Invading Space

Modul:	Fortgeschritten Programmierung
Dozent:	Rüdiger Severin
Gruppe:	Patrick Bigge, Leon Henne, Maximilian Kinzler, Jan Moormann, Jasmin Noll
Kurs:	WWI2020F

Inhaltsverzeichnis

1 Einleitung

Dieses Programmierprojekt ist im Rahmen der Vorlesung „Fortgeschrittene Programmierung“, gehalten von Rüdiger Severin, als Abschlussaufgabe entstanden.

Die Umsetzung des Projektes erfolgte durch fünf StudentInnen, die ihre Kenntnis zusammengetragen, um ihre Zielsetzung in der vorgegebenen Zeit von drei Wochen, zu erreichen.

2 Zielsetzung

Das Ziel diese Programmierprojekts ist es ein kompilierbares und ausführbares Programm zu erstellen, dass die gestellten Anforderungen erfüllt.

Dabei soll es unter anderem die Konzepte und Elemente, die wir in der Vorlesung kennengelernt haben, sinnvoll und korrekt verwenden. Kernelemente waren die Objektorientierung, die Vererbung und die Datenkapselung. Das Endprodukt soll ein Programm sein, dass das bekannte Spiel “Space Invaders” nachbaut und zu unseren Wünschen anpasst.

2.1 Anforderungen

Die grafische Oberfläche des Programms soll mit Hilfe des Frameworks JavaFX erstellt werden.

Durch das Starten des Programms gelangt man auf einen Startbildschirm, auf dem man einen Spielernamen eingeben kann und eine Legende sieht, wie viele Punkte welcher Gegner hat. Sobald man einen Spielmodus ausgewählt hat, startet das Spiel.

Dafür werden gleich zu Beginn 60 Gegner und das Raumschiff erzeugt. Das Raumschiff wird über die Tastengruppen W-A-D oder den Pfeiltasten gesteuert. Es kann sich auf einer horizontalen Linie nach Links und Rechts bewegen und auf Tastendruck (W / Pfeiltaste hoch) feuert es einen Schuss ab.

Die Gegnerobjekte bewegen sich nach einem vorprogrammierten Zyklus. Der Block an Gegner bewegt sie sich so lange nach rechts oder links, bis der äußerste Gegner an den Rand stößt. Wenn das passiert, rücken alle Gegner eine Reihe nach unten und wechseln die Richtung. Genauso wie das Raumschiff können die Monster ebenfalls einen Schuss abgeben.

Für das Abschießen der Gegner erhält der Spieler Punkte, die im Laufe des Spiels zu einer Gesamtpunktzahl aufaddiert werden. Hat der Spieler alle Gegner abgeschossen, so steigt er ein Level auf und eine neue Gegnerwelle erscheint. Diese bewegen sich mit jeder Welle schneller, so dass die Schwierigkeit sich ebenfalls erhöht.

Das Spiel endet, wenn das Raumschiff von einem Schuss der Gegner getroffen worden ist oder die Gegner so weit nach unten gerückt sind, dass sie mit dem Raumschiff kollidieren.

Wenn das Spiel verloren ist, gelangt man zum Endbildschirm, auf dem die erreichten Punktzahl zu sehen ist und die Top drei Spieler des Modus. Die Daten jeder Runde werden mit dem Spielernamen in einer Datei gespeichert.

Der Unterschied zwischen dem Normal-Modus und dem Höllen-Modus ist, dass gleich zu Beginn die Gegner sich im Höllenmodus viel schneller bewegen und es somit schwieriger ist die Runden zu überleben.

3 Organisation

Gleich zu Beginn haben wir uns Gedanken gemacht, wie wir am besten an dieses Projekt herangehen, um das für uns bestmögliche Ergebnis zu erzielen. Dazu haben wir eine Projektleitung bestimmt, die von Jasmin Noll übernommen worden ist.

Über ein Git-Repository konnten wir unsere Programmfortschritte zwischenspeichern, mit unseren Teammitgliedern teilen und zusammenarbeiten.

Durch die Git-Repository-Funktion “Projects” konnten wir einen Überblick beibehalten, mit welchen Aufgaben wir uns noch beschäftigen müssen, wo es zwischendurch Probleme gibt und was bereits erledigt ist. Dafür haben wir ein “Projekt” für jeden Teilbereich unseres Programmierprojekts erstellt und bestmöglich gepflegt.

Zusätzlich haben wir in unserem Programmcode in IntelliJ todo-Kommentare hinterlassen, um genauere Angaben zu machen oder auch damit Teammitglieder sehen, dass an gewissen Stellen noch etwas fehlt und demnächst implementiert wird.

Da wir uns entschieden haben, dass in erster Linie jeder für sich programmiert, haben wir zwei “Pflicht”-termine in der Woche vereinbart in denen jeder Studierende seinen aktuellen Stand, was seit dem letzten Mal präsentiert und was jener bis zum nächsten Treffen erreichen möchte. Außerdem haben wir hier Zeit geschaffen Probleme zu besprechen oder weitere organisatorische Dinge zu klären.

Jeden Sonntag haben wir einen optionalen Termin festgelegt, an dem sich die Mitglieder treffen können, um gemeinsam zu Programmieren und Probleme anzugehen.

Am letzten Wochenende vor der Abgabe hat sich das ganze Team zusammengesetzt und die finale Version des Programmierprojekts wurde erstellt.

3.1 Aufteilung der Aufgaben

Die einzelnen Teile des Projekts haben sich wie folgt auf die Gruppenmitglieder verteilt:

Gruppenmitglied	Übernommene Aufgabe
Jan Moormann	Klasse Spielablauf (Threading & Gameticks) Zusammenbringen der verschiedenen Klassen
Jasmin Noll	Teamleitung Klassen BeweglicheObjekte, Raumschiff, Gegner und ObjektSteuerung Erstellung visueller Ressourcen
Leon Henne	Framework Anbindung der Klassen an die Oberflächen GUI Styling
Maximilian Kinzler	Klasse Schuss Erstellung und Korrektur der Dokumentation und PowerPoint
Patrick Bigge	Klassen Spieler und Punkteliste Mockups Code Refactoring / Dokumentation / Cleaning Korrektur der Dokumentation

Tab. 1: Gruppenaufteilung

In der Dokumentation wurden die entsprechenden Teile auch von den Personen, die die Aufgaben übernommen haben, verfasst.

