Dokumentation- Projekttage 2020: Space Invaders

Projektbezeichnung	Projekttage 2020 – Space Invaders
Ausbildungsberuf	Fachinformatiker für Anwendungsentwicklung
Projektgruppe	Patrick Matern, Monique Vanessa Ostermann, Erik Elges, Joy Webster
Bearbeitungszeitraum	20.04.2020 bis 21.04.2020

Inhalt

Abbildungsverzeichnis

Einführung

Das vorliegende Dokument beschreibt die Umsetzung einer objektorientierten Space Invaders-Anwendung, die im Rahmen eines Schulprojektes in der Ausbildung zum Fachinformatiker an dem Carl-Severing-Berufskollege entwickelt wurde. In diesem Dokument sollen die Umsetzung und Funktionalität des Projektes genauer erläutert werden.

Dazu dienen die Betrachtung der einzelnen Klassen des Projektes, eine Problemanalyse und eine Reflektion hinsichtlich des zu Beginn erstellten Pflichtenheftes.

Beschreibung der Anwendung

Die Anwendung wurde objektorientiert in Java programmiert und soll die Grundfunktionalität des Computerspiels Space Invaders realisieren. Für die Grafische Umsetzung wurde Switch verwendet. Insgesamt besteht die Anwendung aus 11 Klassen und einem Interface, das zur projektweiten Bereitstellung von Einstellungsparametern dient.

Zur optischen Aufwertung wurde neben dem Spiel auch noch ein Start- und End-Bildschirm implementiert. Die Anwendung verfolgt einen immer gleichen Ablauf, der in der Abbildung 1 gezeigt wird. Solange die Anwendung nicht beendet wird, wiederholt sich dieser Ablauf.

