Projektdokumentation für "Helikopter und Raketen" ein JavaScript Spiel mit Phaser Framework

Samuel Altuntas

Inhalt

Projektplanung

Projektziel

Ziel war die Entwicklung eines 2D-Top-Down-Survival-Shooters, in dem der Spieler einen Helikopter frei über die 1250x720 Pixel große Spielfläche steuern kann. Zudem sollte man sich gegen drei verschiedene Gegnertypen behaupten und Punkte sammeln können, die dann im Game Over Screen ausgegeben werden. Zusätzlich sollen Wolken spawnen die den Schwierigkeitsgrad erhöhen in dem sie die Sicht einschränken. Das Spiel soll mit HTML, JavaScript und dem Phaser-Framework erstellt werden.

Ressourcenplanung

Für die Entwicklung des Projekts habe ich mein MacBook Air M1 aus dem Jahr 2020 genutzt.

Als Software verwendete ich Visual Studio Code, einen vielseitigen Code-Editor, der zahlreiche Erweiterungen für die JavaScript-Entwicklung bietet. JavaScript war die Hauptprogrammiersprache meines Projekts, ergänzt durch das Phaser Framework, das speziell für die Entwicklung von 2D-Spielen konzipiert ist und eine breite Palette an Funktionen bereitstellt.

Für die Versionskontrolle nutzte ich GitHub. Dies ermöglichte es mir, den Quellcode zu speichern, Änderungen nachzuverfolgen und das Projekt als öffentliches Repository zu veröffentlichen. Die Spielgrafiken und Assets erstellte und bearbeitete ich mit Adobe FireFly und Adobe Photoshop, die beide umfangreiche Tools für digitales Design und Bildbearbeitung bieten.

Zusätzlich setzte ich ChatGPT ein, das mir bei der Programmierung half, Fragen beantwortete und Unterstützung bei der Erstellung der Projektdokumentation bot. Durch diese Kombination aus leistungsfähiger Hardware und vielseitiger Software konnte ich eine optimale Entwicklungsumgebung schaffen, die den erfolgreichen Abschluss des Projekts "Helikopter und Raketen" ermöglichte.

Zeitplan

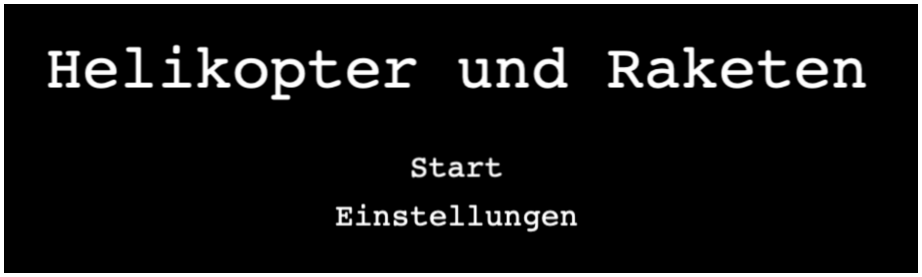
Für das Projekt "Helikopter und Raketen" habe ich eine strukturierte und effiziente Vorgehensweise gewählt, um die Entwicklungsarbeit über die Woche hinweg zu verteilen. Am Montag begann ich mit Brainstorming, Ideenfindung sowie der Planung und Konzeptionierung des Spiels, um eine klare Vision und ein solides Konzept zu entwickeln. Am Dienstag erstellte ich erste Prototypen und einen Proof of Concept, um die grundlegenden Spielmechaniken wie die Steuerung des Helikopters und das Schießen von Projektilen zu testen. Der Mittwoch war der Hauptarbeit gewidmet: Ich implementierte die Kernfunktionen des Spiels und erstellte erste Grafiken. Am Donnerstag fügte ich die letzten Features hinzu, führte intensive Tests durch, behebe alle gefundenen Bugs und erstellte die Projektdokumentation, um den gesamten Entwicklungsprozess und die erzielten Ergebnisse zusammenzufassen. Dieser strukturierte Ansatz ermöglichte es mir, das Projekt effizient und erfolgreich abzuschließen.