4 Umsetzung

Die Basis der Umsetzung sollte auf der externen Bibliothek JavaFX aufbauen. Diese Entscheidung erforderte dabei anfänglich viel Zeit zum Einlesen in den Aufbau und die Funktionsweisen der Bibliothek. Nach der Installation der JavaFX SDK war die nächste Aufgabe den Aufbau der Oberflächen anhand der ersten Version des Mockups zu modellieren. Um den Spielfeldaufbau zu vereinfachen, wurde sich auf eine feste Fenstergröße von 600 x 768 Pixeln geeinigt, damit sich Elemente anhand ihrer festen Koordinaten platzieren und bewegen lassen, und nicht mit einem Padding zu jedem Rand oder relativen Einheiten definiert sein müssen. Allgemein ist für die Darstellung der Schriften eine Schriftart ähnlich der Spielfilmreihe “Star Wars” gewählt worden, aufgrund des gleichen Themenschauplatzes. Diese Schriftart ist dabei nicht selbst erstellt, sondern aus einer externen Quelle[1] bezogen worden. Bilder und Symbole hingegen sind in Anlehnung an die Spielfilmreihe durch Jasmin Noll erstellt worden.

[1] https://www.dafont.com/de/star-jedi.font?psize=s

5 GUI

Die Grafische Benutzeroberfläche ist anhand der Abbildung 7 strukturiert. Das Spiel besitzt insgesamt drei verschiedene Oberflächen. Die Eigenschaften der statischen Elemente sind über das Programm “SceneBuilder” in hierarchischer Form in verschiedenen FXML Dateien festgelegt, und können so ohne aufwendiges Erstellen in dem Programm hinzugefügt werden.

An der oberen Seite des Startfensters werden die besten Spieler der verschiedenen Spielmodi ausgegeben, dessen Punktzahl es zu übertreffen gilt. In der Mitte des Startbildschirmes muss der Spieler einen Namen in ein Textfeld eintragen, mit dem die Punktzahl später gespeichert wird. Außerdem existieren zwei Knöpfe, über die sich die verschiedenen Spielmodi starten lassen. Neben den funktionalen Elementen sind noch Labels und Bilder zur Benennung des Spieltitels und zur grafischen Aufbereitung der Startseite platziert.

Die Spieloberfläche zeigt ähnlich, wie das Startfenster den Spieler mit der höchsten Punktzahl im ausgewählten Spielmodus, am oberen Fensterrand an. In der Mitte des Fensters befindet sich das Spielfeld, in welchem über Tasteneingaben das Spielgeschehen vom Spieler gesteuert wird. Die Darstellung des Raumschiffs und der Monster wird anders als bei den restlichen Elementen nicht über SceneBuilder in der FXML Datei gespeichert, sondern entsteht aus dem Programmablauf.

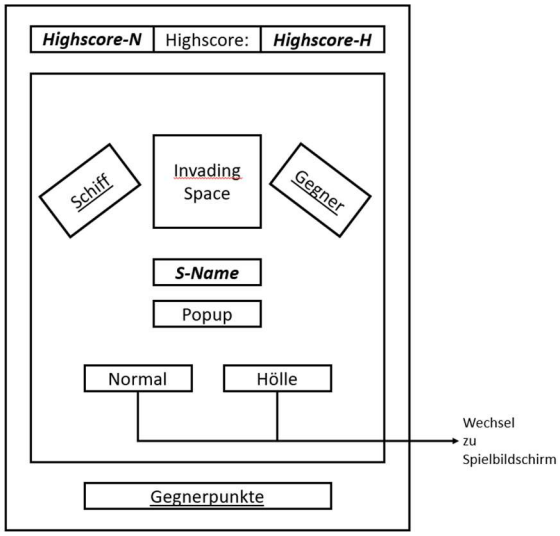

Abb. 1: Mockup Start

Der Popup Bereich wird zur Ausgabe besonderer Ereignisse, wie dem Starten einer neuen Gegnerwelle benutzt. An der unteren Fensterseite ist der aktuelle Spielername, sowie die bisherige Punktzahl aus den Labeln abzulesen. Ebenso befindet sich dort ein Knopf, der das Spielgeschehen vorzeitig beendet und den Spieler erneut auf die Endseite leitet.

Auf der Endoberfläche wird die erreichte Punktzahl des Spielers ausgegeben, sowie die drei Spieler mit den höchsten Punktzahlen in dem zuvor gespielten Spielmodus. Des Weiteren befinden sich zwei Knöpfe auf der Oberfläche, mit welchen direkt eine neue Runde gestartet wird, oder der Spieler zur Startseite weitergeleitet wird.

