Ausarbeiten komplementärer Aufgaben und Methoden zur Steigerung der Motivation im Java Programmierunterricht

Osweiler Ronny

30. Januar 2018

Je, soussigné, Osweiler Ronny, déclare avoir réalisé ce travail par mes propres moyens.

Signature: _____

Nom: Osweiler

Prénom: Ronny

Fonction: candidat-professeur en informatique au Lycée des Arts et Métiers

Ausarbeiten komplementärer Aufgaben und Methoden zur Steigerung der Motivation im Java Programmierunterricht

Patron:

Fisch Robert

Lycée des Arts et Métiers 2018

Zusammenfassung

Die Motivation im Programmierunterricht lässt oft zu wünschen übrig. Dies könnte daran liegen, dass die Schüler nicht genug Visualisierung ihrer eigenen Klassen erhalten, oder aber den Kontext nicht genau erkennen.

In dieser Arbeit wurde ein Versuch unternommen das bestehende Programm *Unimozer* durch sogenannte interaktive Projekte zu erweitern um den Schülern bereits auf einer 3GTG grafische Interfaces zu zeigen und dadurch die Motivation Schülers und die Wertschätzung des Fachs zu steigern.

Die erstellte Plattform für interaktive Projekte, so wie die Projekte und deren Aufgaben wurden auf zwei 3GTG Klassen getestet, bei denen eine leichte Motivationssteigerung messbar war.

Inhaltsverzeichnis

1 Einleitung **9**

1.1 Ausgangspunkt und Zielsetzungen 9

1.2 Überblick 9

2 Bestandsaufnahme und theoretischer Rahmen **11**

2.1 Ausgangssituation 11

2.2 Motivation des Schülers zum Lernen und der Beteiligung am Unterricht 13

2.3 Gamification 15

2.4 Probleme beim Lernen einer Programmiersprache 16

2.5 Unimozer 18

2.6 Bestehende Arbeit in diesem Kontext 21

2.7 Bestehende Programme 23

2.7.1 Code Combat 24

2.7.2 Java-Hamster 26

2.7.3 Greenfoot 28

2.7.4 Moenagade 30

3 Methoden **33**

3.1 Geplante Erweiterung von Unimozer 33

3.1.1 Motivation zur Erweiterung 33

3.1.2 Geplante Umsetzung 33

3.2 Geplante Aufgaben und Projekte 34

3.2.1 Projekte die einen Controller erfordern 35

3.2.2 Projekte die ein Modell erfordern 36

4 Praktische Umsetzung **39**

4.1 Implementation in Unimozer 39

4.1.1 Erstellte Klassen 40
4.1.2 Speichern und Laden eines Projekts 42
4.1.3 Erstellen neuer interaktiver Projekte 44
4.2 Erstellte Projekte und Aufgaben 45
4.2.1 Bank Account 46
4.2.2 Slot Machine 48
4.2.3 Space Simulator 51
4.2.4 TurtleBox 56

5 Resultate **59**
5.1 Meinung der Schüler 59
5.2 Persönliche Beobachtungen 63
5.3 Meinung der Lehrer 64

6 Schlussfolgerung und Ausblick **65**

Bibliographie **68**

Anhang **69**

A Inhalt der Datei projects.xml **69**

B Erstellte Aufgaben **71**

C Erstellte Dokumentation für Lehrer **87**

Abbildungsverzeichnis

2.1 Interface des Scratch-Editors 12
2.2 UML Diagramm 19
2.3 Generieren von Quellcode 19
2.4 Nassi–Shneiderman Diagramm 20
2.5 Objekt erstellen 20
2.6 Methoden aufrufen 21
2.7 Übersicht über die verglichenen Programme 22
2.8 Auswahlbildschirm der Welten von *CodeCombat* 25
2.9 Level aus *CodeCombat* 25
2.10 Java-Hamster Interface 27
2.11 Interface von *Greenfoot* 28
2.12 Interface von *Moenagade* 30

4.1 QR-Code zum Quell-Code von *Unimozer* 39
4.2 Objectizer 42
4.3 Menü zum Erstellen neuer interaktiver Projekte 44
4.4 Bank Account Interface 46
4.5 UML der Klasse Account 47
4.6 Slot Machine Interface 49
4.7 Slot Machine UML Diagramm 50
4.8 Levelauswahl in Space Simulator 52
4.9 Toolbar von Space Simulator 52
4.10 UML Diagramm der Klasse Player 53
4.11 Erstes Level von Space Simulator 54
4.12 Zweites Level von Space Simulator 55
4.13 Drittes Level von Space Simulator 55
4.14 Viertes Level von Space Simulator 56