Entwicklung

Entwicklung der Benutzeroberfläche

StartScene:

Die erste Interaktion die der Spieler mit dem Spiel hat findet in der StartScene statt. Hier wird der Titel des Spiels prominent in der Mitte des Bildschirms angezeigt. Darunter befinden sich zwei interaktive Buttons: Ein Start-Button und ein Einstellungen-Button. Der Start-Button führt den Spieler direkt in die Hauptspielszene, während der Einstellungen-Button zur Einstellungsszene führt.



SettingsScene

Die Einstellungsszene ermöglicht es dem Spieler, den Hintergrund des Spiels zu ändern. Beim Laden dieser Szene werden zu erst die verfügbaren Hintergrundassets in der `preload`-Methode geladen und bietet dem Spieler drei Auswahlmöglichkeiten für verschiedene Hintergründe. Jeder Hintergrund wird durch ein kleines Vorschaubild dargestellt, das interaktiv ist. Wenn der Spieler auf eines dieser Bilder klickt, wird das ausgewählte Hintergrundbild im lokalen Speicher gespeichert, um es später im Spiel zu verwenden. Zusätzlich gibt es noch einen Zurück-Button, der den Spieler zurück zur StartScene bringt.

MainScene

Die MainScene ist das zentrale Element des Spiels "Helikopter und Raketen". Sie umfasst die Hauptlogik und das Gameplay, in dem der Spieler Helikopter steuert, Gegner bekämpft und Punkte sammelt. Hier werden alle wesentlichen Spielmechaniken implementiert.

GameOverScene

Die GameOver-Szene zeigt das Ende des Spiels an. Hier wird der endgültige Punktestand des Spielers und die gespielte Zeit angezeigt. Zusätzlich gibt es einen Neustart-Button, der es dem Spieler ermöglicht, das Spiel von vorne zu beginnen. Diese Szene wird durch die Methode `init` initialisiert, die die Endpunktzahl und die Spielzeit aus der vorherigen Szene übernimmt. In der `create`-Methode werden die Endpunktzahl und die Spielzeit angezeigt und der Neustart-Button wird erstellt.

Implementierung der Spielmechaniken

preload()

Die preload()-Methode ist dafür verantwortlich, alle notwendigen Ressourcen zu laden, bevor das Spiel startet. Dazu gehören Bilder und Sprite-Sheets für den Helikopter, die Gegner, die Projektile und die Hintergrundbilder. Durch die Nutzung der Phaser-Methode this.load.image und this.load.spritesheet werden diese Ressourcen in den Speicher geladen, sodass sie später im Spiel verwendet werden können.

Implementierung:

```
preload() {
    this.load.spritesheet('rotor', 'assets/rotor.png', { frameWidth: 50, frameHeight: 50 });
    this.load.image('player_up', 'assets/player_up.png');
    this.load.image('enemy_up', 'assets/enemy_up.png');
    this.load.image('bullet_up', 'assets/bullet_up.png');
    this.load.image('background1', 'assets/background1.png');
    // Weitere Ressourcen...
}
```

create()

Die `create()`-Funktion der MainScene initialisiert die Spielszene und richtet alle wesentlichen Elemente ein, um das Spiel zu starten. Zunächst wird das ausgewählte Hintergrundbild basierend auf gespeicherten Benutzereinstellungen oder einer Standardeinstellung geladen und angezeigt. Anschließend wird der Spieler als Sprite in der Mitte des Bildschirms platziert und so konfiguriert, dass er sich innerhalb der Spielfeldgrenzen bewegen kann.

Die Animation für den Rotor des Helikopters wird erstellt, indem eine Animationssequenz definiert wird, die die Rotorblätter dreht. Diese Animation wird auf ein Rotor-Sprite angewendet, das in Verbindung mit dem Spieler-Helikopter angezeigt wird, um eine realistische Bewegungsdarstellung zu erzeugen. Die Steuerung des Spielers erfolgt über die Pfeiltasten, die mit der Methode `createCursorKeys` von Phaser aktiviert werden.

Zusätzlich wird die Punktzahl des Spielers initial auf null gesetzt und oben links auf dem Bildschirm angezeigt. Die Anzahl der Leben des Spielers wird ebenfalls dargestellt, indem Herzsymbole neben der Punktanzeige platziert werden. Diese Symbole werden initialisiert und in einem Array gespeichert, um sie später bei Bedarf aktualisieren zu können.