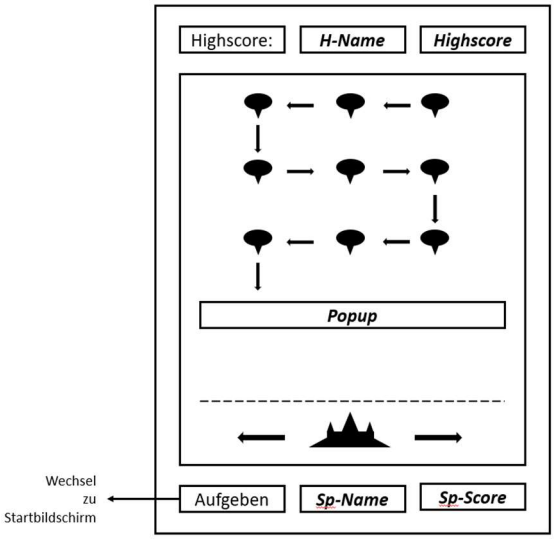

Abb. 2: Mockup Spiel

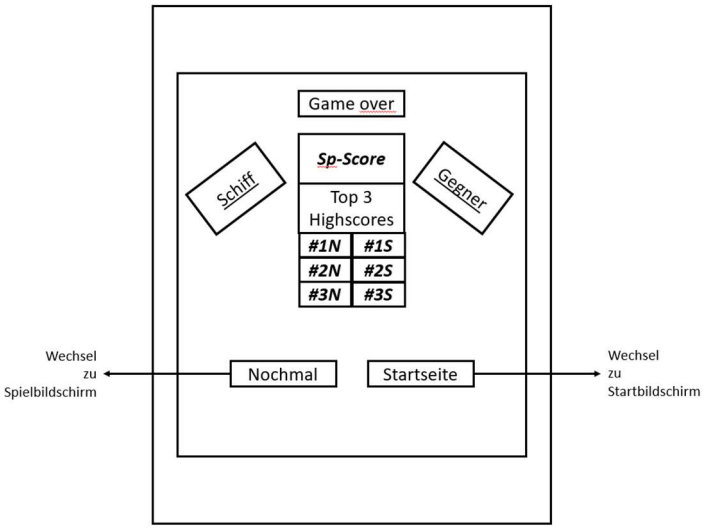

Abb.3: Mockup Ende

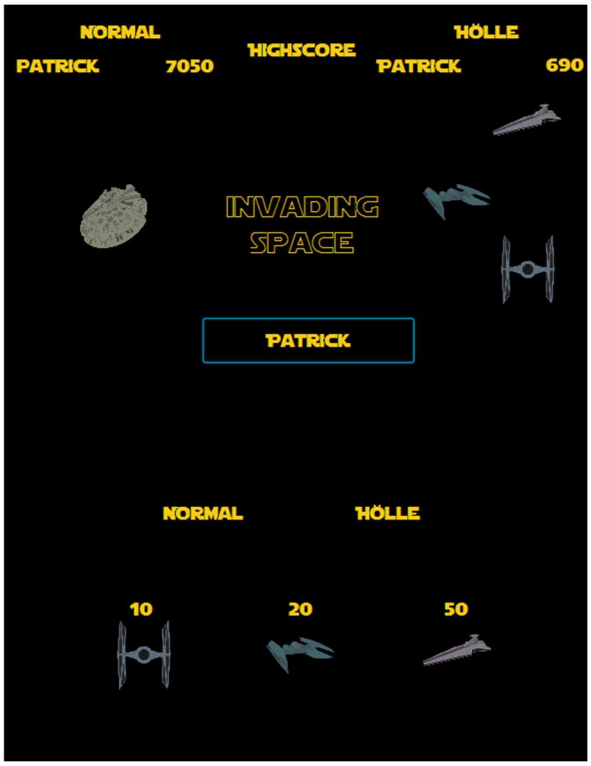

Abb. 4: Oberfläche

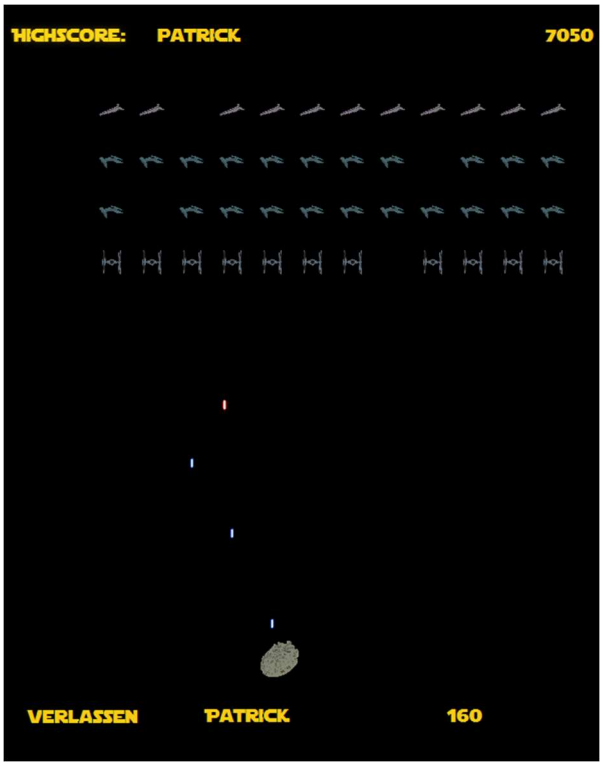

Abb. 5: Oberfläche

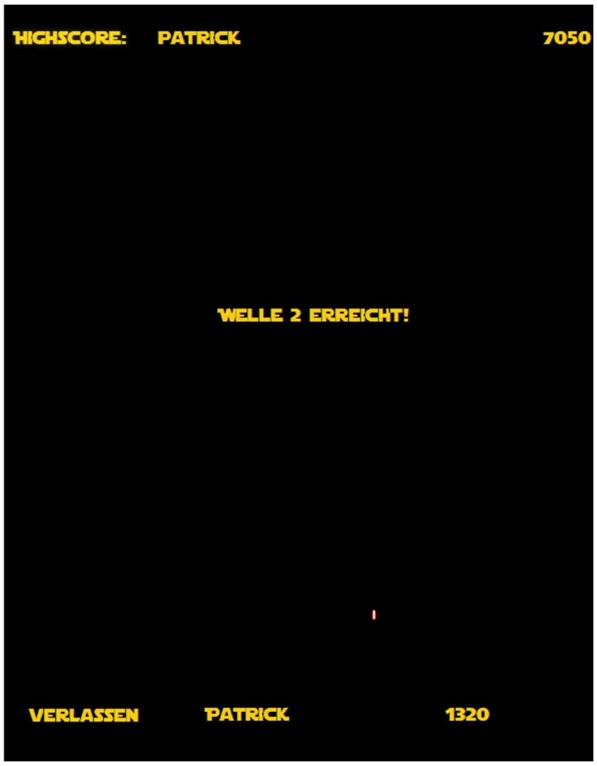

Abb. 6: Oberfläche

Abb. 7: Oberfläche

6 Framework

Zur Anbindung der Oberflächenelemente wie Labels, Knöpfe oder Textfelder an die restliche Logik sind drei “Controllerklassen” erstellt worden. Diese Klassen bilden über die direkte Verknüpfung an die FXML Dateien die Schnittstelle zwischen der integrierten, oder in anderen Klassen implementierten Logik und den GUI Elementen. Dazu enthalten sie alle Methoden, die entweder direkt oder aus anderen Klassen importiert, Logik umsetzen, die Auswirkungen in der Oberfläche haben.

Zum Starten des Spiels fungiert eine Hauptklasse “MainGui”, in der die Startoberfläche aufgerufen und die dazugehörige Logik aus dem “Startbildschirmcontroller” verwendet wird. Zudem werden grundsätzliche Eigenschaften, wie Fenstergröße und Fenstertitel festgelegt. Abschließend wird die Beendigung des Programms und der Threads über den Fensterknopf (“X”) abgehandelt.

