugger

Der Debugger ist ein Development Hilfsobjekt welches daten darstellen und Funktionen ermöglicht, die für die Entwicklung des Spiels von Nutzen sind.
Bei der Erstellung der Spiel-Szene wird der Debugger kreiert, welcher optional den Level Editor initialisieren kann.

Level Editor

Bei Initialisierung des Editors lädt er alle Ressourcen in seiner “EditorAssets.json” Datei, und stellt diese in der Ressourcenliste dar.
Diese Ressourcen Lassen sich in der Scene platzieren, wobei der Editor nach dem Ressourcentyp entscheidet, wie dies genau passiert.
Implementiert sind “image” und “polygon” Typen, welche respektive ein Image-Objekt mit der Ressource platzieren oder ein Polygon mit der angegebenen Funktion erstellen.

Die editierbaren interaktiven Objekte “CollisionInstance” und ”ImageInteractive” sind jedoch nur in Nutzen, wenn der Level Editor initialisiert ist.
Wenn dies nicht der Fall ist, sind diese Objekte vom Typ generischer Phaser Objekten.
Ohne den Level Editor sind zahlreiche Funktionen wie z.B. die interaktive Komponente nicht mehr gebraucht, und der Nutzen von simpleren Klassen ist ausreichend.
Der Editor hat mehrere Modie und interne Systeme, die er selber verwaltet, indem er sie aktiviert oder deaktiviert, wie es für den aktuellen Modus oder andere Umstände angebracht ist. Für eine Anwendungsanleitung konsultieren Sie bitte die Level Editor Manual.
Diese liegt unter “/src/Objects/Systems/LevelEditor_Manual.md”.

Speichern und laden

Alle Assets werden per json file geladen, über die Phaser interne „files“ property. Es gibt 4 Arten von asset json files.

"alwaysAssets.json" beinhaltet alle assets die dauerhaft im Spiel gebraucht werden. diese Assets werden immer geladen, sobald das Spiel hochgefahren wird. "EditorAssets.json" beinhaltet assets die nicht dauerhaft genutzt werden, diese Datei wird nur geladen, wenn der Level Editor instanziiert wurde, also nicht in der finalen Version des Spiels. Der Editor hat Zugriff auf die Assets, kann sie platzieren und mit Zonen assoziieren, die dann wiederum in der Datei „zoneData.json“ gespeichert werden, sodass das Spiel dynamisch Assets laden kann, die für die jeweilige Zone gebraucht werden. Object spezifische asset files sind assets die konkret für ein Objekt benötigt werden, z.B. „MainMenuAssets.json“ welche alle assets beinhaltet die für das Main Menü benötigt werden.

Editor

Der Editor hat zwei Modi. Im Create Modus kann man Zonen, collider und images erstellen. Die Zone ist vom Typ Polygon und definiert einen Bereich in dem alle Polygon-, und Image Objekte gesammelt in der json Datei „zoneData“ gespeichert und geladen werden. Das Erstellen von Zonen ist noch nicht implementiert, die Tutorial Zone ist hart kodiert, Das Zonensystem an sich, also speichern und laden von Zonen funktioniert.

Collider sind im Editor eine Custom Class die abgeleitet ist interaktiv sind. Man erzeugt neue Polygone, indem man durch Klicken einen Punkt hinzufügt, Steuerung vervollständigt das Polygon. außerhalb des Level Editors werden sie zu statischen, nicht interaktiven, unsichtbares Physics-Body Objekten konvertiert. Konkave Polygone lassen sich bisher zwar platzieren, sorgen aber für fehlerhafte Kollisionen in der Physik-Engine. Image Objekte sind im Editor einer Custom Class abgeleitet von der Klasse Image. Man kann aus einer Liste an Images aussuchen und sie durch Klicken platzieren.

Im Edit Modus werden verschiedene Eigenschaften/ Werte der Objekte angezeigt und können bearbeitet werden. Der Typ des Objekts wird ganz oben angezeigt. Welche Ressource ausgewählt, also der Name steht darüber. Sowohl Images als auch Collider können im Edit Modus bearbeitet werden. Per Tastatur lassen sich Objekte außerdem rotieren skalieren, entfernen. Mit J lässt sich der Debugger ein und ausschalten, mit L kann der Leveleditor ein und ausgeschaltet werden.

Spiel

Das Spiel wurde bisher nur in seinen Grundzügen umgesetzt. Die generelle Steuerung, das heißt die Bewegung entlang von Wänden, und springen funktioniert und ein Tutorial Level wurde erstellt.

Tutorial Level:

Das Tutorial stellt eine Einleitung in die Spielmechaniken dar, bietet aber gleichzeitig auch einen Einblick in die Möglichkeiten des Level Editors. Durch die Überleitung von einem engen Gang in einen großen Raum, wird einerseits das weitläufige Ambiente dargestellt, andererseits wird dem Spieler bewusst gemacht das Bewegung nur entlang von Wänden möglich ist. Im nächsten Raum wird der Spieler dazu animiert zu springen. Das Level zeigt außerdem, das mit dem Editor ein Parallax Effekt erzeugbar ist, Sprites sich auf verschiedene Ebenen verteilen lassen, ein färbbar sind und andere Funktionen. Das erste Level vermittelt außerdem einen Eindruck von der Allgemeinen Atmosphäre des gesamten Spiels, Assets wurden vielfach gelayert, rotiert, verzerrt, sodass ein dynamisches und weitläufiges Gefühl entsteht. Die Spielerbewegung ist noch nicht vollständig implementiert. bisher ist die Bewegung des Spielers an Wänden möglich. Abseits von Wänden lässt sich der Player nicht Steuern und fliegt konstant in eine Richtung. Noch nicht umgesetzt wurden die Beine des Charakters welche mit inverser Kinematik bei der Laufbewegung animiert werden sollen.

Genutzte Ressourcen

Für den das Editor Menü im Editor wurde dat.gui verwendet, eine lightweight controller library mit der man Variablen live vom Browser aus verändern kann

Für die Erstellung der Sounds wurden einige Sound Assets von Envato Elements verwendet.