Darüber hinaus wird ein Ereignis eingerichtet, das in regelmäßigen Abständen neue Gegner auf dem Spielfeld erscheinen lässt. Diese Gegner werden durch die `spawnEnemy()`-Methode erzeugt. Insgesamt stellt die `create()`-Funktion sicher, dass alle grundlegenden Elemente und Mechaniken des Spiels bereitgestellt und korrekt initialisiert sind, um ein nahtloses Spielerlebnis zu ermöglichen.

update()

Die `update()`-Funktion in der MainScene wird in jedem Frame des Spiels aufgerufen und ist für die kontinuierliche Aktualisierung der Spielszene verantwortlich. Zunächst wird die

Bewegung des Spielers basierend auf den Eingaben der Pfeiltasten gesteuert. Wenn die linke oder rechte Pfeiltaste gedrückt wird, ändert sich die horizontale Geschwindigkeit des Spielers entsprechend. Ebenso bewirken die obere und untere Pfeiltaste eine Änderung der vertikalen Geschwindigkeit. Um eine geschmeidigere Bewegung zu ermöglichen, wird die Geschwindigkeit allmählich reduziert, wenn keine Pfeiltaste gedrückt wird.

Die Funktion prüft auch, ob der Spieler die Leertaste drückt, um zu schießen. Wenn dies der Fall ist und das Schussintervall überschritten wurde, wird die Methode `shootBullet()` aufgerufen, um ein Projektil abzufeuern. Zudem wird die Position des Rotor-Sprites ständig aktualisiert, sodass es mit dem Spieler-Helikopter synchron bleibt.

Die Position und Blickrichtung des Spielers wird ebenfalls in jedem Frame angepasst. Je nachdem, welche Pfeiltasten gedrückt werden, ändert sich die Textur des Spielers, um die entsprechende Blickrichtung anzuzeigen. Dies trägt zur visuellen Rückmeldung und Immersion des Spielers bei.

Darüber hinaus bewegt die `update()`-Funktion alle aktiven Gegner in Richtung des Spielers. Für schießende Gegner wird überprüft, ob sie schießen können, und gegebenenfalls wird die Methode `shootFromEnemy()` aufgerufen.

Zuletzt aktualisiert die `update()`-Funktion den Spieltimer und die Punktzahlanzeige. Die vergangene Zeit wird berechnet und im entsprechenden Format angezeigt, während die Punktzahl basierend auf den Spieleraktionen aktualisiert wird.

shootBullet()

Die `shootBullet()`-Funktion feuert Projektile vom Spieler ab, wenn die Schusstaste gedrückt wird und das Schussintervall überschritten wurde. Ein neues Projektil wird an der aktuellen Position des Spielers erstellt, aktiviert und sichtbar gemacht. Seine Geschwindigkeit und Richtung werden basierend auf der aktuellen Blickrichtung des Spielers festgelegt: nach links, rechts, oben, unten oder diagonal. Die Projektiltextur wird entsprechend angepasst, um visuelle Rückmeldung zu geben. Diese Funktion gewährleistet eine präzise und reaktionsschnelle Schussmechanik im Spiel.

spawnEnemy()

Die `spawnEnemy()`-Funktion in "Helikopter und Raketen" ist dafür verantwortlich, neue Gegner auf dem Spielfeld zu erzeugen. Zunächst generiert die Funktion zufällige X- und Y-Koordinaten innerhalb der Spielfeldgrenzen. Um sicherzustellen, dass die Gegner nicht zu nahe am Spieler erscheinen, wird die Position angepasst, wenn sie sich innerhalb eines bestimmten Abstands zum Spieler befindet.

Anschließend wird zufällig einer von drei Gegnertypen ausgewählt: Standard-Gegner, schneller Gegner oder schießender Gegner. Basierend auf dem ausgewählten Typ wird der entsprechende Gegner erstellt und konfiguriert. Standard-Gegner bewegen sich mit normaler Geschwindigkeit, während schnelle Gegner eine höhere Geschwindigkeit haben. Schießende Gegner haben zusätzlich die Fähigkeit, regelmäßig Projektile in Richtung des

Spielers abzufeuern und erhalten ein animiertes Rotor-Sprite, das ihre Bewegung realistischer erscheinen lässt.