5.1 Umfrage vor den interaktiven Projekten 60
5.2 Umfrage nach den interaktiven Projekten 62

Kapitel 1

Einleitung

1.1 Ausgangspunkt und Zielsetzungen

Ich habe in den letzten Jahren festgestellt, dass ein großer Teil der Schüler der *Division Technique Générale* (11TG, 12GE & 13GE) nicht besonders am Informatikunterricht interessiert sind. Diese Klassen werden im Folgenden mit Ihren neuen Bezeichnungen seit 2017 benannt (3GTG, 2GGE & 1GGE). Des Weiteren sind sie auch sehr träge und nur schwer zu motivieren mithilfe des Computers Problemstellungen informatisch zu lösen. Dies erscheint mir in der aktuellen Zeit (cf. Rifkin Studie) jedoch sehr wichtig.

Ich will also in meiner Arbeit versuchen dieses Problem anzugehen. Ich will andere Methoden, komplementär zu den bereits bestehenden Methoden, ausarbeiten, die eine Motivationssteigerung mit sich bringen. Zusätzlich sollen neue Aufgaben eingebracht werden, die sich mit den motivationssteigernden Methoden vereinbaren lassen.

Es geht mir nicht darum neue Kursunterlagen zu erstellen, sondern neue Methoden und Wege zu finden um die Schüler motivierter an den Programmierunterricht heranzuführen. Ein meiner Meinung nach nicht zu vernachlässigender Faktor ist die hierbei einzusetzende Software. Ich will demnach das bereits bestehende Tool *Unimozer* erweitern.

1.2 Überblick

Folgend einen Überblick auf die gesamte Struktur dieser Arbeit:

- **Kapitel 2, Bestandsaufnahme und theoretischer Rahmen:** Dieses Kapitel umfasst die theoretischen Hintergründe die mit dem Programmierunterricht und der Motivation der

Schüler zusammenhängen, so wie eine Übersicht über das bestehende Programm *Unimozer* und andere Programme dieser Art.

- **Kapitel 3, Methoden:** Dieses Kapitel umfasst die Vorstellung der geplanten Erweiterung von *Unimozer*, so wie der geplanten Aufgaben und Projekte.
- **Kapitel 4, Praktische Umsetzung:** Dieses Kapitel beschreibt die Implementation die in *Unimozer* gemacht wurde und stellt ebenfalls die erstellten Projekte im Detail vor.
- **Kapitel 5, Resultate:** Dieses Kapitel präsentiert die Ergebnisse die während des Ausprobieren der Projekte gesammelt wurden und die persönlichen Beobachtungen die gemacht wurden und versucht diese zu interpretieren.
- **Kapitel 6, Schlussfolgerung und Ausblick:** Dieses Kapitel gibt einen Ausblick auf die geplante Zukunft des Projekts.

Kapitel 2

Bestandsaufnahme und theoretischer Rahmen

In diesem Kapitel werde ich eine theoretische Grundbasis errichten, welche sich mit den Schwierigkeiten, sowie der Motivation der Schüler im Programmierunterricht befasst. Ebenfalls wird das bereits im Unterricht verwendete Programm *Unimozer* vorgestellt. Dann werden weitere Programme auf deren Funktionen analysiert.

2.1 Ausgangssituation

In den folgenden Abschnitten lege ich die bestehenden Methoden der Klassen 3GTG, 2GGE & 1GGE dar, sowie das Programm der vorherigen Jahre.

Den ersten Kontakt erhalten die Schüler schon auf einer 5GTG, wo sie mittels der Block-Programmiersprache *Scratch*[1] einen visuellen Einstieg in die Welt der Programmierung erhalten. Hier wird ihnen eine direkte visuelle Darstellung der von ihnen programmierten Inhalte geboten. Sie werden zu keinem Zeitpunkt mit Quellcode konfrontiert. In der Abbildung 2.1 ist das Interface des Sratch-Editors, so wie ein Beispielprogramm, das aus Blöcken zusammengesetzt wird zu sehen.

Dies bietet den Schülern eine Möglichkeit mit schnellen Erfolgserlebnissen die logische Denkweise die zum Programmieren nötig ist zu erlernen. Diese Denkweise werden sie in den folgenden Jahren brauchen um im Java-Programmieren erfolgreich zu sein.