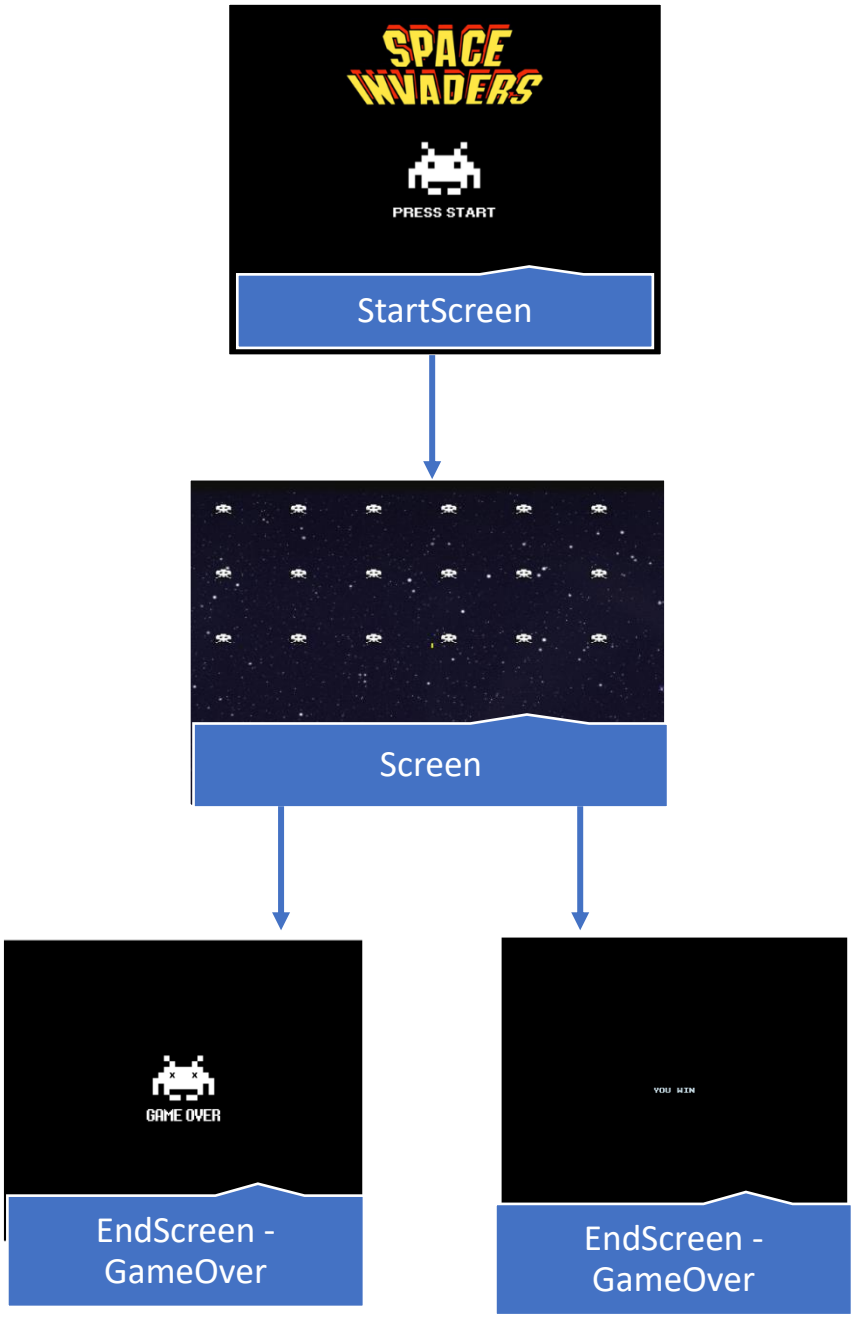

Abbildung 1: Ablauf der Anwendung

Nachfolgend sollen die einzelnen Klassen der Anwendung und deren Funktionalität und Umsetzung genauer erläutert werden.

Config.java

Hierbei handelt es sich um ein Interface, das im Projekt als Konfigurations-Datei dient. Wichtige Parameter für andere Klassen werden hier in einer Datei gespeichert. Dadurch stehen die Parameter projektweit zur Verfügung. Weiterhin dient das Interface der Übersicht und schnellen Anpassung, müssen Parameter geändert werden, muss dies nur über das Interface erfolgen. Unteranderem werden hier Parameter wie die Breite und Höhe des Spielfeldes gesetzt, die Bewegungsgeschwindigkeit, aber auch Hintergrundbild und Icons für die Spielfiguren.

Aus Gründen der Übersicht wurden die Parameter nach Spielelementen geordnet und mit Kommentaren versehen. Die nachfolgende Abbildung zeigt einen Teil der Strukturierung

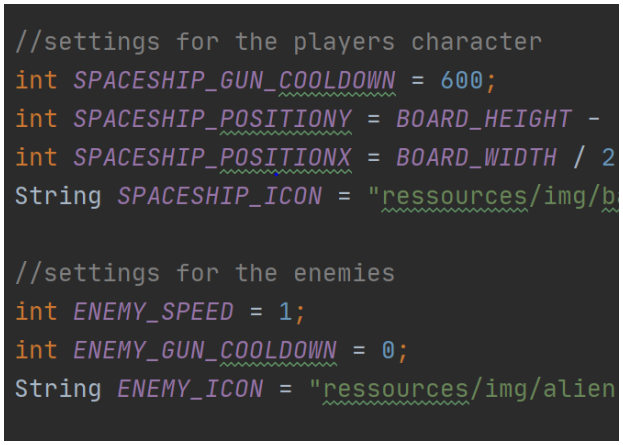

Abbildung 2: Struktur Config.java

Ein weiterer Vorteil von Config.java ist, dass dadurch auch der Grundstein für eine spielergesteuerte Anpassung des Spieles gelegt ist. Bei beispielsweise späterer Anpassung des Schwierigkeitsgrades oder Bildschirmgröße, müssen nur die entsprechenden Parameter im Interface angepasst werden.

Main.java

Main.java beinhaltet die main-Methode, die die Anwendung startet. Hier wird ein neues Game-Objekt erstellt und anschließend über die zugehörige Methode des Objektes run() gestartet.

Game.java

Aus der Klasse Game können Objekte erzeugt werden, die alle relevanten Elemente für ein Spiel enthalten. Als Attribute werden hier Start-, Spiel- & End-Screen gesetzt, die dazu dienen den entsprechenden Bildschirm für den jeweiligen Spielstatus zu zeigen. Zum Anzeigen der

Bildschirme/Screens wird JFrame benutzt. Beim Start der Anwendung wird über den Konstruktor der Klasse ein StartScreen-Objekt angezeigt.

Für den Spielstatus selbst besitzt die Klasse ebenfalls ein eigenes statisches Attribut.