Alle Gegner werden schließlich so konfiguriert, dass sie innerhalb der Spielfeldgrenzen bleiben und sichtbar sind. Diese Funktion sorgt dafür, dass das Spiel dynamisch und herausfordernd bleibt, indem ständig neue Gegner mit unterschiedlichen Verhaltensweisen erscheinen.

updateEnemyDirection()

Die updateEnemyDirection(enemy, angle)-Funktion passt die Richtung des Gegners basierend auf dem Winkel (angle) zwischen dem Gegner und dem Spieler an. Abhängig vom Winkel wird die Textur des Gegners so geändert, dass er in die Richtung schaut, in die er sich bewegt. Wenn der Winkel beispielsweise in einem bestimmten Bereich liegt, zeigt der Gegner nach oben, unten, links, rechts oder diagonal. Diese Anpassung sorgt dafür, dass die Gegner immer korrekt in die Richtung schauen, in die sie sich bewegen, und trägt zur visuellen Konsistenz und Immersion des Spiels bei.

shootFromEnemy()

Die `shootFromEnemy()`-Funktion ermöglicht es gegnerischen Einheiten, Projektile auf den Spieler abzufeuern. Diese Funktion wird regelmäßig für schießende Gegner aufgerufen. Zunächst wird ein neues Projektil an der Position des Gegners erstellt und aktiviert. Die Richtung des Projektils wird anhand des Winkels zwischen dem Gegner und dem Spieler berechnet. Anschließend wird die Geschwindigkeit des Projektils so eingestellt, dass es sich in einer geraden Linie auf den Spieler zubewegt. Die Textur des Projektils wird basierend auf seiner Richtung angepasst, um eine visuelle Rückmeldung zu geben. Die Funktion stellt sicher, dass die Schüsse der Gegner präzise und dynamisch sind, was die Herausforderung für den Spieler erhöht.

hitEnemy()

Die `hitEnemy()`-Funktion wird aufgerufen, wenn ein Projektil des Spielers einen Gegner trifft. Diese Funktion deaktiviert sowohl das Projektil als auch den getroffenen Gegner, indem sie sie unsichtbar und inaktiv macht. Zudem wird die Physik des getroffenen Gegners und des Projektils deaktiviert, um Kollisionen zu verhindern.

Sobald ein Treffer registriert wird, erhöht sich die Punktzahl des Spielers um 100 Punkte, und die Punktanzeige wird entsprechend aktualisiert. Zusätzlich wird eine Explosion an der Position des getroffenen Gegners angezeigt, um den Treffer visuell darzustellen. Diese Explosion wird nach kurzer Zeit automatisch entfernt.

Falls der getroffene Gegner ein schießender Gegner ist, der über einen Rotor verfügt, wird auch dieser Rotor zerstört und aus dem Spiel entfernt. Durch diese Funktion wird das Trefferfeedback verbessert und die Spielererfahrung intensiviert.

hitEnemyBullet()

Die `hitEnemyBullet()`-Funktion wird aufgerufen, wenn ein Projektil des Spielers auf ein Projektil eines Gegners trifft. In dieser Funktion werden beide Projektile deaktiviert, indem sie unsichtbar und inaktiv gemacht werden und ihre Physik-Body-Komponenten deaktiviert werden, um weitere Kollisionen zu verhindern.

Sobald eine Kollision zwischen den beiden Projektilen erkannt wird, erhöht sich die Punktzahl des Spielers um 50 Punkte, und die Punktanzeige wird entsprechend aktualisiert. Zudem wird an der Position der Kollision eine Explosion angezeigt, um den Treffer visuell darzustellen. Diese Explosion wird nach kurzer Zeit automatisch entfernt.

Durch diese Funktion wird sichergestellt, dass Projektilkollisionen korrekt verarbeitet werden, was zur Dynamik und zum taktischen Element des Spiels beiträgt. Es ermöglicht dem Spieler, gegnerische Projektile zu neutralisieren und gleichzeitig zusätzliche Punkte zu sammeln.

playerHit()

Die `playerHit()`-Funktion wird aufgerufen, wenn ein Gegner den Spieler trifft. Diese Funktion reduziert die Anzahl der Leben des Spielers um eins. Wenn der Spieler ein Leben verliert, wird eines der Herzsymbole, die die verbleibenden Leben anzeigen, unsichtbar gemacht.

