Grundkurs: Informatik

Freiherr-vom-Stein-Gymnasium

Schuljahr-2025/26

Herr Meinhardt

Entwicklung eines Programmierspiels in Godot

Planung und Implementation eines Weltraum-Programmierspiels in [Programmiersprache einfügen]

Inhaltsverzeichnis

1. Einleitung

1.1 Relevanz und Zielsetzung

Computerspiele sind mittlerweile ein Teil des Lebens vieler Teenager. Meistens dienen sie nur zur Unterhaltung, doch sie haben auch anderweitig Potenzial. Unterricht wird heutzutage oft „gamifiziert“, also spielerisch Gestaltet, und einige Computerspiele existieren nur fürs Lernen. Diese sogenannten Lernspiele werden teilweise bereits in Grundschulen eingesetzt.

Ein besonders vielversprechender Anwendungsbereich ist das Erlernen von Programmierkenntnissen. Da die Informatik in unserer zunehmend digitalisierten Welt an Bedeutung gewinnt, stellt sich die Frage, wie der Einstieg in das Programmieren möglichst zugänglich und motivierend gestaltet werden kann. Sogenannte ‚Coding Games‘ verbinden dabei spielerische Elemente mit dem Erwerb grundlegender Programmierfähigkeiten und sprechen insbesondere jüngere Zielgruppen an. Nur sehr wenige dieser Spiele sind darauf ausgerichtet, auch für erfahrenere Programmierer interressant zu sein. Das Ziel dieser Facharbeit ist, ein Programmierspiel was sowohl für neuanfänger als auch für erfahrene Entwickler interressant ist zu entwickeln.

1.2 Verwandte Spiele und Inspiration

CodeCombat ist ein Beispiel für ein erfolgreiches Coding Game. Es vermittelt dem Spieler schrittweise grundlegende Programmierkenntnisse. Dabei steuert der Spieler einen Charakter durch einen Kerker mit verschiedenen Monstern, wobei sämtliche Bewegungen des Charakters im Voraus programmiert werden müssen. Das Spiel überzeugt durch ein motivierendes Konzept und bietet einen hohen Lerneffekt. Neben CodeCombat existieren zahlreiche weitere Coding Games mit ähnlichen Ansätzen. Ein Aspekt, der jedoch in vielen populären Beispielen fehlt, ist die Kompetitivität. Kompetitivität steigert Motivation, wenn Wettbewerb als Herausforderung erlebt wird und Kompetenzgefühl stärkt[1].

1 Deci, E. L., & Ryan, R. M. (2000). *The “What” and “Why” of Goal Pursuits.* Psychological Inquiry.

2. Theorethische Grundlagen

2.1 Gamifizierung und Lernspiele

Gamifizierung bedeutet, Spielerische Konzepte auf spielfremde Kontexte zu übertragen. Dabei werden typische Spielelemente wie Punkte, Bestenlisten, Abzeichen oder Fortschrittsbalken in nicht-spielerische Umgebungen integriert. Ein Beispiel wäre eine Mathematik-Lernplattform, bei der Schüler für gelöste Aufgaben Punkte sammeln und in Ranglisten aufsteigen. Das Ziel ist, die Motivation zu steigern, indem das Lernen durch diese spielerischen Anreize ansprechender gestaltet wird..

Lernspiele, auch als Serious Games bezeichnet sind hingegen vollständige Spiele, die hauptsächlich fürs Lernen ausgerichtet sind. Im Gegensatz zur Gamifizierung, bei der nur einzelne Spielelemente hinzugefügt werden, ist hier das Spielen selbst die Hauptaktivität, während das Lernen nahtlos in die Spielmechanik integriert ist. Ein bekanntes Beispiel ist die Sprachenlern-App Duolingo, die Sprachen in kurzen, spielerischen Lektionen vermittelt. Nutzer machen Fortschritte durch ihren Kurs, steigen auf Ranglisten auf und versuchen, Herausforderungen abzuschließen. Weitere Beispiele sind CodeCombat für das Programmierlernen oder Minecraft Education Edition für verschiedene Schulfächer.