Die wichtigste Methode der Klasse ist run(). Mit dieser wird der Ablauf des Spiels realisiert und entschieden, welcher Bildschirm gerade angezeigt werden muss. Dazu wird das Attribut gameStatus in einer while-Schleife ständig überprüft. Abbildung 3 zeigt diese while-Schleife. Ist das Spiel noch nicht gestartet und die Methode getStarted() von StartScreen liefert true zurück, wird StartScreen ausgeblendet und Screen eingeblendet. Der Gamestatus wird dann aus RUNNING gesetzt.

Signalisiert das Attribut gameStatus, dass das Spiel beendet ist, in dem er den Wert WON oder LOST hat, wird die entsprechende Ausgabe des EndScreens über die Methode setEndText() gesetzt. Danach wird EndScreen eingeblendet und Screen zerstört. Das Programm wartet drei Sekunden, ehe die while-Schleife unter dieser Bedingung durchbrochen wird und das Game-Objekt am Ende der Methode zurückgesetzt.

```
while (true) {      //loops run till game is won or lost
    try {
        //if game has not started & start Button is pressed
        if(gameStatus == GameStatus.NOT_STARTED && startScreen.getStarted()) {
            startScreen.setVisible(false);
            screen.setVisible(true);
            frame.add(screen);                      //show actual game screen
            frame.remove(startScreen);
            frame.pack();
            Thread.sleep( millis: 1000);
            screen.start();
            screen.grabFocus();
            setGameStatus(GameStatus.RUNNING);          //change status
            //game has ended
        } else if (gameStatus == GameStatus.WON || gameStatus == GameStatus.LOST) {
            endScreen.setEndLabel(gameStatus == GameStatus.WON);    //choose which label to display
            screen.setVisible(false);
            endScreen.setVisible(true);
            frame.add(endScreen);               //display endscreen
            frame.remove(screen);
            frame.pack();
            screen.destroy();
            endScreen.grabFocus();
            Thread.sleep( millis: 3000);
            break;                          //break while.loop
        }
```

Abbildung 3: while-Schleife der Methode run()

Das zurücksetzen des Objektes geschieht über die Methode resetGame(). Alle Gegner und Geschosse, die sich noch im Spiel befinden können, werden ebenfalls resettet. Danach wird ein einer StartScreen und Screen erzeugt, gameStatus auf NOT_STARTED gesetzt und der Ablauf mit run() von vorne gestartet.

StartScreen.java

Mit der Klasse StartScreen wird der Bildschirm repräsentiert, der zu sehen ist, wenn die Anwendung gestartet wird. StartScreen erbt von JPanel und besitzt als Attribute einen JButton startButton und einen boolean-Wert started. startButton ist der Button, mit dem das eigentliche SpaceInvaders-Spiel gestartet wird. Für den Button wird JButton benutzt, damit diesem ein Icon gesetzt werden kann.

Neben einer Methode getStarted, die den Wert von started zurückgibt, besitzt die Klasse sonst nur einen Konstruktor. Über diesen ein JPanel erzeugt und diesem ein Bild, das als Logo dient, und der startButton hinzugefügt.

EndScreen.java

Die Klasse EndScreen ähnelt in etwa StartScreen, auch sie erbt von JPanel. Das einzige Attribut hier ist jedoch endText, dass ein JLabel-Objekt ist. Dieses dient einen entsprechenden Text oder Bild anzuzeigen, das den Ausgang des Spiels anzeigt.

Wie auch bei StartScreen, werden im Konstruktor der Klasse die wichtigsten Parameter für EndScreen gesetzt und die anzuzeigenden Elemente hinzugefügt.

In der Methode setEndText entscheidet das EndScreen-Objekt, ob der Bildschirm gewonnen oder verloren anzeigt. Der Methode muss ein boolean-Wert übergeben werden. Wurde das Spiel gewonnen, sollte dieser true sein. In diesem Fall wird für endText versucht ein PNG als Icon zu laden. Den Pfad der Bilddatei bekommt das Objekt aus dem Config-Interface. Kann das Bild nicht geladen werden und es tritt eine Exception auf, bekommt endText anstatt einem Icon einfach einen Text. Für den Fall, dass der übergebene Parameter der Methode false ist, das Spiel also verloren wurde, läuft das Programm ähnlich dem ersten Fall ab. Allerdings wird nun versucht ein anderes PNG-Bild mit dem Schriftzug Game Over zu laden. Ist dies nicht möglich, wird auch hier stattdessen ein Text gesetzt.