Die Klasse des Startbildschirmcontrollers besitzt eine Methode “aktiviereStartbildschirm”. Diese kümmert sich um die Ausgabe des besten Spielers. Im weiteren Programmverlauf ist die Eingabe eines Spielernamens in das Textfeld vorgesehen. Ist kein Name, oder ein Name mit einem Komma eingegeben, so wird beim Betätigen des Knopfes ein Fehler erzeugt, welcher aufgefangen und über einen Popup dem Spieler angezeigt wird. Ist der eingegebene Name valide, kann das Programm zum Aufruf des Spielbildschirms (“wechselZuSpielbildschirm”), durch das Drücken einer der beiden Knöpfe ohne Fehler ausgeführt werden. Das Programm ruft die Spieloberfläche auf und erstellt ein Objekt von der Klasse “Spieler” mit dem eingegebenen Namen. Zusammen mit einem Wert für den ausgewählten Spielmodus wird die weitere Steuerung des Spielgeschehens an die Klasse “Spielbildschirmcontroller” übergeben.

Das Objekt des “Spielbildschirmcontrollers” aktiviert als erstes ihren Prozess zum funktionalen Aufbau der geladenen Oberfläche. Dieser Prozess beginnt mit der Ausgabe des besten Spielers im ausgewählten Modus und dem aktuellen Spielernamen. Der weitere Schritt ist das Starten des Threads über die Programmteile des Objektes der Klasse “Spielablauf”, welche auch die Verarbeitung aufkommender Tastatureingaben und Popupnachrichten übernehmen und so vollständig das Spielgeschehen steuern. Zur Beendigung des Spiels, durch Knopfdruck oder Verlieren des Spielers,

werden weitere Methoden des “Spielbildschirmcontroller” Objektes aufgerufen. In der Methode zur Beendigung des Spiels wird zunächst der laufende Spielthread gestoppt. Daraufhin wird die Oberfläche des Endbildschirms geladen, dem Spielerobjekt die Punktzahl zugewiesen und notwendige Parameter werden an das erzeugte “Endbildschirmcontroller” Objekt übergeben. So kann der “Endbildschirmcontroller” die weitere Verarbeitung auf der neuen Oberfläche übernehmen.

Ähnlich wie die anderen Controllerobjekte beginnt das “Endbildschirmcontroller” Objekt in der Methode “aktiviereEndbildschirm” mit dem funktionalen Aufbau der Oberfläche, wie bspw. der Zuweisung von Werten zu den vorgesehenen Labels. Auf dem Endbildschirm werden daraus folglich die erreichte Punktzahl des Spielers, die drei besten Spieler im gewählten Spielmodus, sowie Bilder zur visuellen Aufbereitung der Oberfläche ausgegeben. Intern wird das Ergebnis des Spiels nach Hinzufügung des Spielers in die “PunkteListe” in einer Textdatei festgehalten. Den Schluss des Spiel- und Programmverlaufs bilden die beiden Knöpfe auf dem Endbildschirm. Diese springen zu verschiedenen Punkten im Programmverlauf, sodass eine neue Runde gestartet, oder zum Startbildschirm zurückgekehrt werden kann.

7 Klassen der Spielobjekte

Für die Objekte des Spielers und der Gegner wurde sich eine Klassenhierarchie überlegt, die die Konzepte der Vererbung und der Kapselung beinhalten und beachten (siehe Abb. 12).

7.1 Klasse Bewegliches Objekte

Die Oberklasse “BeweglicheObjekte” beinhaltet alle Methoden, die das Verhalten beschreiben, die sich alle beweglichen Objekte teilen.

Die Oberklasse selbst wurde als “abstract” deklariert, da sie selbst nie instanziiert werden soll. Sie repräsentiert nur die Gemeinsamkeiten ihrer Unterklassen.

Dazu zählen das Erstellen der Position und des Aussehens, genauso wie die Standard Getter- und Setter-Methoden (bei uns Erhalte- und Setzte-Methoden), sofern diese benötigt sind. Zusätzlich gehört zu dem gemeinsamen Verhalten das Erstellen und Entfernen des Objekts und das Überprüfen, ob die Projekte mit den Spielrändern rechts und links kollidieren, bzw. von einem Schussobjekt getroffen werden.

Dabei wurden die einzelnen Methoden entsprechend als “private”, “protected” oder “public” eingestuft, um das Konzept der Kapselung zu wahren. Vor allem die Setter-Methoden werden nicht an fremde Klassen weitergegeben, da diese nichts an den betroffenen Werten ändern sollen, sondern nur die Unterklassen selbst.

Die Attribute der Oberklasse sind zum Großteil auf “private” gesetzt, da auf diese nicht direkt zugegriffen werden muss und von den Unterklassen nur durch die Setter-Methoden geändert werden können. “Protected” sind nur die Methoden, die auch für die Unterklassen relevant sind und die darauf zugreifen und diese nutzen können sollen. “Public” sind die Methoden, die auch von andren Klassen aufgerufen werden können, die nicht im direkten Zusammenhang zur “BeweglichenObjekte”-Klasse stehen.

7.1.1 Klasse Raumschiff

Die Klasse “Raumschiff” erbt von der Oberklasse “BeweglicheObjekte”. In dieser Klasse wird das Verhalten des Spielerraumschiffes festgelegt.

Die Klasse setzt für das Objekt ein neues Aussehen. Dazu gehören die Höhe und Breite des Objekts, genauso wie das Bild, dass für die Darstellung genutzt wird. Diese Anpassungen werden direkt beim Erstellen des Objekts, durch das Aufrufen des Konstruktors vorgenommen.

Es verwendet die entsprechenden Methoden seiner Oberklasse fügt allerdings noch die Methoden, die für die Links- und Rechtsbewegung des Objekts zuständig sind, genauso wie die Funktion zu schießen, hinzu.

Diese Methoden werden als “public” eingestuft, da sie von außenstehenden Klassen aufgerufen werden müssen, um das Raumschiff nutzen zu können, wie es angedacht ist.