Die Vorteile beider Ansätze sind offensichtlich. Sie fördern Motivation indem sie sofortiges Feedback bieten und erlauben risikofreies Ausprobieren. Studien belegen, dass gut gestaltete Lernspiele die extrinsische Motivation erhöhen können. Die Lernelemente müssen allerdings gut integriert und möglichst unauffällig sein, da es sonst wieder abschreckt.

2.2 Motivation durch Wettbewerb

Es gibt zwei Wichtige Arten von Motivation: Intrinsische und Extrinsische. Intrinsische Motivation beschreibt Handlung um ihrer selbst willen. Es wird also aus Spaß, Interesse oder Neugier gehandelt. Extrinsische Motivation beschreibt Handlung wegen äußerer Anreize, wie zum Beispiel Belohnung oder Anerkennung. Lernspiele sollten im besten Fall die intrinsische Motivation fördern. Es gibt drei

Grundbedürfnisse, die diese förden[2]. Die Kompetenz, die Autonomie und die Soziale Eingebundenheit. Wettbewerb ist in der Lage, jedes dieser Bedürfnisse anzusprechen. Die Herausforderung, die der Wettbewerb durch Vergleich mit anderen bietet ist ein guter Motivator. Wettbewerb erfüllt im Lernkontext eine wichtige motivationale Funktion, indem er Lernende dazu anregt, sich mit anderen zu messen und dadurch ihre eigene Leistung zu hinterfragen. Diese Herausforderung durch den Vergleich mit anderen wirkt als natürlicher Antrieb, da sie den Ehrgeiz weckt und das Bedürfnis anspricht, sich zu verbessern. Gleichzeitig steigert Wettbewerb das Engagement und die Anstrengungsbereitschaft der Lernenden, da ein klares Ziel vor Augen steht. Durch messbare Leistungsverbesserungen, die im Wettbewerb sichtbar werden, entsteht zudem ein starkes Kompetenzerleben: Lernende erfahren direkt, dass ihre Bemühungen Früchte tragen. Darüber hinaus besitzt Wettbewerb eine wichtige soziale Komponente, denn er setzt Interaktion mit anderen Spielern oder Lernenden voraus und fördert so das Gefühl der sozialen Eingebundenheit.

Der Psychologe Mihaly Csikszentmihalyi beschreibt mit dem Begriff „Flow" einen Zustand völligen Aufgehens in einer Tätigkeit, in dem Konzentration und Handeln mühelos ineinandergreifen. Dieser Zustand entsteht dann, wenn die Anforderungen einer Aufgabe in einem ausgewogenen Verhältnis zu den Fähigkeiten der Person stehen. Ist eine Aufgabe zu leicht, führt dies zu Langeweile und sinkendem Interesse; ist sie hingegen zu schwer, entsteht Frustration und die Motivation bricht ein. Wettbewerb kann den Flow-Zustand gezielt fördern, indem er durch adaptive Schwierigkeit genau jene Balance herstellt, die für das Erleben von Flow notwendig ist.

Wettbewerb geht mit einer Reihe positiver Effekte auf das Lernen einher. Er steigert Aufmerksamkeit und Konzentration, da das Wettkampfgeschehen erhöhte Präsenz erfordert. Zugleich entsteht eine starke Motivation zur Verbesserung der eigenen Fähigkeiten, weil der Vergleich mit anderen deutlich macht, wo noch Potenzial liegt. Langfristig kann Wettbewerb außerdem zu einer dauerhaften Bindung an die Lernumgebung führen, da der Wunsch, sich weiterzuentwickeln und besser abzuschneiden, immer wieder zur Rückkehr animiert.

