SPIEL-PROJEKT Q2

RUN



AND

JUMP

**Teilnehmer**: Jakob Engel, Jakob Meyer, Mason Wins

Inhaltsverzeichnis

Inhalt

[2 Ziel des Projekts 2](#_Toc1043000681)

[3 Darstellung des Spiels und Implementierung 3](#_Toc1751269857)

[4 Meilensteine 3](#_Toc998114973)

[4.1 Loop 3](#_Toc2028034348)

[4.2 Steuerung 3](#_Toc2103257087)

[4.3 Kollisionen 3](#_Toc1966484448)

[4.4 Map 4](#_Toc1592448537)

[4.5 Konsole 4](#_Toc196901074)

[4.6 Texturen 4](#_Toc1210174478)

[4.7 Untermenüs 4](#_Toc486557444)

[5 Ausblick 4](#_Toc419770844)

# **Ziel des Projekts**

Unser Ziel ist es ein präsentierbares, gutaussehendes und funktionierendes Jump and Run Spiel zu entwickeln. Wichtig ist uns dabei, dass auf jeden Fall die Grundfunktionen eines Jump and Run Spiels funktionieren, also das der Charakter sich vorwärts, rückwärts, hoch und runter bewegt, auch sollen genug Maps vorhanden sein, damit das Spiel abwechslungsreich wird. Außerdem liegt unser Augenmerk darauf, dass das Spiel flüssig läuft und es zu keiner unerwarteten Unterbrechung kommt.

Das Spiel soll in einer Auflösung von 1920x1080 spielbar sein. Wichtig sind uns auch der spielbare Charakter und wechselnde Landschaften in den Maps. Wir möchten zudem auch unser Wissen anwenden können und wollen herausfinden wie weit uns unsere Java Kenntnisse bringen können. Es ist daher unser Ziel etwas spielbares Spiel zu haben, also ein Produkt, welches aus unserem eigenen Wissen und Können entstanden ist. Darüber hinaus wollen wir auch an sich ein wenig Zeit- bzw. Projektmanagement üben und für uns wichtige Erfahrung für unser späteres Leben sammeln.

# **Darstellung des Spiels und Implementierung**

Das Spiel ist­­ in 2d-Optik dargestellt und hat einen Charakter, welcher von Objekt zu Objekt springen muss, um an sein Ziel zu gelangen. Die Objekte haben jeweils verschiedene Texturen (Erde, Stein, Sand oder Wasser). Alles hat einen Pixel Art Style, sowohl der Charakter als auch die Texturen und der Hintergrund. Das Ziel des Spiels ist es jeweils über verschiedene Hindernisse (Objekte) zum Ziel zu gelangen. Das Spiel besitzt eine Startseite, von der man zur Level-Auswahl kommt oder in den Einstellungen die Tastenbelegung ändern kann. Von der Level-Auswahl kommt man direkt zur Level-Übersicht, auf welcher ein PNG des Levelaufbaus zu sehen ist sowie die lokale Bestzeit.

Unser Spiel haben wir in Java mit den Swing- und AWT-Paketen programmiert. Das Spiel besitzt 20 Klassen bestehend aus Klassen für die Map, für das Allgemeine, den Charakter und jeweils eine Klasse für die Texturen. Die Klasse die sich mit der Physik und den Kollisionen beschäftigt vererbt an einen Großteil der anderen Klassen, wie die Map-Elemente oder den Charakter.

Die Map wird aus einer Datei ausgelesen, in welcher die Texturen, Positionen sowie die Dimensionen gespeichert sind. Die Datei kann zur Laufzeit durch die Konsole verändert werden, was das Programmieren so wie das gestalten der Map um einiges erleichtert.

# **Meilensteine**

## Loop

Ein Loop Objekt, das von Thread erbt aktualisiert 60-mal in der Sekunde die Position der Objekte. Dafür wird eine Dauerschleife gestartet, die durch die Thread.sleep() Methode 60 mal in der Sekunde die Update- und Draw-Methode der Map aufruft. Die Objekte werden vorerst durch Rechtecke dargestellt.

# 

## 4.2 Steuerung

Wir haben uns erst einmal die Steuerung überlegt und umgesetzt. Wenn eine Taste gedrückt wird, dann wird ein boolean auf true gesetzt. In der calculatePosition Methode des Spielers wird dann zuerst die Geschwindigkeit und dann die Position berechnet. Dabei wird die vertikale Geschwindigkeit mithilfe einer Beschleunigung berechnet, um ein realistischeres Spielgefühl zu erreichen. Wenn man losläuft, ist man auch nicht sofort bei der maximalen Geschwindigkeit, sondern braucht eine kurze Zeit, um diese zu erreichen. Schließlich wird in der draw Methode die gesamte Map neu gezeichnet.