[1] `https://scratch.mit.edu/`

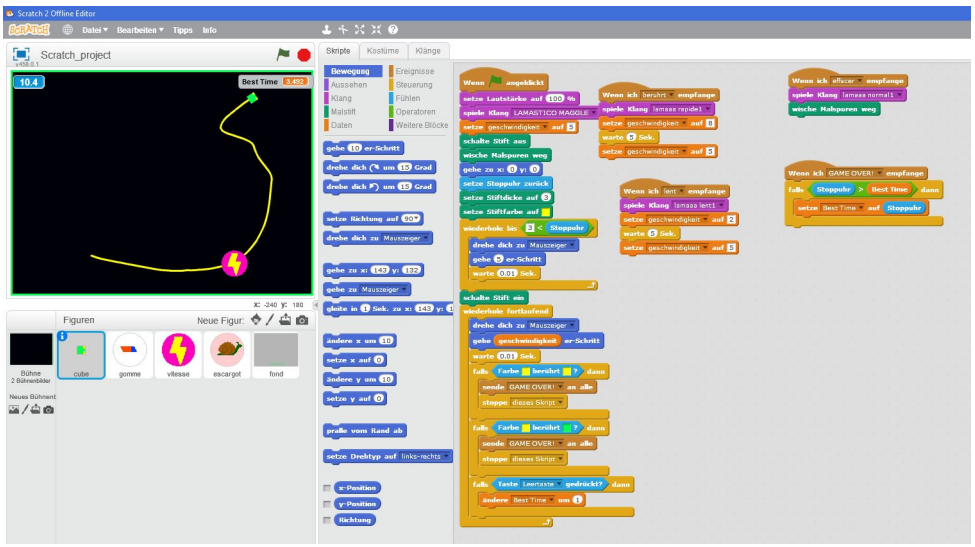


Abbildung 2.1: In diesem Screenshot ist das Interface des Scratch-Editors zu sehen. Die Programm-Teile werde mit Hilfe von Blöcken zusammengesetzt.

Im Folgejahr, auf einer 4GTG wird das Programmieren ein wenig vernachlässigt. Lediglich durch simple Excel-Formeln kommt der Schüler noch ein wenig in Kontakt mit der Programmierung. Ich meinerseits finde diese Lücke etwas unpassend, da es optimal wäre die durch Scratch erlernte Logik direkt in Java umzusetzen. Durch diese Unterbrechung wird kein direkter Übergang gewährt, der den Schülern den Einstieg in Java eventuell erleichtern würde. Dies steht jedoch in dieser Arbeit nicht zur Debatte.

Erstmals in Kontakt mit textueller Programmierung kommen die Schüler dann auf einer 3GTG, wo objekt-orientierte Programmierung unterrichtet wird. Hier kommt das Tool Unimozer zum Einsatz, das den Schülern einen leichteren Einstieg in die Programmierung bieten soll. Unimozer ermöglicht es erstellte Klassen einfacher zu testen und bietet ebenfalls eine grafisch unterstütze Erstellung der einzelnen Elemente in Java.

Hier kann der Schüler ganze Klassen, so wie deren Attribute und Methoden durch eine grafische Oberfläche sehr leicht erstellen. Dies bietet unter anderem die Möglichkeit, dass der Schüler einige Elemente generieren kann und sich dann den erstellten Quellcode ansieht und versuchen kann diesen nachzuvollziehen. Im Abschnitt 2.5 wird dieses Tool genauer beschrieben. Was den Schülern aber oft fehlt ist direktes visuelles Feedback der von ihnen erstellten Klassen, oder der Zusammenhang einzelner Klassen im Kontext eines größeren Programmes.

Auf einer 2GGE wird dann zu dem IDE Netbeans [2] gewechselt. Dies unter anderem, weil es das Erstellen von grafischen Interfaces ermöglicht. Die Schüler lernen in diesem Jahr hauptsächlich das Erstellen von grafischen Oberflächen, Listen und die dazugehörigen Algorithmen, das MVC-Modell und das Zeichnen auf einem Panel in der grafischen Oberfläche. Im Gegensatz zu der 3GTG Klasse, erstellen die Schüler hier schon selbst ihre eigenen Visualisierungen ihrer Programme. Sie bekommen oft direktes visuelles Feedback zu denen von ihnen erstellten Klassen.