Trotz dieser Vorteile birgt Wettbewerb auch erhebliche Risiken. Wiederholte

2 Deci & Ryan Selbstbestimmungstheorie

Niederlagen können zu Demotivation führen, insbesondere wenn Lernende das Gefühl entwickeln, grundsätzlich unterlegen zu sein. In solchen Fällen verlagert sich der Fokus häufig vom Lernziel hin zum reinen Gewinnen-Wollen, was den pädagogischen Mehrwert untergräbt. Besonders schwächere Lernende sind gefährdet, da sie strukturell benachteiligt werden und schnell den Anschluss verlieren können. Es ist daher wichtig, eine Balance zwischen kooperativen und kompetitiven Elementen herzustellen, damit Wettbewerb inklusiv und motivierend wirkt, ohne einzelne Lernende auszugrenzen oder zu überfordern.

3. Konzeption

3.1 Spielidee und Designentscheidungen

Das geplante Spielkonzept ist ein Programmierspiel, in dem zwei oder mehr Spieler gegeneinander antreten. Eine erste Idee war ein Wettrennen, vergleichbar mit einer vereinfachten Version von *Mario Kart*. Dieses Konzept bietet jedoch nicht genügend Varianz, um die Spieler zum Schreiben fortgeschrittener Skripte zu motivieren. Zwar könnte durch verschiedene Karten mehr Abwechslung geschaffen werden, dies würde jedoch einen zu hohen Zeitaufwand bedeuten und den Rahmen dieser Facharbeit überschreiten.

Eine alternative und geeignetere Idee ist ein Kampf zwischen zwei Spielfiguren. In diesem Szenario müssten die Spieler ihre Strategien an das Verhalten des Gegners anpassen, wodurch eine interessante Simulation entsteht. Da das Spiel auch dem Lernen dienen soll, muss auf ein brutales Szenario verzichtet werden. Stattdessen fiel die Wahl auf ein Weltraumthema, bei dem die Spielfiguren Raumschiffe sind, die sich gegenseitig mit Lasern angreifen.

3.2 Spielmechaniken

Der nächste Gedanke war die Ausrüstung des Raumschiffes. Dieses sollte mehrere Triebwerke und Waffen haben. Ein Triebwerk links und rechts, welche zur Drehung dienen und ein starkes am Heck. Das Schiff sollte einen ausrichtbaren Laser oben haben, und einen Speer oder ähnliches vorne. Dadurch entsteht weitere Varianz, da

verschiedene Strategien möglich werden. Potenziell hat das Schiff auch ein Schutzschild, welches ein und ausgestellt werden kann. Jedes Stück Ausrüstung verbraucht Energie, welche nur langsam regeneriert. Damit wird vorsichtige Planung belohnt und aggressive Strategien bestraft. Anders als bei zum Beispiel Code Combat, welches Spieler ihre Bewegungen im voraus hardcoden lässt, müssen Spieler hier dynamische Skripte die jeden Frame aufgerufen werden programmieren. Die Programmierschnittstelle sollte einfach nutzbar sein. Dabei besteht die möglichkeit, Spieler einfach in der nativen Sprache der Game Engine programmieren zu lassen, es wäre aber hilfreicher sie mit blockbasierter Programmierung zu unterstützen.

4. Implementierung

4.1 Technische Grundlagen

Das Spiel wird mit der Godot Game Engine in der Version 4.6 (stable) entwickelt. Ein großer Vorteil dieser Engine ist, dass sie kostenlos und Open-Source ist. Sie unterstützt sowohl zweidimensionale als auch dreidimensionale Spieleentwicklung und nutzt die integrierte Programmiersprache GDScript. GDScript hat eine ähnliche Syntax wie Python und ist damit vergleichsweise einfach und selbst für Anfänger gut verständlich. Alternativ besteht durch Plugins auch die Möglichkeit, in vielen verschiedenen Sprachen zu programmieren.