## Kollisionen

Die Kollisionen werden durch einen von Java entwickelten Algorithmus überprüft, dieser stammt aus der Java AWT Rectangle Klasse. Um zu verhindern, dass der Spieler mit einem Map-Element überlappt, wird sobald eine Kollision zwischen dem Spieler und einem Map-Elemet festgestellt wird die Position des Spielers zurückgesetzt und Pixel für Pixel und die richtige Richtung verschoben, bis wieder eine Kollision dieser Richtung vorliegt.

## Map

Die Map wird über eine Textdatei ausgelesen und in unser Spiel eingefügt. In der Datei ist zuerst die Art des Map-Elements festgelegt, also die Textur, die es haben soll. Danach steht die Position und zum Schluss die Höhe und Breite.

## Konsole

Über Eingabe von Text, kann man mit der Konsole die Map bearbeiten und muss so nicht nach jeder Änderung das Spiel neustarten.

## Texturen

Als nächstes haben wir unsere selbst gestalteten Texturen in unsere Map hinzugefügt, um das Spiel etwas aufzuhübschen. Die Texturen gehören jeweils eine Klasse an und werden geladen, sobald das erste Mal ein Objekt der Klasse erstellt wird. Die paintComponent Methode der Map, die von JPanel erbt wird überschrieben. Von dort aus wird die selbstgeschrieben draw Methode der Map-Elemente und des Spielers aufgerufen und das Graphics Objekt wird übergeben. In der draw Methode Ist für jede Klasse festgelegt, wie sie auf der Map zu erscheinen hat und sie malt sich mithilfe des Graphics Objekts auf die Map.

## Untermenüs

Außerdem wurden Untermenüs hinzugefügt wie: Einstellungen, veränderbare Belegung der Steuerung und die Levelauswahl.

# Ausblick

Neben mehreren spielbaren Maps, wäre auch eine Auswahl aus mehreren Charakteren ein Ziel, welches wir erreichen wollen. Jedoch fehlt uns zur, bis wir eine Charakterauswahl implementieren können, zum einen mehrere selbst erstellte Charaktere und zum anderen ein eigenes Menü für die Auswahl an Charakteren. Ein eher schwieriges Ziel ist es die Kamera sowie die Map an sich dynamischer zu machen. Das heißt dass die Map z.B. drehende oder sich hoch und runter bewegende Elemente besitzt, die zu mehr Interaktion mit dem Spieler führen. Interaktive Elemente könnten z.B. Wolken sein, auf denen man nur wenige Sekunden stehen darauf stehen kann und nach wenigen Sekunden herunterfällt. Zudem könnten sich diese besagten Wolken noch bewegen, um das darauf springen zu erschweren. Eine weitere Idee wären interaktive Elemente wie ein Heißluftballon, welcher den Spieler z.B. über eine Schlucht fliegt oder eine sich öffnende und schließende Brücke, wo es auf das Timing des Spielers ankommt. Es soll auch durch die Kamera dynamischer wirken, indem die Kamera quasi den Charakter verfolgt und so einen größeren Spielraum abbildet. Dafür muss sich die Map auch dynamisch erweitern damit die Kamera den Spieler verfolgen kann. Ein größeres Vorhaben ist, Gegner in unser Spiel zu implementieren. Dafür braucht es natürlich wieder mehrere Klassen, Gegnermodelle und Animationen. Der Charakter benötigt dann auch etwas wie eine Waffe, um sich zu verteidigen. Auch braucht das Spiel ein komplettes Lebens- und Schadenssystem, wobei die Gegner dem Charakter Schaden zufügen, wenn sie gegen ihn laufen, sprich eine Kollision erkannt wird.

Des Weiteren wollen wir den Charakter an sich agiler machen, durch die Möglichkeit des Doppelsprunges oder, dass man sich von Wänden abstoßen kann. Um sich von der Wand abzustoßen, muss der Charakter für wenige Sekunden an den Wänden festhalten können.

Zu guter Letzt, soll sich das Spiel automatisch an die Bildschirmgröße anpassen können, bisher ist das Spiel immer in 1920 x 1080 abgebildet.