Screen.java

Das eigentliche Spielgeschehen wird durch die Klasse Screen realisiert. Auch diese ist eine Kind-Klasse von JPanel, implementiert zusätzlich aber noch einen ActionListener.
In der Klasse versteckt sich die grundlegende Funktionalität des Space Invaders-Spiels und sie kann daher als Herzstück der Anwendung betrachtet werden.

In den Attributen werden hier unter anderem die Größe und Hintergrund des Spielfeldes angegeben, so wie ein Timer, das Spaceship und ein Delay gesetzt.

Im Konstruktor der Klasse wird die Methode initScreen() aufgerufen, die dann entsprechende Parameter für die Attribute setzt.

Mit der Methode actionPerformed, der ein ActionEvent übergeben wird, werden eine Reihe weiterer Methoden der Klasse oder anderer Objekte aufgerufen. Die meisten dieser Methoden werden genutzt um die unterschiedlichen Spielelemente über den Bildschirm zu bewegen.

Im Näheren soll in diesem Dokument noch auf zwei weitere Methoden der Klasse eingegangen werden.

In der Methode draw() werden die einzelnen Spielelemente auf den Bildschirm gezeichnet. Als Parameter wird der Methode ein Graphics-Objekt übergeben. Dieses wird zu Beginn der Methode in ein Graphics2D-Objekt umgewandelt.

Als nächstes wird mit dem Objekt das Hintergrundbild gezeichnet. Der Pfad des Bildes wird auch hier, wie bei anderen Klassen, über das Interface übergeben.

Wenn das Spaceship, das als Figur des Spielers dient, noch nicht getroffen wurde, wird dieses ebenfalls als Bild an der entsprechenden Position des Spaceships gezeichnet. Auf die gleiche Weise werden danach auch die Geschosse, repräsentiert durch die Klasse Bullet, und Gegner, repräsentiert durch Enemy, gezeichnet. Der große Unterschied bei diesen Spielelementen ist jedoch, dass gleichzeitig mehrere Objekte davon auf dem Spielfeld existieren können. Deshalb wird mithilfe von for-Schleifen über die Arraylisten iteriert, die die entsprechenden Objekte beinhalten. Für jeden Gegner oder jedes Geschoss wird dann ein Bild oder im letzten Fall ein Rechteck gezeichnet.

Damit die Gegner sich auf dem Spielfeld „selbstständig“ bewegen, existiert die Methode moveAliens(). Hier wird ebenfalls über alle Gegner iteriert. Wenn die Methode shouldMoveRight() von Enemy true zurück liefert, wird dem Objekt eine neue X-Koordinate gesetzt, indem die alte mit dem Parameter ENEMY_SPEED aus Config addiert wird. Liefert die Methode false, wird die X-Koordinate um die Bewegungsgeschwindigkeit subtrahiert. Dadurch bewegen sich die Gegner nach links oder rechts.

Sobald einer der Gegner an den Bildschirmrand stößt, tritt eine Exception auf. In der Behandlung dieser wird ein boolean Wert moveY auf true gesetzt und die for-Schleife manipuliert. Es wird zurück zum Anfang der for-Schleife gesprungen, aber wird nicht mehr über alle Gegner iteriert, sondern nur noch zu bis zu dem, der an den Rand des Spielfeldes gestoßen ist.

Nach der for-Schleife wird der Wert von moveY überprüft. Sollte dieser wahr sein, bekommt jeder Gegner eine neue Y-Koordinate, wodurch die Gegner sich auch nach unten bewegen. Auch hier kann eine Exception auftreten, wenn die Gegner den unteren Spielfeldrand erreichen. In diesem Fall ist das Spiel allerdings verloren und das Spaceship des Spielers wird zerstört.

DrawableObject.java

Bei DrawableObject handelt es sich um eine Eltern-Klasse von Spaceship, Enemy und Bullet. Sie fasst alle Methoden und Attribute zusammen, die jedes Spielelement benötigt, dass auf dem Screen gezeichnet werden muss.

Zu den gemeinsamen Attributen gehören dabei x & y für die Position, height und width für die Höhe und Breite und img das es erlaubt das Objekt durch ein Bild zu repräsentieren.