Der getroffene Gegner wird deaktiviert, indem er unsichtbar und inaktiv gemacht wird, und seine Physik-Body-Komponente wird deaktiviert. Falls der getroffene Gegner ein schießender Gegner ist, der über einen Rotor verfügt, wird auch dieser Rotor zerstört und aus dem Spiel entfernt.

Eine Explosion wird an der Position des Spielers angezeigt, um den Treffer visuell darzustellen. Diese Explosion wird nach kurzer Zeit automatisch entfernt.

Wenn die Leben des Spielers auf null reduziert sind, wird die Spielzeit seit Beginn berechnet und formatiert. Anschließend wird die Szene zur GameOver-Szene gewechselt, wobei die endgültige Punktzahl und die Spielzeit übergeben werden. Diese Funktion sorgt dafür, dass Treffer korrekt verarbeitet werden und der Übergang zum Spielende reibungslos erfolgt. Diese Funktionen zusammen gewährleisten ein dynamisches und herausforderndes

playerHitByBullet()

Die `playerHitByBullet()`-Funktion wird aufgerufen, wenn der Spieler von einem gegnerischen Projektil getroffen wird. Zunächst wird das Projektil deaktiviert, indem es unsichtbar und inaktiv gemacht wird und seine Physik-Body-Komponente deaktiviert wird, um weitere Kollisionen zu verhindern.

Bei einem Treffer wird die Anzahl der Leben des Spielers um eins reduziert. Entsprechend wird eines der Herzsymbole, die die verbleibenden Leben anzeigen, unsichtbar gemacht.

Eine Explosion wird an der Position des Spielers angezeigt, um den Treffer visuell darzustellen. Diese Explosion wird nach kurzer Zeit automatisch entfernt.

Wenn die Leben des Spielers auf null reduziert sind, wird die Spielzeit seit Beginn berechnet und formatiert. Anschließend wird die Szene zur GameOver-Szene gewechselt, wobei die endgültige Punktzahl und die Spielzeit übergeben werden. Diese Funktion sorgt dafür, dass Treffer durch gegnerische Projektile korrekt verarbeitet werden und der Übergang zum Spielende reibungslos erfolgt.

spawnItem()
Die `spawnItem()`-Funktion erzeugt regelmäßig sammelbare Gegenstände auf dem Spielfeld, wenn die Punktzahl des Spielers 2000 oder mehr erreicht hat. Diese Gegenstände erscheinen an zufälligen Positionen, die überprüft und angepasst werden, um sicherzustellen, dass sie nicht zu nah am Spieler spawnen. Der erzeugte Gegenstand wird aktiviert, sichtbar gemacht und so konfiguriert, dass er sich innerhalb der Spielfeldgrenzen bewegt. Diese Funktion fügt dem Spiel eine zusätzliche Herausforderung und Möglichkeit für den Spieler hinzu, Punkte zu sammeln.

collectItem()
Die `collectItem()`-Funktion in "Helikopter und Raketen" wird aufgerufen, wenn der Spieler einen sammelbaren Gegenstand aufnimmt. Der Gegenstand wird deaktiviert und unsichtbar gemacht. Anschließend wird die Anzahl der aktiven Gegner ermittelt. Alle aktiven Gegner werden deaktiviert und unsichtbar gemacht, was die Bedrohung für den Spieler kurzfristig beseitigt. Die Punktzahl des Spielers wird basierend auf der Anzahl der deaktivierten Gegner um einen festen Betrag erhöht, und die Punktanzeige wird entsprechend aktualisiert. Diese Funktion belohnt den Spieler für das Einsammeln von Gegenständen und bietet eine strategische Möglichkeit, Gegner zu neutralisieren.

spawnCloud()
Die `spawnCloud()`-Funktion erzeugt Wolken, die langsam über den Bildschirm ziehen. Diese Wolken werden in regelmäßigen Abständen an zufälligen X-Koordinaten außerhalb des oberen Bildschirmrands erstellt. Es gibt drei verschiedene Wolkentypen mit unterschiedlichen Transparenzstufen, die die Vielfalt und visuelle Tiefe im Spiel erhöhen.