Godot bietet eine umfangreiche Dokumentation und eine aktive Community, die unter anderem ein hilfreiches Forum betreibt. Der integrierte Editor vereinfacht den Entwicklungsprozess erheblich, und Funktionen wie die eingebaute Physik-Engine oder das Szenen-System nehmen viele aufwendige Aufgaben ab. Im Vergleich zu schwergewichtigen Engines wie Unity oder Unreal Engine ist Godot deutlich leichtgewichtiger und damit ressourcenschonender.

Godot arbeitet mit sogenannten Nodes (Knoten) als Grundbausteine für Spielumgebungen. Diese Knoten lassen sich hierarchisch anordnen und miteinander kombinieren. Scenes (Szenen) sind Sammlungen von Nodes, die wiederverwendbar sind und selbst wieder als Knoten instanziiert werden können. Alle Nodes sind dabei mit dem Root-Node, dem sogenannten Tree, verbunden, auf den jedes Skript

zugreifen kann. Objekte können zusätzlich über das Event-System mittels Signals miteinander kommunizieren.

Für dieses Projekt sind folgende Knotentypen von zentraler Bedeutung: Der CharacterBody2D dient als Basis für bewegliche Spielobjekte mit integrierter Physik. Dieser benötigt eine CollisionShape2D, die den Kollisionsbereich – die sogenannte Hitbox – definiert. Die visuelle Darstellung des Raumschiffs erfolgt durch ein Sprite2D, das die entsprechende Grafik oder Textur anzeigt. Eine Area2D wird verwendet, um Überlappungen zu erkennen, beispielsweise für Laser oder Schilde. Zusätzlich kommen Timer für zeitgesteuerte Ereignisse zum Einsatz.

Der Aufbau eines Raumschiffs in Godot folgt dieser Struktur: Als Root-Node dient ein CharacterBody2D, der die physikalischen Eigenschaften verwaltet. Diesem werden als Child-Nodes eine CollisionShape2D für die Kollisionserkennung und ein Sprite2D für die grafische Darstellung hinzugefügt. An den CharacterBody2D wird außerdem ein Skript angehängt, das die Steuerungslogik implementiert. Weitere Module wie Waffen oder Triebwerke können als separate Child-Nodes ergänzt werden. Diese modulare Struktur ermöglicht eine klare Trennung von Darstellung, Physik und Logik, macht Komponenten wiederverwendbar und erlaubt eine einfache Erweiterbarkeit des Spiels.

Ein Raumschiff kann man so erschaffen, dass man einen ‚CharacterBody2D‘ erstellt, diesem eine CollisionShape2D und ein Sprite2D anhängt. Wenn man darauf ein Skript anhängt, kann man in diesem anfangen, die Logik zu erstellen. Das Schiff muss in der Lage sein, das Linke-, Rechte- und Hintertriebwerg zu starten, den Laser zu drehen und zu schießen und wichtige Daten abzurufen.

4.2 Spiellogik und Steuerung

Jede Fähigkeit des Schiffs wird in einer seperaten Funktion geregelt. Die Ship-Szene besteht dabei aus dem CharacterBody2D als Basis, dem Sprite2D-Node für die visuelle Darstellung, drei Partikelsysteme für das linke, das rechte und das mittlere Triebwerk (Left Thruster, Right Thruster, Middle Thruster), einer Laser-Node und einem Knoten namens CodeObject, welcher sich um die Ausführung des Spieler-

Codes kümmert. Jede dieser Komponenten besitzt zudem eine CollisionShape2D für die Kollisionserkennung.

Die ready-Funktion ist eine integrierte Godot-Funktion, welche ausgeführt wird, sobald das Schiff im Tree initialisiert ist. Darin wird hier die Spieler-ID gesetzt, die bestimmt, welches Spielerschiff es ist. Mit dieser ID wird die Position des Schiffs festgelegt. Spieler 0 spawnt im oberen Bereich mit zufälligen Koordinaten, während Spieler 1 im unteren Bereich erscheint.

