Softwareprojekt 24/25 - Einzel
Institut für Softwaretechnik und Programmiersprachen
Sabrina Böhm, Valentin Kolb, Matthias Tichy

universität ulm

Lastenheft **Starfighter Alliance**

1 Spielidee und Hintergrund

Dieses Lastenheft beschreibt die Anforderungen an das Endlos-Flug-Spiel *Starfighter Alliance*, welches in diesem Wintersemester (WiSe 24/25) entwickelt werden soll.

Die Rebellen-Allianz muss durch einen Meteoritengürtel fliegen, um vor der großen Macht des Imperiums zu fliehen. Dabei stehen sich nun ein Starfighter und eine Horde an Meteoriten gegenüber. Könnt ihr mehr Meteoriten ausweichen als ein Yedi oder endet ihr frühzeitig wie ein Stormtrooper?

In diesem Dokument werden die genaue Funktionsweise sowie die Implementierungsrichtlinien beschrieben.

2 Technische Rahmenbedingungen

Starfighter Alliance ist ein lokales Single-Player-Game. Die Aufgabe für das Einzelprojekt besteht darin, die Logik und die Nutzeroberfläche zum Spiel *Starfighter Alliance* zu implementieren, sodass alle geforderten Funktionalitäten des Spiels umgesetzt sind. Dabei muss ein vorgegebener Prozess eingehalten werden.

2.1 Sprachen und Technologien

Die Anwendungssprache kann Deutsch oder Englisch sein. Die Implementierungssprache und die Dokumentationssprache muss Englisch sein.

Für das Spiel werden von uns drei Programmiervorlagen in verschiedenen Programmiersprachen zur Verfügung gestellt: Java, Unity und Python. Dabei ist jeder Studentin / jedem Studenten überlassen, ob sie / er eine dieser Vorlagen in Anspruch nimmt oder eine von Start an eigene Implementierung umsetzt. Nach Absprache mit der tutorierenden Person sind auch andere Frameworks und Programmiersprachen erlaubt.

2.2 Plattformen

Die Anwendung muss mindestens auf einer der nachfolgenden Plattformen lauffähig sein:

- Windows 10 / 11
- Linux
- MacOS

3 Spielregeln

Starfighter Alliance ist ein Endlos-Flug-Spiel bei dem man mit einem Raumschiff durch das Weltall fliegt und Hindernissen (Meteroiten) ausweichen muss. Das Raumschiff kann sich dabei in alle Richtungen (links, rechts, hoch und runter) bewegen. Es kann sich jedoch nicht drehen. Außerdem hat das Raumschiff eine gewisse Anzahl an Leben, welche abgezogen werden beim Zusammenprall mit Hindernissen. Die Hindernisse bewegen sich in regelmäßigen Abständen (Wellen) von oben nach unten, wobei sie am oberen Bildrand auftauchen und am unteren Bildrand verschwinden. Hindernisse können vom Raumschiff zerstört werden und sobald dies geschehen ist verschwindet es vorzeitig. Das Raumschiff bekommt pro ausgewichener Welle an Hindernissen Punkte. Wenn das Raumschiff mit einem Hindernis kolidiert verliert das Raumschiff ein Leben. Das Ziel des Spiels ist es, so viele Punkte wie möglich zu sammeln, bis man keine Leben mehr hat.

3.1 Weltraum

Der Weltraum stellt den Hintergrund des Spiels dar. Der Weltraum kann vom Raumschiff nicht verlassen werden. Hindernisse bewegen sich immer senkrecht vom oberen Rand des Weltraumes zum unteren Rand. Das Raumschiff und die Hindernisse sind an die Größe des Weltraumes angepasst (siehe 3.2 und 3.5).

3.2 Raumschiff

Das Raumschiff ist ein quadratisches Objekt, das im Weltraum bewegt werden muss. Die Größe des Raumschiffs muss dem Weltraum angepasst sein und sollte ca. 1/10 der Breite des Weltraumes betragen. Bevor das Spiel beginnt soll aus verschiedenen Raumschiffen ausgewählt werden können. Beispielsweise soll aus mindestens folgenden Raumschiffen gewählt werden:

- Y-Sternenjäger
- X-Sternenjäger

- Millenium Falke

Die Raumschiffe haben jeweils eine Bewegungsgeschwindigkeit und eine Schusskraft. Zum Beispiel ist ein Y-Sternenjäger ein relativ schnelles Raumschiff, dass jedoch eine schwache Kanone gegen Hindernisse besitzt. Diese Werte sollen beim Auswählen der Raumschiffe dem Spieler / der Spielerin angezeigt werden.