7.1.2 Klasse Gegner

Die Klasse “Gegner” erbt von der Oberklasse “BeweglicheObjekte” fungiert aber ebenfalls als Oberklasse für die einzelnen Gegnerklassen. Sie ist ebenfalls als “abstract” deklariert, da diese Klasse selbst nicht aufgerufen werden soll, sondern nur das Verhalten der Gegner generell implementiert.

Anders als ihre Oberklasse oder auch die Raumschiffklasse, haben die Gegnerklassen das zusätzliche Attribut der Punkte, indem vermerkt wird welche Punkte der Spieler bekommt, wenn er den Gegner abschießt. Entsprechend dafür gibt es Getter- und Setter-Methoden, wobei die Setter-Methode nur von den Unterklassen verwendet werden kann, um ihre eigene Punktzahl anzupassen.

Neben den Methoden aus ihrer Oberklasse werden in dieser Klasse, ähnlich zur Raumschiffklasse, die Methoden zum Bewegen der Objekte und ihre Fähigkeiten zu schießen hinzugefügt. Der Unterschied zur Raumschiffklasse zum Thema Bewegung ist der, dass die Gegner sich neben rechts und links auch nach unten bewegen können. Dafür wurde eine passende Methode ebenfalls implementiert.

Dadurch, dass die Gegner die zusätzliche Fähigkeit haben sich nach unten zu bewegen wird hier auch eine weiter Methode erstellt, die überprüft, dass die Gegner sich nicht zu weit nach unten bewegen.

Die Unterklassen “GegnerZehn”, “GegnerZwanzig” und “GegnerFuenfzig” der Gegner-Klasse implementieren keine neuen Methoden. In diesen Klassen wird das Verhalten der aufgerufenen Gegnerklassen in Hinsicht ihrer Punktzahl und ihres Aussehens angepasst. Im Konstruktor werden diese Anpassungen vorgenommen und angewendet.

7.1.3 Klasse Schuss

Die Klasse „Schuss“ ist eine Unterklasse der Klasse „BeweglicheObjekte“. Die Klasse erzeugt Rechtecke, die sich nach oben und unten bewegen. Diese ist unterteilt in zwei Programmabschnitte „schiessenRaumschiff()“ und „schiessenGegner()“.

Hierbei ist zu beachten, dass der Nullpunkt des Spielfeldes an der oberen linken Ecke ist. Daraus resultierend müssen die Schüsse nach unten „fliegen“ mit der Y-Koordinate addiert werden und umgekehrt die Schüsse, die nach oben „fliegen“ müssen, subtrahiert werden. Des Weiteren wird geprüft, ob die Schüsse mit dem Spielfeldrand kollidieren.

Die Klasse „Schuss“ wird in der Klasse „Gegner“ und „Raumschiff“ verwendet.

7.2 Objektsteuerung

Die Klasse “ObjektSteuerung” ist eine eigenstehende Klasse deren Aufgabe es ist die beweglichen Objekte zu steuern und damit auch zu überprüfen, ob diese mit Rändern oder anderen Objekten kollidieren. Sie handhabt auch was mit den entsprechenden Objekten passiert, wenn eine Kollision eintrifft.

Sie ist dafür zuständig eine Liste an Schüssen sowohl für das Raumschiff als auch für die Gegner zu erzeugen und die Schusslisten und die ihr übergebene Liste an Gegnerobjekten zu verwalten. Während dem Spiel ist diese Klasse die einzige, die darüber in Kenntnis ist, welche Objekte im Moment auf dem Spielfeld existieren und was diese machen, bzw. machen sollen.

Für den Vermerk in welche Richtung sich die Gegner bewegen, wird ein “Enum” verwendet. Dies bietet sich an, da es nur zwei relevante Richtungen (links und rechts) gibt und durch “Enum” eine typsichere Definition gewährleistet wird.

Den Bewegungsvorgang der Gegner bestimmt ebenfalls die Klasse “ObjekteSteuerung”. Sie überprüft vor jeder Bewegung, ob die Objekte an einem der Spielfeldränder ankommen. Wenn das passiert, können sich die Objekte nicht weiter in diese Richtung bewegen. Im Falle der Gegner werden diese beim Erreichen des Randes eine Reihe nach untern gerückt und ein Richtungswechsel wird vorgenommen.

Um die Kollision mit den Rändern überprüfen zu können werden in dieser Klasse die X- bzw. Y-Koordinaten der Ränder als Variablen festgehalten. Diese Variablen sind als “private” eingestuft, da auf diese Werte von keiner anderen Klasse zugegriffen werden muss.

Durch die Klasse “ObjekteSteuerung” wird berechnet, wann welcher Gegner schießen soll. In der Methode “schiessenGegner()” wird überprüft welches der Gegnerobjekte sich in einem bestimmten Abstand der X-Achse entlang um das Raumschiff befindet. Von diesen Gegnern schießt dann das Objekt, welches sich in der untersten bestehenden Reihe befindet. Der Schuss wird über diese Methode gelöst und der Schussliste hinzugefügt.

Wenn Gegnerobjekte von einem Schuss getroffen werden, sorgt die Klasse “ObjektSteuerung” dafür, dass der Gegner sowohl von der grafischen Oberfläche als auch aus der Liste der Gegner verschwindet, genauso wie der treffende Schuss. Wird das Raumschiffobjekt von einem Schuss getroffen sorgt die Klasse dafür, dass dem Programm mitgeteilt wird, dass damit die Spielrunde vorbei ist.

Die Klasse “ObjektSteuerung” sorgt auch dafür, dass falls ein Schussobjekt kein anderes Objekt trifft, dieses rechtzeitig von der grafischen Oberfläche verschwindet und aus der Liste der Schussobjekte entfernt wird.

Sobald der Spieler alle Gegnerobjekte zerstört hat, benötigt die Klasse eine neue Liste von Gegnerobjekten. Dafür gibt es eine Methode, die einem anderen Teil des Programms mitteilt, wann eine neue Liste notwendig ist und eine Methode, die diese neue Liste entgegennimmt.

Die Klasse “ObjekteSteuerung” überprüft, ob die Kriterien für den Abbruch der Spielrunde, erfüllt sind und gibt einen booleschen Wert an einen anderen Teil des Programms weiter, damit dieser die Spielrunde beenden kann.