Die Bewegung des Schiffs wird in der physics_process-Funktion verwaltet, die jeden Frame aufgerufen wird. Hier wird zunächst der Code des Spielers ausgeführt, indem eine Funktion des CodeObjects aufgerufen wird. In einem realistischen Weltraumszenario würden die Schiffe keine Reibung haben, und sich nur selber ausbremsen können. Da das Spiel aber nicht zu schwer sein sollte wird Geschwindigkeit mit linearer Interpolation (lerp) jeden Frame verringert damit das Schiff langsam automatisch bremst. Die drei Triebwerke haben unterschiedliche Funktionen und werden durch die Funktionen midthruster, leftthruster und rightthruster gesteuert. Das mittlere Triebwerk löst einen starken Schub in Flugrichtung aus mit einer Kraft von 5 mal dem angegebenen Kraft-Wert. Die seitlichen Triebwerke dienen hauptsächlich der Drehung. Das linke Triebwerk erhöht die Rotationsgeschwindigkeit um 5 mal Power, während das rechte sie entsprechend verringert. Sie erzeugen auch einen kleinen Vorwärtsschub. Die meiste Geschwindigkeit gewinnt man also, wenn man alle Triebwerke gleichzeitig aktiviert. Alle Triebwerke akzeptieren nur Power-Werte zwischen 0 und 1.

Das Waffensystem besteht aus einem drehbaren Laser. Die turnlaser-Funktion erlaubt es, den Laser um 1 Grad pro Aufruf nach links oder rechts zu drehen, abhängig vom übergebenen Parameter. Die shoot-Funktion instanziert ein Bullet-Objekt, welches in Laserrichtung schießt. Die Geschwindigkeit des Geschosses setzt sich zusammen aus einer Grundgeschwindigkeit von 300 in Schussrichtung plus der doppelten aktuellen Schiffsgeschwindigkeit, sodass Geschosse von einem sich bewegenden Schiff schneller fliegen. Auf das schießen gibt es ein Cooldown von 0,3 Sekunden.

Jede Aktion verbraucht Energie. Das mittlere Triebwerk kostet 10 Energiepunkte, die seitlichen Triebwerke jeweils 1 Punkt, und ein Laserschuss verbraucht 15 Punkte.

Wenn nicht genügend Energie vorhanden ist, wird die entsprechende Aktion nicht ausgeführt. Dies zwingt die Spieler dazu, ihre Energie strategisch einzusetzen und nicht alle Systeme gleichzeitig auf Maximum zu betreiben. Energie wird mit einer Rate von 2 pro Sekunde aufgeladen.

Für die Programmierung stehen dem Spieler einige Funktionen zur Datensammlung zur Verfügung. Darunter ist die aktuelle laser-drehung und die Energie. Die zwei wichtigsten Hilfsfunktionen sind angle_to_enemy und dist_to_enemy, die jeweils die Richtung zum Gegner und die Distanz zum Gegner angeben.

4.3 Umsetzung der Spielerschnittstelle

Die Umsetzung der Spielerschnittstelle stellt eine der wichtigsten Herausforderungen des Projekts dar, da sie es ermöglichen muss, dass Spieler ihren eigen Code schreiben können der zur Laufzeit kompiliert und ausgeführt werden muss. Der Code selber wird ausserhalb dieses Skripts gespeichert und mit der ID des Spielers, von dem es geschrieben wurde versehen. Mit der compile_dynamic_code funktion wird der Code kompiliert. GDScript erlaubt einem eine Skriptvariable zu erstellen, und in dieser den Source Code einzusetzen. Dem Skript wird dann das UserCode Objekt weitergegeben damit es Methoden ausführen kann. In der Regel ist dieser Ansatz unsicher und nicht empfohlen, da ein Spieler theoretisch so auf den Tree zugreifen könnte und Variablen ändern könnte, die er nicht ändern sollte, doch es ist der einfachste Ansatz und es ist hier nicht nötig ihn sicherer zu machen.