Objekte können entweder erstellt werden, in denen einem Konstruktor ein Bild-Pfad und eine Bildbeschreibung übergeben wird, es können aber auch „leere“ Objekte ohne Parameter erstellt werden. Der letzte Fall ist für die Bullets wichtig.

Als Methoden besitzt die Klasse Standard Getter- und Setter-Methoden für die einzelnen Attribute. Daneben existiert noch eine abstrakte Methode destroy(), die in jeder der Kind-Klassen anders implementiert werden muss und die Methode hit(). Hit() wird ein anderes DrawableObject übergeben um zu überprüfen ob sich die Objekte an einer Stelle Koordinaten auf dem Spielfeld teilen. Ist dies der Fall, bedeutet das, dass die Objekte mit einander kollidieren. Diese Überprüfung ist wichtig für die Methode hitDetection aus der Klasse Screen, mit der entschieden wird, wann ein Spielelement getroffen wurde und vom Spielfeld entfernt werden muss.

Spaceship.java

Wie bereits in g. beschrieben, handelt es sich bei Spaceship um eine Klasse, die von DrawableObject erbt. Neben den vererbten Attributen besitzt Spaceship zusätzlich noch zwei Attribute speed und moveSpeedX, die die horizontale Bewegungsmöglichkeit des Spaceships realisieren, isAlive, das anzeigt ob der Spieler getroffen wurde und ein Gun-Objekt.

Über die Methode move() wird dem Objekt eine neue X-Koordinate mithilfe des Attributs moveSpeedX gegeben. Die Methoden setMoveDirectionLeft() und setMoveDirectionRight() bestimmten dabei, ob moveSpeedX negativ (nach links) oder positiv (nach rechts).

KeyPressedListener.java

Mit KeyPressedListener werden die vom Nutzer gedrückten Tasten mit den Aktionen auf dem Spiel verknüpft. Hier ist in den Klassen-Attributen auch verankert, welche Taste welches Event auslösen soll.

Entspricht das KeyEvent (die vom Nutzer gedrückte Taste), das der Methode keyPressed() übergeben wird, einem der Attribute der Klasse, wird die entsprechende Action ausgeführt.

```
public void keyPressed(KeyEvent keyEvent) {
    if(keyEvent.getKeyCode() == SHOOT)
        pressedKeys.add(keyEvent.getKeyCode());
    if(keyEvent.getKeyCode() == MOVE_RIGHT   &&  !pressedKeys.contains(MOVE_LEFT))
        pressedKeys.add(keyEvent.getKeyCode());
    else if(keyEvent.getKeyCode() == MOVE_LEFT && !pressedKeys.contains(MOVE_RIGHT))
        pressedKeys.add(keyEvent.getKeyCode());
}
```

Abbildung 4: Methode keyPressed

Enemy.java

Die Klasse Enemy ist so gesehen der Gegenpart zu Spaceship. Mit ihr werden die Gegner des Spielers realisert. Auch sie erbt von DrawableObject.

Wie auch Spaceship besitzt die Klasse als Attribut ein Objekt vom Typ Gun. Darüber hinaus gibt es aber noch die Attribute canShoot, das besagt ob ein Gegner gerade in der Lage ist zu schießen, moveRight, das darüber entscheidet, ob sich die Gegner auf dem Screen nach links oder rechts bewegen und eine ArrayList, dieser werden die einzelnen Gegner hinzugefügt.

Wird ein Objekt der Klasse erstellt, wird dem geerbten Attribut img ein Bildpfad übergeben, ein neues Gun-Objekt erstellt und das erzeugte Enemy-Objekt der ArrayList hinzugefügt.

Die Methode getEnemies() liefert entweder die bereits existierende Liste von Enemies zurück oder aber erzeugt eine neue. Wird eine neue Erzeugt werden dieser mithilfe zweier for-Schleifen in der oberen Hälfte des Spielfeldes für jeden 9ten Abschnitt der Breite ein eines Enemy-Objekt hinzugefügt. Zur Veranschaulichung der Schleifen dient Abbildung 5.

```
for (int x = Config.BOARD_BORDER_LEFT; x < (Config.BOARD_WIDTH - Config.BOARD_BORDER_RIGHT); x += Config.BOARD_WIDTH / 9) {
    for (int y = Config.BOARD_BORDER_UP; y < Config.BOARD_HEIGHT / 2; y += Config.BOARD_WIDTH / 9) {
```

Abbildung 5: for-Schleifen für Enemy-Generierung

Da die Gegner ein Objekt der Klasse Gun besitzen und damit fähig sind Bullets abzuschießen, existiert noch die Methode randomShoot(). Diese sucht einen zufälligen Gegner aus der ArrayList aus und lässt ihn schießen. Anschließend wird das Attribut canShoot auf false gesetzt und erst nach 1 Sekunde wieder geändert. Dies geschieht, damit nicht alle Gegner auf einmal schießen können.