8 Spielablauf

Der Spielablauf unseres Invading Space Spiels muss zentral gesteuert werden, um den eigentlichen Ablauf zu koordinieren. So müssen zu Beginn das Raumschiff und die Gegner erzeugt werden. Außerdem müssen die Gegner, Schüsse und das Raumschiff während des Spiels immer wieder bewegt werden. Zudem muss regelmäßig überprüft werden, ob die Gegner getroffen wurden, diese müssen entfernt und die Punkte erhöht werden. Auch ein mögliches Ende des Spiels muss abgefragt werden. Diese Aufgaben übernimmt die eigenstehende Klasse “Spielablauf” (Abb. 8).

Die Klasse “Spielablauf” erbt von der Klasse “Thread”, damit sie als eigener Thread parallel zu der eigentlichen GUI laufen kann. Dieser Thread ermöglicht es, dass das Spiel unabhängig von den Aufgaben der Benutzeroberfläche laufen kann. Allerdings bringt das die Schwierigkeit mit sich, Änderungen auf der JavaFX Oberfläche zu tätigen, da die Oberfläche in einem eigenen JavaFX Application Thread läuft. Um Änderungen vorzunehmen, müssen diese daher in eine Art Warteschleife gebracht werden, die abgearbeitet wird, sobald der Thread der Oberfläche diese bearbeiten kann. Dafür musste in jede Klasse, die Änderungen an der Oberfläche vornimmt und von der Klasse “Spielablauf” aufgerufen wird, diese Änderungen in einem “Runnable” mit der Methode "runLater” an die

Oberfläche übergeben werden. So kann zum Beispiel der Popup Text, der bei einer neuen Welle angezeigt wird, an die Oberfläche gesendet werden und wird angezeigt, sobald der Oberflächen Thread alle vorherigen Methoden ausgeführt hat.

Um den eigentlichen Spielablauf zu berechnen wird, sobald der Thread der Klasse “Spielablauf” gestartet wurde, alle 40 Millisekunden ein Tick ausgeführt, welcher die notwendigen Methoden aufruft. In einem Tick werden zunächst immer alle Schüsse bewegt, indem die entsprechende Methode in einem Objekt der Klasse “ObjektSteuerung” aufgerufen wird, da diese die Schüsse und Gegner verwaltet. Die Schüsse bewegen sich daher 25-mal pro Sekunde. Mit dem Bewegen der Schüsse wird auch überprüft, ob diese einen Gegner oder das Raumschiff getroffen haben und die Punkte erhöht bzw. das Spiel über einen Aufruf in dem Objekt der Klasse “Spielbildschirmcontroller” beendet. Da die Gegner sich aber nur alle paar Sekunden bewegen sollen, bewegen diese sich abhängig von der Schwierigkeit, z.B. nur alle 20 Ticks. Um ein Vorhersagen eines Schusses der Gegner zu vermeiden, wird nach jedem Schuss der Gegner, eine zufällige Zeit, in einer von der Schwierigkeit abhängigen Bereich, erzeugt. Die Gegner können daher erst nach Ablauf dieser Zeit erneut schießen.

Es wird außerdem bei jedem Tick die Überprüfung, ob der Spieler alle Gegner einer Welle eliminiert hat, in dem Objekt der “ObjektSpieler” Klasse aufgerufen. Wenn dies zutrifft, wird ein Popup Text auf der Oberfläche erzeugt und anschließend neue Gegner erstellt. Zudem werden die Attribute “gegnerGeschwindigkeit” und “naechsterSchussZeitspanne” angepasst, um die Schwierigkeit zu erhöhen.

Um die Benutzereingaben verarbeiten zu können gibt es in der Klasse “Spielablauf” Methoden, die von der Oberfläche aufgerufen werden können. Zwei Methoden verschieben dabei das Raumschiff nach links bzw. rechts. Die dritte Methode überprüft, wie viel Zeit seit dem letzten Schuss vergangen ist und setzt eine boolesche Variable auf “true”. Dieses Attribut sorgt dafür, dass ein neuer Schuss bei dem nächsten Tick ausgelöst wird, sodass dieser Schuss synchron mit den anderen Schüssen bewegt wird.

9 Spieler

Die Spielerdaten sind in einer Textdatei gespeichert. Pro Eintrag gibt es jeweils den Namen des Spielers und die erreichte Punktzahl. Diese Struktur findet sich in der Klasse “Spieler” (Abb. 10) wieder.

Anfangs hatten wir überlegt, ob wir pro Spielernamen eine Liste von erreichten Punktzahlen haben wollen. Letztendlich haben wir uns dagegen entschieden, da mit der momentanen Implementierung das Gefühl der eigentlichen Highscoreliste erhalten wird. Dies weist somit eine ähnliche Struktur auf wie bei den ersten Automatenspielen, wozu Space Invaders auch zählt. So kann man gut auf einen Blick insgesamt absteigend die Highscores der Spieler betrachten.

Um die Spieler richtig in der Textdatei zu erfassen, darf kein Komma im Spielernamen enthalten sein und muss ausgefüllt sein, da die Textdatei durch Kommas separiert wird. Somit würde das ganze Konstrukt nach einer Überschreibung der Datei und erneutem auslesen scheitern, da dadurch die vorgegebene Struktur zur iterativen Erfassung der Spielerdaten zerstört werden würde. Um das zu verhindern haben wir noch zwei neue Exceptions (Abb. 11) definiert, welche in einem Fehlerfall geworfen werden und zu einem Popup auf der GUI führen.

10 Punkteliste

Die Klasse “PunkteListe” (Abb. 10) ist der Hauptakteur im Kontext der Klasse “Spieler”. Der Konstruktor dieser Klasse liest zunächst die Spielerdaten aus und sortiert diese absteigend. Für die GUI gibt es Methoden, wodurch man die höchsten Punktzahlen erhält, einen neuen Eintrag hinzufügt und die Textdatei mit dem neuen Stand überschreibt.