Sobald die Wolken erstellt sind, bewegen sie sich langsam vertikal über den Bildschirm. Eine Tween-Animation wird verwendet, um die Bewegung der Wolken zu steuern, die nach einer bestimmten Zeitspanne automatisch zerstört werden, sobald sie den unteren Bildschirmrand erreichen. Diese Funktion trägt zur dynamischen und visuellen Atmosphäre des Spiels bei, indem sie einen beweglichen Hintergrundeffekt erzeugt.

updateSpawnDelay()
Die `updateSpawnDelay()`-Funktion in "Helikopter und Raketen" passt die Spawn-Intervalle der Gegner basierend auf der aktuellen Punktzahl des Spielers an. Je höher die Punktzahl, desto kürzer wird das Intervall, um den Schwierigkeitsgrad progressiv zu erhöhen.

Zu Beginn prüft die Funktion die Punktzahl des Spielers und legt ein neues Spawn-Intervall fest. Bei Punktzahlen unter 1000 beträgt das Intervall 5000 Millisekunden. Mit zunehmender

Punktzahl wird das Intervall schrittweise verkürzt: bei 1000 bis 1999 Punkten auf 4000 Millisekunden, bei 2000 bis 2999 Punkten auf 3000 Millisekunden, bei 3000 bis 3999 Punkten auf 2000 Millisekunden und bei Punktzahlen über 4000 auf 1000 Millisekunden.

Wenn das neue Intervall vom aktuellen Intervall abweicht, wird das Spawn-Event der Gegner mit dem neuen Intervall zurückgesetzt. Diese Anpassung sorgt dafür, dass das Spiel kontinuierlich anspruchsvoller wird und der Spieler immer wieder neuen Herausforderungen begegnet.

Gestaltung der Grafiken und Assets

Für mein Projekt "Helikopter und Raketen" habe ich großen Wert auf selbstgestaltete Grafiken und Assets gelegt. Die Erstellung dieser Elemente erforderte sorgfältige Planung, Kreativität und den Einsatz verschiedener Software-Tools.

Tools und Software
Für die Erstellung der Grafiken nutzte ich Adobe Firefly und Adobe Photoshop. Adobe Firefly ermöglichte es mir, erste Skizzen und Designs zu erstellen, während ich mit Adobe Photoshop die Details der Grafiken ausarbeitete. Photoshop bietet eine Vielzahl von Werkzeugen und Funktionen, die es mir ermöglichten, Sprites zu erstellen.

Erstellung der Sprites
Für die Helikopter erstellte ich mehrere Ansichten, um die Bewegungsrichtungen (nach oben, unten, links, rechts und diagonal) im Spiel darzustellen. Jeder Helikopter-Sprite wurde in verschiedenen Positionen erstellt, um die Animation der Rotoren zu simulieren. Dies verleiht dem Spiel eine dynamischere und realistischere Darstellung der Helikopterbewegungen.

Für die Gegner-Sprites entschied ich mich für verschiedene Designs, um Abwechslung ins Spiel zu bringen. Es gibt einfache Gegner, schnelle Gegner und schießende Gegner, die alle unterschiedliche Sprites und Animationen haben. Diese Vielfalt trägt dazu bei, das Spielerlebnis spannend und herausfordernd zu gestalten.

Hintergründe und Umgebungen
Die Hintergründe wurden ebenfalls in Adobe Photoshop gestaltet. Ich erstellte mehrere Hintergrundbilder, die der Spieler in den Einstellungen auswählen kann. Diese Hintergründe sollten sowohl optisch ansprechend als auch thematisch passend zum Spiel sein. Sie wurden in verschiedenen Farbschemata erstellt, um dem Spieler die Möglichkeit zu geben, die visuelle Atmosphäre des Spiels nach seinen Vorlieben anzupassen.

Animationen und Effekte
Zusätzlich zu den statischen Sprites erstellte ich Animationen für verschiedene Spielereignisse. Die Rotoren der Helikopter und die Bewegung der Raketen wurden animiert, um das Spiel lebendiger zu gestalten. Auch für Explosionen und andere Effekte nutzte ich die Animationswerkzeuge in Photoshop, um nahtlose und beeindruckende Animationen zu erzeugen.