Im letzten Schuljahr, der 1GGE, kommen zu dem bereits erlernten noch Mausevents, Timer und Vererbung hinzu. Wie auf der 2GGE sind die Schüler hier schon in der Lage grafische Interfaces zu erstellen.

Durch eigene Beobachtungen und Aussagen von Arbeitskollegen, kann ich feststellen, dass es den Schülern oft an Interesse am Informatikunterricht mangelt. Die Eigenmotivation der Schüler lässt oft zu wünschen übrig.

2.2 Motivation des Schülers zum Lernen und der Beteiligung am Unterricht

Die folgenden Abschnitte sind zum Teil angelehnt an meinen *Mémoire pédagogique* aus dem *Stage pédagogique* angehender Sekundarschullehrer [Osweiler, 2015].

Eines der Ziele dieser Arbeit ist es, durch eine Implementation in das bestehende Programm *Unimozer*, die Motivation der Schüler im Java-Unterricht zu steigern. Bevor man über motivationssteigernde Methoden diskutieren kann, muss man zuerst feststellen was Motivation überhaupt ist. [Deci, 1971] definiert zwei Arten von Motivation die einen dazu veranlassen können eine Aktivität zu unternehmen, diese werden durch [Entwistle, 1998] durch eine dritte erweitert:

- **Intrinsische Motivation:** Wenn eine Aktivität von einer Person nur durch eigene Motivation durchgeführt wird, so ist diese Person intrinsisch motiviert. Dies geschieht durch Interesse an der Aktivität selbst [Deci, 1972], [White, 1959], [Entwistle, 1998]. Im Programmierunterricht bezieht sich das auf das Interesse am Fach selbst [Jenkins, 2001].
- **Extrinische Motivation:** Diese Beweggründe kommen, im Gegensatz zur intrinsischen

[2] `https://netbeans.org`

Motivation von außen und nicht von der Person selbst. Diese Motivation kommt durch eine externe Belohnung, die verschiedener Arten sein können [Deci, 1972],[Entwistle, 1998].

- **Leistungsbezogene Motivation:** (vom englischen *achievement motivation*) Diese basiert auf dem Verlangen gut abzuschneiden, zum Beispiel in einem schulischen Kontext, und manchmal auch auf dem Verlangen besser als andere, sogenannte *peers*, zu sein [Entwistle, 1998].

Dies sind die drei Arten von Motivation, die sich bei einem Schüler auf seine Beteiligung und sein Interesse am Kurs auswirken. [Fallows and Ahmet, 1999] stellte noch eine vierte Art von Motivation auf, dem Verlangen dem Lehrer zu gefallen, welche in der hier gegebenen Liste nicht vorkommt [Jenkins, 2001]. Diese kann ebenfalls erweitert werden um generell das Verlangen auszudrücken einer dritten Partei zu gefallen, welche zum Beispiel auch die Eltern darstellen können [Jenkins, 2001]. Diese Art der Motivation wird von [Biggs, 2011] auch **soziale Motivation** genannt. Hauptsächlich bei Frauen drückt sich diese *soziale Motivaion* oft unter der Form der Angst vor dem Scheitern aus [Greasley, 1998], [Jenkins, 2001].

In meinem *Mémoire pédagogique* [Osweiler, 2015] versuchte ich die extrinsische Motivation des Schülers im Programmierunterricht durch die Vergabe digitaler Abzeichen zu steigern, was sich ebenfalls positiv auf die intrinsische Motivation des Schülers auswirken kann [Deci, 1971]. In dieser Arbeit hingegen versuche ich die intrinsische Motivation des Schülers auf direkte Art und Weise zu steigern.

Ein anderes Modell, das versucht die Motivation die man für etwas ein Fach darzustellen, ist das *Erwartung-mal-Wert-Modell* (Eng: Expectancy-value theory) von [Atkinson, 1957] und [Wigfield and Eccles, 2000]. Dieses Modell besagt, dass zwei Faktoren für die Motivation eines Schülers ausschlaggebend sind:

- **Erfolgserwartungen:** Die Erwartungen für ein erfolgreiches Abschließen eines Schulfachs
- **Wertschätzung:** Die Wertschätzung die der Schüler einer gewissen Aktivität oder einem Fach zuschreibt. Dies ist ein subjektiver Wert der auf den Lernenden zutrifft.