Gun.java

Besitzt ein Objekt einer Klasse ein Objekt der Klasse Gun, besitzt es damit die Fähigkeit Bullets abzufeuern.

Zur Unterscheidung, ob die Geschosse zum Spieler oder dessen Gegner gehören, besitzt die Klasse das Attribut type, zusätzlich gibt es noch timer, isReady und COOLDOWN. Mit den letzten dreien wird in der Methode shoot() bestimmt, ob und wann ein neues Geschoss abgeben werden kann. Dazu wird in der Methode ein neues Bullet-Objekt erzeugt und diesem x- und y-Koordinaten übergeben, die auch der Methode bereits als Parameter übergeben wurden.

Bullet.java

Bei Bullet handelt es sich um eine abstrakte Klasse, die von DrawableObject erbt. Ihre Attribute sind isFired, das besagt ob das Objekt abgefeuert wurde, type, das unterscheidet, ob das Objekt zum Spieler oder den Gegnern gehört und eine ArrayList, in der alle Bullets gespeichert werden, die sich auf dem Spielfeld befinden.

Tests

Bereits während der Entwicklung der Anwendung wurden einzelne Klassen und Methoden in ihrer Funktionalität getestet. Durch die grafische Umsetzung und Ausgaben über die Konsole konnte überprüft werden ob Abläufe wie geplant abliefen oder bestimmte Werte erreicht wurden.

Im fertigen Zustand des Hauptspieles wurde dieses mehrmals von den Mitgliedern des Projektteams getestet. Dabei wurde darauf geachtet, ob ein reguläres Space Invaders-Spiel gespielt werden konnte. Weiterhin wurden in der Test-Phase kleinere grafische Unstimmigkeiten behoben und der Spielablauf optimiert.

Reflektion Projektanforderungen

Zu Beginn des Projektes wurde vom Team ein Pflichtenheft erstellt, dass die Anforderungen und Erwartungen an die Anwendung festhalten sollte.

Als Hauptanforderungen wurden darin genannt die grundlegende Funktionalität des Computerspiels SpaceInvaders zu programmieren. Der Ablauf des Spiels sollte dabei grafisch über ein Fenster ausgegeben werden.

Neben der Hauptanforderung wurden noch erweiterte Anforderungen benannt, bei diesen handelt es sich um:

- Ein Spielmenü
- Die Möglichkeit verschiedene Spiel-Themen auszuwählen
- Spiel-Erweiterung durch Power-Ups

Die Hauptanforderung des Projektes wurde erfüllt. Die Anwendung ermöglicht es ein Space Invaders-Spiel zu starten und nach den bekannten Regeln zu spielen. Auch die grafische Realisierung erfolgt. Diese Aspekte wurden auch in Test bestätigt.

Allerdings ist anzumerken, dass für das Erreichen dieser Anforderung deutlich mehr Zeit benötigt wurde, als im Pflichtenheft eingeschätzt.

Diese Tatsache führte auch dazu, dass der Großteil der erweiterten Anforderungen nicht erfüllt werden konnte. Lediglich der Punkt Spielmenü konnte zum Teil durch den vorhandenen Start- und End-Bildschirm realisiert werden. Das im Pflichtenheft beschriebene zugehörige Optionsmenü wurde jedoch der fehlenden Zeit wegen nicht umgesetzt.

Eine der größten Hindernisse des Erreichens der kompletten Anforderungen war eine zeitliche Fehleinschätzung. Der Aufwand des eigentlichen Spieles wurde zu gering gewertet.

Problemanalyse

Während der Umsetzung des Projektes kam es zu mehreren Schwierigkeiten und Problemen. Einige waren so gravierend, dass sie die Erfüllung der im Pflichtenheft genannten Anforderungen verhinderten.