3.3 Steuerung

Mittels den Pfeiltasten (oder W,A,S,D) soll das Raumschiff nach oben, unten oder zur Seite bewegt werden können. Das Raumschiff bewegt sich immer mit konstantem Tempo und stoppt sobald keine Richtungstaste mehr betätigt wird. Hierbei soll die Bewegungsgeschwindigkeits-Eigenschaft der Raumschiffe einbezogen werden, sodass Raumschiffe mit einem höheren Wert schneller fliegen können.

3.4 Schüsse

Durch Drücken der Leertaste kann das Raumschiff einen Schuss abfeuern. Die Schüsse haben eine festgelegte Geschwindigkeit und können Hindernisse zerstören. Ein Schuss fliegt dabei vom oberen Rand des Raumschiffs mittig aus der Abschussstelle senkrecht nach oben bis es ein Hindernis trifft oder aus dem Bild verschwindet. Sobald ein Schuss ein Hindernis trifft, verliert das Hindernis eine bestimmte Anzahl an Leben, abhängig von der Schusskraft des Raumschiffs. Ein Schuss kann maximal ein Hindernis treffen und löst sich daher nach einem Treffer auf. Ein Schuss sollte nicht ständig ausgeführt werden können, deshalb soll durch einen Timer festgelegt werden, wie viel Zeit vergehen muss um den nächsten Schuss abfeuern zu können.

3.5 Hindernisse

Ein Hindernis hat eine bestimmte Geschwindigkeit, Größe und eine gewisse Anzahl an Leben. Durch Schüsse können Hindernisse Leben verlieren und zerstört werden, sobald sie keine Leben mehr besitzen. Die Anzahl der Leben hängt dabei von der Größe des Hindernisses ab (größere Hindernisse haben mehr Leben). Hindernisse sollen als Meteoriten oder auch wahlweise als andere Weltraumelemente dargestellt werden.

Hindernisse werden immer in Wellen generiert. Eine Welle bezeichnet das Erscheinen von Hindernissen auf einer Höhe, die sich gemeinsam mit der gleichen Geschwindigkeit senkrecht von oben nach unten bewegen. Pro Welle darf eine beliebige Anzahl an Hindernissen generiert werden, solange mindestens die 1,5-fache Breite des Raumschiffs als Weg für das Raumschiff irgendwo in der Welle frei bleibt. Die Höhe und Breite der Hindernisse sind variable, jedoch darf die Höhe der Hindernisse pro Welle nicht

unterschiedlich sein. Zwischen zwei Hinderniswellen muss mindestens 3 mal die Höhe des Raumschiffs als Platz frei gelassen werden. Hindernisse, die am unteren Rand den Weltraum verlassen, werden automatisch zerstört und verschwinden.

Damit das Spiel über die Zeit anspruchsvoller wird, sollen die Geschwindigkeit und die Anzahl der Hindernisse über die Zeit leicht ansteigen.

4 Spielablauf

Beim Start des Spiels wechselt die Spielerin / der Spieler von der Start-Ansicht zur Raumschiffauswahl-Ansicht. Nach der Auswahl des Raumschiffs wird das Raumschiff in der Spiel-Ansicht am unteren Bildschirmrand mittig positioniert. Durch das Drücken eines Startknopfes beginnt ein sichtbarer Countdown, der von 3 auf 0 herunterzählt. Nach Ablauf des Countdowns ertönt ein Startsignal, das entweder als Ton oder durch den Text „Start“ angezeigt wird.

Nach dem Startsignal kann das Raumschiff gesteuert werden und die Hinderniswellen beginnen. Ein Zähler, der die Anzahl der Wellen anzeigt, sowie eine Punkteanzeige sollten am Bildschirmrand sichtbar sein. Wenn eine Welle von Hindernissen den unteren Bildschirmrand erreicht, erhöhen sich die Punkte. Wenn eine Welle komplett zerstört wurde gibt es mehr Punkte.

Wenn das Raumschiff mehrfach getroffen wird und somit irgendwann keine Leben mehr übrig hat, wechselt das Spiel in die Spielende-Ansicht. Auf dieser werden Statistiken des Spiels angezeigt, sowie ein Button der die Spielenden zum Start zurück bringt.

4.1 Ansichten

Folgende Ansichten, auch Screens genannt, müssen mindestens implementiert werden.

4.1.1 Start Screen

Der Start Screen (Start-Ansicht) ist der erste Bildschirm, den der Spieler / die Spielerin nach dem Starten des Spiels sieht. Durch Buttons soll man einfach navigieren können zwischen dem Starten des Spiels, Laden einer Konfigurationsdatei und Beenden des Spiels. Dieser Screen soll ansprechende Grafiken enthalten, um das Interesse der Spielenden zu wecken. Das Bild der Startseite dient als ein erster Eindruck des Spiels - ihr dürft euch hier natürlich gestalterisch austoben.