Die ausgelesene Datei wird in eine LinkedList transformiert, wodurch der indexierte Zugriff auf die höchsten drei Punktzahlen ermöglicht wird und man der Liste dynamisch neue Einträge hinzufügen kann. Anfangs hatten wir ein Treeset genutzt in diesem Kontext, da hier das absteigende Sortieren noch direkter durchgeführt werden kann. Allerdings hätte man hierfür dann ein Problem gehabt hinsichtlich der Highscores, denn das Set weist nicht die Möglichkeit des indexierten Zugriffes auf Elemente auf. Dieser wird allerdings für die drei höchsten Punktzahlen benötigt. Eine mögliche Lösung hierfür wäre eine Sublist gewesen, allerdings wäre dies umfangreicher gewesen. Deshalb haben wir uns letztendlich für die LinkedList entschieden, obwohl die Sortierung bei Treesets direkter wäre.

Nach jedem neuen Eintrag wird die Struktur absteigend geordnet, um die aktuellen höchsten Punkte immer bereitzustellen. Dies wird durch den Algorithmus InsertionSort erreicht, da dieser in dem geringen Umfang keinen großen Mehraufwand generiert. Außerdem ist dieser Algorithmus in unserem Anwendungsfall sinnvoll, da wir eine fast komplett sortierte Liste haben, wo nur ein Element an der falschen Position ist. In solchen Fällen ist der Algorithmus schneller als andere Algorithmen.

11 Herausforderungen

Eine Herausforderung war das Entfernen der beweglichen Objekte auf der grafischen Oberfläche. Bevor wir unsere jetzige Lösung gefunden haben, hatten wir die Objekte nur schwarz gezeichnet, weil sie somit auf dem schwarzen Hintergrund nicht mehr gesehen wurden. Allerdings waren die Objekte immer noch vorhanden, was dazu führte, dass je länger das Spiel lief die Perfomance immer stärker abnahm.

Die Lösung hierfür war die Objekte schwarz zu zeichnen, damit sie auf der grafischen Oberfläche nicht mehr zu sehen sind und zusätzlich das Objekt von der Oberfläche komplett zu entfernen.

Weiterhin hat es sich zeitweise als schwierig erwiesen, dass manche Teammitglieder an veralteten Versionen weitergearbeitet haben, wodurch eine parallele Weiterentwicklung auf veralteten Versionen entstanden ist. Dies hat einiges an Zeit in Anspruch genommen, weil man sich oft gegenseitig auf den neuesten Stand bringen musste, obwohl jeder den neuesten Stand durch einen einfachen Pull des Repositories gehabt hätte. Für zukünftige Projekte sollte man vorher den Stand der Teammitglieder abklären und ggf. Git Arbeitsmethoden besprechen / verständlich machen.

Gegen Ende des Projekts stellte sich heraus, dass keine genaue Absprache bei der Benennungskonvention von Variablen und Methoden vorlag. Es gab auch keine genaue Absprache zum generellem Codingstil, was im Endeffekt zu erheblichem Mehraufwand durch späteres Refactoring und umstrukturieren gelöst werden musste.

Bei unseren geplanten Treffen haben wir am Anfang immer länger gebraucht, als wir uns eigentlich vorgenommen hatten. Das lag daran, dass wir auch auf unsere aktuellen Probleme eingegangen sind und mögliche Lösungsansätze dafür besprochen haben. Gerade weil alle beim Meeting anwesend waren, wollten wir Probleme oder Dinge, die alle oder die Meisten im Team betreffen ungern noch auf einen anderen Termin verschieben. Dadurch wurden geplante 30 Minuten gerne mal zu einer oder eineinhalb Stunden.

Im Verlaufe des Projekts haben wir unsere zeitlich gesetzte Grenze aufgehoben, weil es uns wichtiger war die Dinge komplett zu besprechen. Dafür haben wir dann darauf geachtet die Themen, die

nur zwei Mitglieder betreffen später im Einzelnen zu besprechen und die Zeit des Meetings auf die Dinge zu beschränken, welche das generelle Vorhaben beeinflussen und für alle wichtig sind.

Für das Projekt hatten wir uns entschieden auf Deutsch zu programmieren. Das hat für uns oftmals eine Herausforderung dargestellt, da wir es gewohnt sind unsere Variablen, Klassen und Methoden auch teilweise Englisch zu benennen und uns manchmal auch eine englische Benennung schneller eingefallen ist, besser klang oder treffender war. Durch die deutschen Bezeichnungen kam es gerne auch mal zu längeren Namen, die nicht optimal waren.

12 Empfehlung/Lessons learned

Empfehlenswert wäre es eine generelle Benennungskonvention am Anfang des Projekts festzulegen. Dadurch spart man sich viel Zeit gegen Ende des Projekts und verhindert mögliche Verwirrung, da die eigentlich etablierten Namen nicht mehr geändert werden müssen.

Trotz einigen Schwierigkeiten bei der Weiterentwicklung an veralteten Versionen ist GitHub sehr empfehlenswert. Es bietet alle nötigen Features für die gemeinsame Entwicklung an einem Projekt. Hierbei sollte gesagt sein, dass der wahre Wert erst dann entsteht, wenn alle Teammitglieder auch Erfahrung mit GitHub haben. Da wir erst im 2. Semester sind, kann sich das nur verbessern und ist deshalb eigentlich schon fast Pflicht in zukünftigen Coding-Projekten.

Immer auf dem neuesten Stand zu sein ist wichtig, selbst wenn der jeweilige Ansprechpartner momentan nicht erreichbar ist. Dies kann durch ein Kanban Board realisiert werden. Durch diese agile Arbeitsweise wird der Fortschritt für alle sichtbar erfasst. Es gibt eine Vielfalt an Anbietern für diese Funktion, Trello oder der Tab Projekte bei einem GitHub Repository sind zwei nennenswerte Bereiche hierfür.

Im agilen Kontext lässt sich noch eine weitere empfehlenswerte Methode erwähnen, was in unserem Fall die Scrum Meetings waren. Diese haben uns geholfen die nächsten Schritte gemeinsam zu planen und zu koordinieren. Außerdem helfen Meetings dieser Art ungemein bei der Fehlerbehebung. Der mögliche Input von vielen zu einem Thema hilft schnell eine Lösung zu einem vermeintlich schwierigen Problem zu finden.