Der Spieler-Code muss eine run-Funktion enthalten, die vom CodeObject aufgerufen wird. Innerhalb dieser Funktion können die Spieler beliebige Logik implementieren. Die einzige Einschränkung ist, dass sie auf die vom CodeObject bereitgestellten Funktionen beschränkt sind und nicht direkt auf die Engine oder andere Nodes zugreifen können. Dies sorgt für Fairness und verhindert, dass Spieler das Spiel auf unvorhergesehene Weise manipulieren können.

4.4 Herausforderungen und Lösungsansätze

Literaturverzeichnis

Primärliteratur

SekundarVliteratur

Internetquellen

Einsatz von Künstlicher Intelligenz

Abbildungsverzeichnis

Deckblatt

---

1. Einleitung

1.1 Relevanz und Zielsetzung Bedeutung von Gamifizierung im Bildungskontext, Defizit bestehender Coding Games (fehlende Kompetitivität, kein Anreiz für Fortgeschrittene), Ziel der Facharbeit, Vorstellung der Fragestellung, kurzer Überblick über den Aufbau der Arbeit.

1.2 Verwandte Spiele und Inspiration Vorstellung von CodeCombat als Referenzbeispiel, weitere ähnliche Spiele, Ableitung der Anforderungen an das eigene Spiel.

---

2. Theoretische Grundlagen

2.1 Gamifizierung und Lernspiele Definition und Abgrenzung der Begriffe, Einsatz in Bildungskontexten, wissenschaftlicher Stand.

2.2 Motivation durch Wettbewerb Warum steigert Kompetitivität den Lerneffekt? Theoretische Einbettung, z.B. Selbstbestimmungstheorie oder Flow-Theorie.

2.3 Anforderungen an ein Programmier-Lernspiel Zugänglichkeit für Anfänger, Tiefe für Fortgeschrittene, Feedbackmechanismen, Lernziele.

---

3. Konzeption des Spiels

3.1 Spielidee und Designentscheidungen Begründung des kompetitiven Ansatzes, Vergleich verworfener Ideen mit der gewählten Lösung, Begründung des Weltraumthemas.

3.2 Spielmechaniken Ausrüstung des Raumschiffs, Triebwerke, Waffensysteme, Schild, Energiesystem. Verknüpfung mit den Lernzielen: Welche Programmierfähigkeiten werden konkret gefördert?

3.3 Schnittstelle für den Spieler Wie gibt der Spieler seinen Code ein? Wie wird er ausgeführt? Welche Befehle stehen zur Verfügung?

---

4. Implementierung

4.1 Technische Grundlagen Vorstellung und Begründung der Wahl der Godot Engine, grundlegende Strukturen und Konzepte.

4.2 Umsetzung der Spielmechaniken Implementierung von Bewegung, Waffensystemen und Energiemanagement.

4.3 Umsetzung der Spielerschnittstelle Technische Umsetzung der Code-Eingabe und -Ausführung durch den Spieler.

4.4 Herausforderungen und Lösungsansätze Technische Probleme während der Entwicklung und deren Lösung.

---

5. Evaluation des Lerneffekts

5.1 Methodik Wie und an wen wurde das Spiel getestet? Vorher-Nachher-Test, Beobachtung oder Befragung?

5.2 Ergebnisse Auswertung der Testergebnisse, Rückbezug auf die Fragestellung.

5.3 Kritische Reflexion Grenzen der Evaluation, Schwächen im Design oder der Umsetzung.

---

6. Fazit und Ausblick

Zusammenfassung der Ergebnisse, Beantwortung der Fragestellung, offene Fragen und Anregungen für Weiterentwicklungen.

Zur Struktur noch ein Hinweis: Kapitel 1 übernimmt die Funktion der Einleitung gemäß den Vorgaben – es orientiert den Leser über Thema, Fragestellung und Aufbau. Die Kapitel 2 bis 5 bilden den Hauptteil, Kapitel 6 den Schluss. Die Begriffe „Einleitung", „Hauptteil" und „Schluss" tauchen dabei bewusst nicht als Überschriften auf, wie es die Vorgaben empfehlen.