Die Zusammenarbeit des Projektteams lief auf menschlicher Ebene gut, allerdings gab es wie im Vorfeld erwartet einige Differenzen hinsichtlich der Erfahrung in der Programmierung mit Java. Auch der Umgang mit Git war nicht bei allen geübt und intuitiv. Die fehlenden Kenntnisse konnten zwar Absprache und Hilfestellung innerhalb des Teams weitgehend gelöst werden, kosteten jedoch einige Zeit. Besonders die Einrichtung von Github und die Nutzung über die IDE gestalteten sich anfangs sehr schwierig. Durch die aktuelle Situation bedingt durch COVID-19 war dieses Tool jedoch unumgänglich.

Auch die im Pflichtenheft erwähnten Schwierigkeiten hinsichtlich der grafischen Umsetzung traten im kleinen Ausmaß ein. Zum Teil war das Umdenken von der Aktion und der grafischen Darstellung schwierig. Dies konnte so umgangen werden, in dem das Teammitglied, das die meiste Erfahrung in der Programmierung mit JPanel besaß, zunächst die groben graphischen Strukturen programmierte. Diese Strukturen konnten anschließend vom restlichen Team als Beispiele und Blaupausen verwendet werden.

Eine große Schwierigkeit trat bei der Bewegung der Gegner auf. Für die Realisierung wurden verschiedene Ansätze mit while- und/oder for-Schleifen probiert. Probleme waren unter anderem, dass die Gegner sich durch einander hindurch bewegten, einige zum Teil am Rand „hängen blieben“ oder nicht alle gleichzeitig die Bewegungsrichtung änderten, sobald ein Gegner den Spielfeldrand erreicht hatte. Eine Lösung wurde letztendlich dadurch erreicht, in dem nicht jedes Mal mit einer for-Schleife über alle Gegner iteriert wurde.

Das größte Problem war jedoch die Fehleinschätzung des Aufwandes für die Grundfunktionalität. Wie bereits im Abschnitt „Reflektion Projektanforderungen“ erwähnt konnten fast alle erweiterten Anforderungen nicht erreicht werden. Hier wäre es sinnvoll gewesen deutlich mehr Zeitpuffer einzuplanen, sich lediglich auf die Grundstruktur zu beschränken und die erweiterten Anforderungen gegebenenfalls als optionale Features zu formulieren. An diesen kann dann bei genügend zeitlichen Freiraum gearbeitet werden.

Die Fehleinschätzung lag vermutlich vor allem daran, dass das Spielkonzept recht simpel erschien, die Umsetzung aber gekoppelt mit den teilweise fehlenden Erfahrungen aufwendiger war.

TO-DO/Verbesserungen

Für die Anwendung können noch die im Pflichtenheft erweiterten Anforderungen implementiert werden. Grundsteine dafür wurden bereits stellenweise gesetzt. So kann beispielsweise im Startmenü ein weiterer Button hinzugefügt werden, der auf ein Optionsmenü führt. Die Umsetzung dieses kann über die Änderung von Werten des Config-Interfaces realisiert werden. So können für verschiedene Schwierigkeitsgrade die Bewegung oder Schussgeschwindigkeit der Gegner erhöht oder verringert werden.

Auch andere Themen könnten über das Optionsmenü ausgewählt werden, in der Config müssen dafür dann andere Bildpfade für die entsprechenden Spielelemente gesetzt werden.

Stellenweise könnte auch die Spielmechanik noch verfeinert werden.

Hinsichtlich des Projektablaufes muss an der Aufwandseinschätzung gearbeitet werden. Für zukünftige Projekte sollte sich zunächst nur auf das wesentliche beschränkt werden und ausreichend Bearbeitungszeit eingeplant werden.

Fazit

Das Projekt ist in seiner Umsetzung der Hauptanforderungen gelungen. Durch die Anwendung konnte ein Einblick in die Planung und Realisierung von Software-Projekten gewonnen werden. Durch das Projekt konnten eigene Schwäche und Verbesserungsansätze erkannt werden, vor allem gutes Zeitmanagement sollte in zukünftigen Projekten geachtet werden.

Darüber hinaus ergaben sich durch die Themenwahl des Projektes Einblicke in die grafische Programmierung mit Swing und die Umsetzung von kleinen Computerspielen.