13 Anhang

Git-Repository: https://github.com/Big-P/Space-Invaders-aber-besser

Abb. 8: UML Handling

Abb. 9: UML Framework

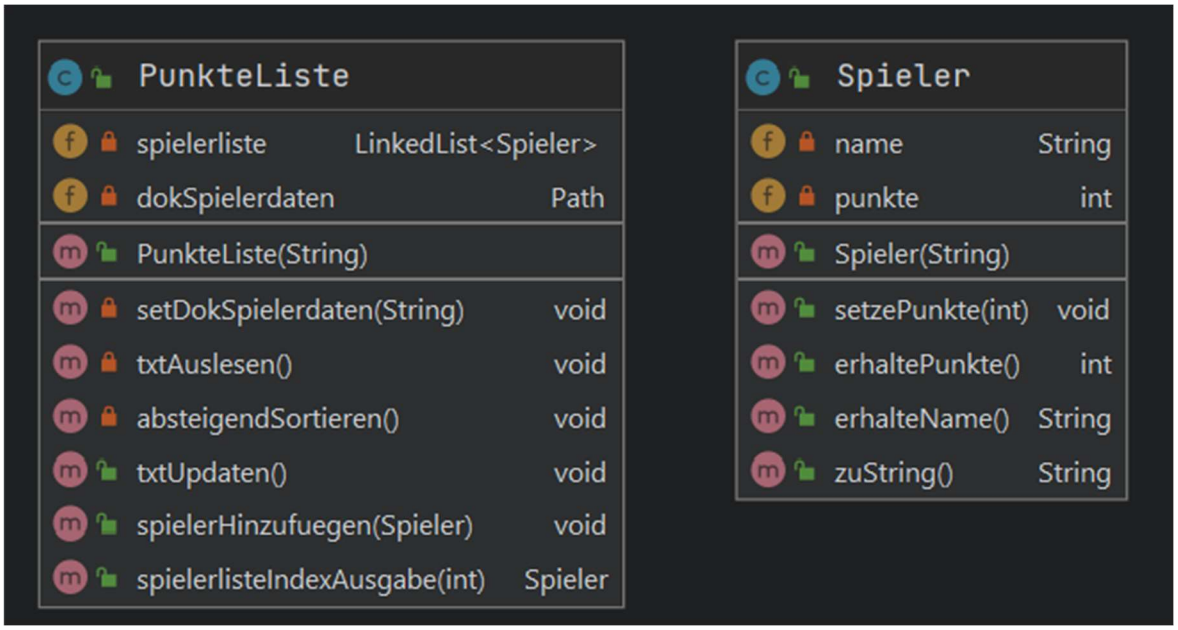

Abb. 10: UML Score

Abb. 11: UML Exception

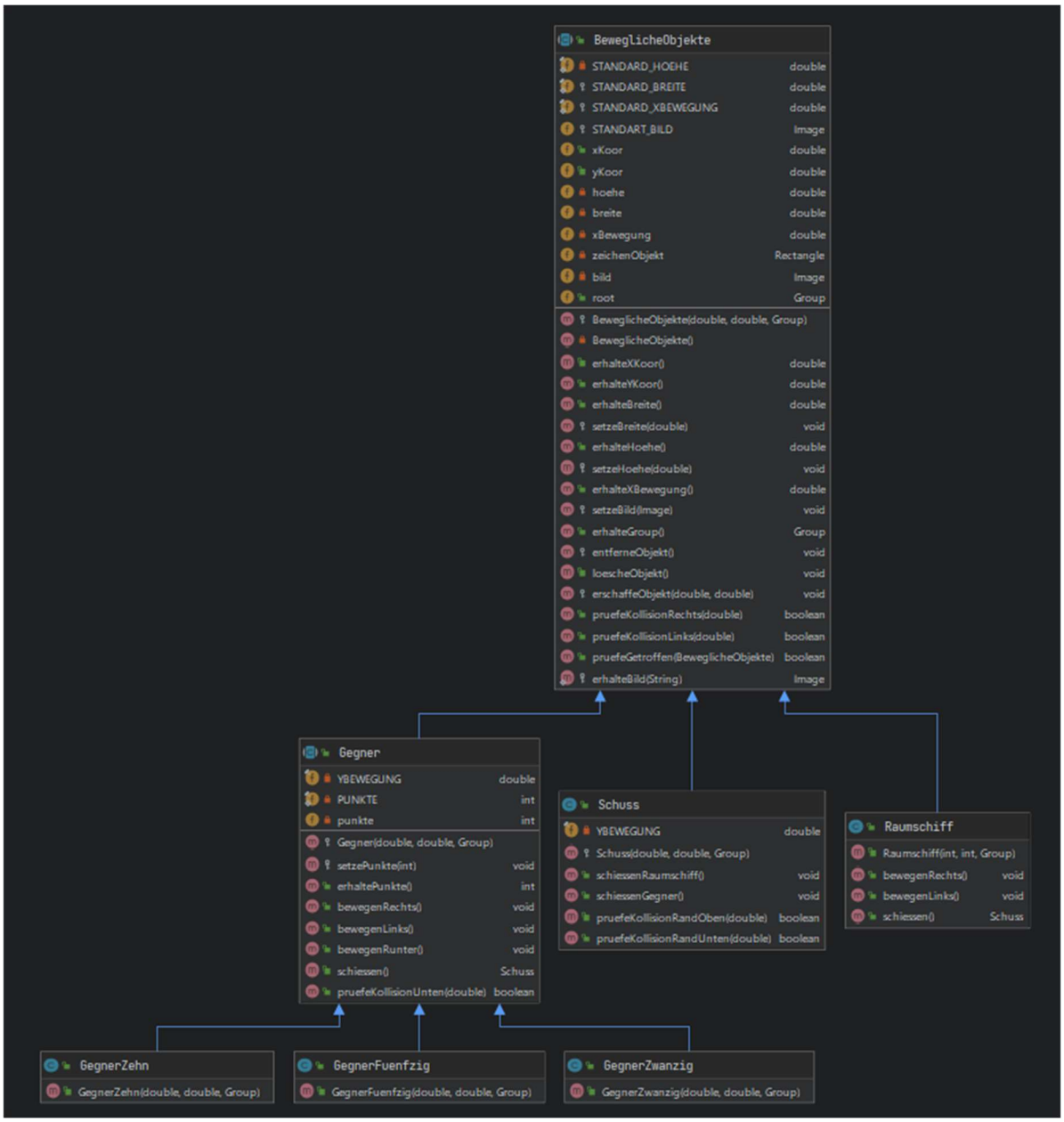

Abb. 12: UML Klassenhierachie