4.1.2 Spaceship Selection Screen

Der Spaceship Screen (Raumschiffwahl-Ansicht) ermöglicht es den Spielenden, ein Raumschiff für das Spiel auszuwählen. Eine Liste von verfügbaren Raumschiffen wird angezeigt, jedes mit einem Vorschaubild, der Geschwindigkeitseigenschaft und Schusseigenschaft. Durch Anklicken des gewünschten Raumschiffs wird in den Game Screen gewechselt.

4.1.3 Game Screen

Der Game Screen (Spiel-Ansicht) ist der Hauptspielbereich, in dem das eigentliche Spiel stattfindet. Er enthält einen Weltraum mit einem Raumschiff und auftauchenden Hindernissen. Während des Spiels sollen die ganze Zeit einige Informationen sichtbar sein, wie zum Beispiel die Anzahl der bisher überstandenen Wellen, die gesammelten Punkte und die derzeitigen Leben des Raumschiffs.

Außerdem soll es eine Möglichkeit geben, das Spiel zu pausieren. Dies kann über einen weiteren Screen oder mit einem Popup umgesetzt werden. Die Pause soll auch wieder beendet werden können, z.B. durch einen Button.

4.1.4 End Screen

Der End Screen (Spielende-Ansicht) wird angezeigt, wenn das Spiel zu Ende ist. Dieser zeigt das Raumschiff, die Anzahl der Wellen und die Punkteanzahl an. Zusätzlich zeigt er die besten 10 Ergebnisse des eigenen Highscores an, welcher in einer extra Datei gespeichert wird. Hier soll das aktuelle Ergebnis hervorgehoben werden, sodass man sieht auf ob man in den besten 10 Versuchen gelandet ist oder nicht. Außerdem soll durch einen Butten zum Start Screen zurückgekehrt werden können, um den Spielenden zu ermöglich das Spiel erneut zu spielen.

5 Highscore (optional mit Datenbank)

Nach jedem Spiel soll der Score des Spiels in einer Datei (z.B. JSON) lokal gespeichert werden. Dabei sollen mindestens die Anzahl an Wellen und die Punkte des aktuellen Spieldurchlaufs angezeigt werden. Optional kann man auch weitere Elemente in der Datenbank speichern, wie zum Beispiel den Raumschiff-Typ und einen Spielenden-Namen.

6 Konfigurationsdatei

Am Anfang des Spiels soll im Start Screen eine JSON Datei als Konfigurationsdatei ausgelesen werden. In dieser Konfigurationsdatei werden Eigenschaften für das Spiel festgelegt, wie die Anzahl der Leben des Raumschiffs und die Eigenschaften der Raumschiffe. Zusätzlich kann die Konfigurationsdatei mit weiteren Optionen ausgestattet werden, wie z.B. dem Leben der Hindernisse. Ein unvollständiges Beispiel für eine solche Datei könnte wie folgt aussehen:

```
{
        "lives": 3,
        "spaceships": [
                {
                        "name": "X-Fighter",
                        "speed": 3,
                        "shotPower": 10
                }
        ]
}
```

Die Anzahl an Leben des Raumschiffs wird hier durch *lives* festgelegt. *spaceships* bestimmt die wählbaren Raumschiffe, sowie die Eigenschaften der Raumschiffe, die durch *name*, *speed* und *shotPower* festgelegt werden.

7 Server für die Farbe des Raumschiffs

Um das Verständnis von Client-Server-Anwendungen zu verstärken, wird bevor das Spiel startet ein Server benutzt, um die Farbe des Raumschiffs festzulegen.

Auf Anfrage an den Server wählt der Server randomisiert eine von neun Farben aus und schickt diese dem Client zurück.

Sie können diesen Server unter der URL `softwaregrund.pro/jekt/ws/color` erreichen. Der Server akzeptiert Verbindungsanfragen für *ws://* unter dem Port 80 und für *wss://* unter dem Port 443.

Der Server arbeitet nach dem Websocket-Protokoll. Sie können eine neue Verbindung herstellen und anschließend die Farbe für das Raumschiff anfragen.

Unter der URL `https://softwaregrund.pro/jekt/` kann der Server über eine WebUI getestet werden.

7.1 Anfragen an den Server

Unter Nutzung einer JSON-Datei sollt ihr am Server anfragen, welche Farbe das Raumschiff im Spiel haben soll. Das vorgegebene JSON-Schema definiert die Struktur der Anfrage.

7.1.1 JSON Schema

```
{
        "$schema": "http://json-schema.org/draft-07/schema#",
        "type": "object",
        "properties": {
                "messageType": {
                        "type": "String"
                }
        },
        "required": ["messageType"]
}
```

Als messageType ist dabei nur die Nachricht des Types *COLOR_REQUEST* erlaubt.