Implementierung der Assets im Spiel
Nach der Erstellung der Grafiken und Animationen wurden diese in das Spiel integriert. Dies erforderte das Laden der Assets in den entsprechenden Szenen und die Implementierung der Logik zur Steuerung der Animationen und Effekte. Dabei stellte ich sicher, dass die Grafiken korrekt skaliert und positioniert wurden, um ein konsistentes und professionelles Erscheinungsbild zu gewährleisten.

Testing und Finalisierung

Das Testing und die Finalisierung des Spiels "Helikopter und Raketen" waren entscheidende Phasen, um sicherzustellen, dass das Spiel reibungslos funktioniert und ein optimales Spielerlebnis bietet. Dieser Prozess umfasste mehrere Schritte, darunter das Identifizieren und Beheben von Bugs, die Optimierung der Spielmechanik und die abschließende Verfeinerung des gesamten Spiels.

Testphase

In der Testphase wurde das Spiel intensiv gespielt, um alle Funktionen und Spielmechaniken auf ihre Funktionalität zu überprüfen. Dabei wurden verschiedene Aspekte des Spiels getestet:

- **Gameplay-Mechanik**: Es wurde sichergestellt, dass die Steuerung des Helikopters flüssig und reaktionsschnell ist. Die Bewegung des Helikopters in alle Richtungen, das Schießen von Projektilen und die Kollisionserkennung wurden gründlich überprüft.
- **Gegnerverhalten**: Das Verhalten der Gegner wurde getestet, um sicherzustellen, dass sie sich korrekt auf den Spieler zubewegen und ihre Angriffsmuster ordnungsgemäß funktionieren. Dabei wurde auch überprüft, ob die Gegner korrekt gespawnt werden und ihre jeweiligen Aktionen ausführen.
- **Punkte- und Lebenssystem**: Es wurde geprüft, ob das Punktesystem ordnungsgemäß funktioniert und die Punkte korrekt berechnet und angezeigt werden. Das Lebenssystem des Spielers wurde ebenfalls getestet, um sicherzustellen, dass der Spielerleben korrekt abgezogen werden und das Spiel endet, wenn alle Leben aufgebraucht sind.
- **Visuelle Effekte**: Die visuelle Darstellung, einschließlich der Animationen und Explosionen wurden getestet, um sicherzustellen, dass sie korrekt abgespielt und synchronisiert werden.

Bugfixing

Während der Testphase wurden verschiedene Bugs identifiziert und behoben. Dazu gehörten:

- Kollisionsprobleme: Einige Probleme bei der Kollisionserkennung zwischen dem Helikopter und den Gegnern sowie den Projektilen wurden behoben. Es wurde sichergestellt, dass die Kollisionen präzise erkannt werden und die entsprechenden Aktionen (wie Schaden oder Zerstörung) ausgelöst werden.
- Performance-Optimierung: Es wurden Optimierungen vorgenommen, um die Performance des Spiels zu verbessern. Dazu gehörte die Reduzierung der Ladezeiten und die Optimierung der Ressourcenverwendung, um eine flüssige Spielerfahrung zu gewährleisten.
- Fehlerhafte Animationen: Einige Animationen wurden angepasst, um sicherzustellen, dass sie korrekt abgespielt werden und keine visuellen Fehler auftreten.

Finalisierung

Nach dem Bugfixing wurde das Spiel finalisiert. In dieser Phase wurden die letzten Anpassungen und Verfeinerungen vorgenommen, um das Spiel zu optimieren und ein abgeschlossenes Produkt zu liefern.

- **Feinschliff der Grafiken**: Die Grafiken und Animationen wurden verfeinert, um sicherzustellen, dass sie ansprechend und konsistent sind. Kleine visuelle Details wurden angepasst, um das Gesamtbild zu verbessern.
-
- **Dokumentation**: Eine umfassende Dokumentation des Projekts wurde erstellt, die den gesamten Entwicklungsprozess, die verwendeten Technologien und die durchgeführten Tests beschreibt. Dies stellte sicher, dass das Projekt nachvollziehbar und für zukünftige Entwicklungen oder Anpassungen gut dokumentiert ist.

Abschließend wurde das Spiel erneut getestet, um sicherzustellen, dass alle Änderungen und Verbesserungen erfolgreich implementiert wurden und keine neuen Bugs aufgetreten sind. Mit der finalen Überprüfung und Freigabe war das Projekt abgeschlossen und das Spiel "Helikopter und Raketen" bereit für die Veröffentlichung.