7.1.2 Beispiel

```
{
        "messageType": "COLOR_REQUEST"
}
```

7.2 Antwort des Servers

Im Falle einer erfolgreichen Anfrage ist der Antwort *Code* 200 und das *data*-Feld enthält einen String mit der gewünschten Farbe.

Falls die Anfrage fehlgeschlagen ist, ist der *Code* 400 für illegale Anfrage und 500 im Falle eines internen Fehlers im Server.

7.2.1 JSON Schema Erfolg

```
{
        "$schema": "http://json-schema.org/draft-07/schema#",
        "type": "object",
        "properties": {
                "color": {
                        "type": "String"
                },
                "code": {
                        "const": 200
                }
        },
        "required": ["color", "code"]
}
```

Folgende Farben sind dabei *RED*, *BLUE*, *YELLOW*, *ORANGE*, *GREEN*, *VIOLET*, *WHITE*, *BLACK* und *TRANSPARENT*.

7.2.2 JSON Schema Fehler

```
{
        "$schema": "http://json-schema.org/draft-07/schema#",
        "type": "object",
        "properties": {
                "error": {
                        "type": "string"
                },
                "code": {
                        "enum": [400, 500]
                }
        },
        "required": ["error", "code"]
}
```

7.2.3 Beispiel Erfolg

```
{
        "color": "RED",
        "code" : 200
}
```

7.2.4 Beispiel Fehler

```
{
        "error" : "wrong messagetype",
        "code" : 400
}
```

8 (Optionale) Weitere Features

Aus folgenden Anforderungen muss **mindestens eine** umgesetzt werden. Alle anderen sind optional und müssen nicht zwangsweise umgesetzt werden. In Absprache mit der tutorierenden Person darf das Spiel darüber hinaus noch weiter angepasst und erweitert werden.

- **Nutzen einer Datenbank** für das Speichern des Highscores. Zur Speicherung der Daten empfehlen wir eine SQLite Datenbank zu verwenden, jedoch sind auch andere Datenbanken zulässig solange die Highscores außerhalb des Spiels gespeichert werden.
- **Weitere Objekte:** Es können noch weitere Hindernisse im Weltraum hinzugefügt werden, die zum Beispiel zusätzliche Punkte beim Zerstören bringen oder nicht nur senkrecht fliegen.
- **Soundeffekte:** Gebt eurem Spiel ein bisschen mehr Tiefe und baut Soundeffekte für Schüsse oder das Zerstören von Hindernissen ein.
- **Upgrades und Boni:** Durch Upgrades oder Boni könnte sich euer Raumschiff über den Verlauf des Spiels weiterentwickeln und dadurch schneller werden oder stärkere Schüsse abfeuern.
- **3D Modus:** Das Spiel kann statt in 2D in 3D gespielt werden (oder sogar in First-Person-View oder Third-Person-View).
- **Gegnerische Raumschiffe:** Um die Schwierigkeit des Spiels zu heben, können gegnerische Raumschiffe hinzugefügt werden, die probieren euch abzuschießen.

9 Vorgaben zum Entwicklungsprozess

Neben den Anforderungen an das Spiel *Starfighter Alliance* müssen folgende Punkte zum Bestehen des Einzelprojekts erfüllt sein:

- Die Implementierung muss in Einzelarbeit erfolgen. Jegliche Art von Plagiats- und Täuschungsversuchen wird mit dem Nicht-Bestehen des Einzelprojekts gewertet und kann gegebenenfalls weitere Sanktionen zur Folge haben.
- Das Projekt muss in einem bereitgestellten Git Repository auf `gitlab.uni-ulm.de` entwickelt und versioniert werden, sodass der Entwicklungsprozess für die tutorierende Person nachvollziehbar ist.
- Der Quellcode muss sinnvoll und ausführlich dokumentiert werden.
- Der gesamte Quellcode muss von der Studentin / dem Studenten verstanden und erklärt werden können.
- Das Projekt muss einem sinnvollen Workflow folgen, der die Nutzung von Gitlab Issues sowie sinnvollen Commits und Commit-Nachrichten beinhaltet.
- Die Abnahme erfolgt über die tutorierenden Person, die gezielt Fragen zum Quellcode stellen wird. Der / die Studierende muss in der Lage sein, diese Fragen zu beantworten.
- Die Blätter 1 bis 5 müssen erfolgreich bestanden werden.

Beim Nicht-Einhalten dieser Vorgaben sind wir gezwungen, Studierende durchfallen zu lassen. Eine erfolgreiche Abnahme des Einzelprojekts ist Voraussetzung zur Teilnahme am zweiten Teil des Softwareprojekts, dem Gruppenprojekt.