Die Leistungsmotivation (M) für eine Aktivität ist von den Erfolgserwartungen (E) derselben und einem subjektiven Wert (W) den der Schüler der Aktivität zuschreibt abhängig, das Modell wird dadurch oft durch die Formel $M = W \cdot E$ dargestellt.

Dies bedeutet, dass die Motivation für einen Kurs sowohl von den Erfolgserwartungen als auch der Wertschätzung des Faches abhängig ist. Wenn nur einer der beiden Werte gegen Null tendiert, so tendiert auch die Motivation gegen Null [Jenkins, 2001]. Dies bedeutet unter anderem, dass es möglich sein muss ein positives Resultat am Ende des Kurses zu erhalten, mit einer vernünftigen Arbeitsinvestition des Schülers. Ansonsten kann der Wert den der Schüler dem Fach zuschreibt so hoch sein wie er will, wenn die Erfolgsaussichten des Schülers gegen Null tendieren, wird auch die Motivation des Schülers gegen Null tendieren.

In dieser Arbeit Versuche ich hauptsächlich die intrinsische Motivation der Schüler zu steigern, dadurch dass sie schneller visuelle Ergebnisse erlangen und ebenfalls dadurch dass die angewandten Methoden und Aufgaben mehr Gefallen bei den Schülern haben. Ebenfalls hoffe ich, dass dadurch die Wertschätzung, beziehungsweise das Interesse das der Schüler am Fach hat, gesteigert werden.

2.3 Gamification

Gamification ist ein Wort, das immer mehr in der pädagogischen Fachliteratur aufkommt. Laut [Deterding et al., 2011] tauchte das Wort *Gamification* vermehrt im Jahre 2008 auf. Das Thema wurde ebenfalls in meinem *Mémoire pédagogique* [Osweiler, 2015] behandelt, wo durch den Einsatz von digitalen Abzeichen eine Gamification eines Projektes in der Scratchprogrammation vorgenommen wurde. Laut [Deterding et al., 2011] kann man *Gamification* wie folgt definieren:

> *To summarize: "Gamification" refers to*
>
> - *the use (rather than the extension) of*
> - *design (rather than game-based technology or other gamerelated practices)*
> - *elements (rather than full-fledged games)*
> - *characteristic for games (rather than play or playfulness)*
> - *in non-game contexts (regardless of specific usage intentions, contexts, or media of implementation)*
>
> *(*[Deterding et al., 2011]*)*

So kann man zusammenfassen, dass *Gamification* der Einsatz ist von spielerischen Elementen in einem Kontext der kein Spiel ist. [Deterding et al., 2011] kommen außerdem zur Schlussfolgerung, dass es objektiv schwierig ist ein Empirisches System zu analysieren und festzustellen ob es

sich um eine Art von *Gamification* oder nicht handelt. Vielmehr sind die subjektiven Absichten des Erstellers, so wie die Aufnahme durch den Benutzer ausschlaggebend dafür.

Eine andere interessante Auffassung von *Gamification* kommt von [Huotari and Hamari, 2011], wo *Gamification* mit *Service Marketing* verglichen wird. Ihre Definition von *Gamification* ist die folgende:

> *Gamification is a form of service packaging where a core service is enhanced by a rules-based service system that provides feedback and interaction mechanisms to the user with an aim to facilitate and support the users' overall value creation.*
>
> *(*[Huotari and Hamari, 2011]*)*

Dies bedeutet, dass man dadurch ein bestehendes System, wie zum Beispiel das Programm *Unimozer* erweitern könnte, durch ein regelbasiertes System, das dem Benutzer eine Interaktion, so wie ein Feedback ermöglicht und so die gesamte Erarbeitung des Schülers zu wertschätzen. Dies ist etwas, das ich mit einem ganz bestimmten interaktiven Projekt versuchen werde, dies wird in Abschnitt 4.2.3 beschrieben.

Bestehende Programme, wie zum Beispiel *CodeCombat*, beschrieben in Abschnitt 2.7.1, und Java-Hamster, beschrieben in Abschnitt 2.7.2, versuchen den Schülern das Programmieren auf eine spielerische Art und Weise näher zu bringen. *CodeCombat* geht dabei sogar sehr weit, indem viele Elemente eines Spiels eingebaut werden, was vielleicht den Erfolg der Plattform bei den Schülern erklären kann.

2.4 Probleme beim Lernen einer Programmiersprache

In diesem Abschnitt werde ich darlegen, was die eigentlichen Probleme sind für Schüler die eine Programmiersprache lernen solle, so wie die pädagogischen Hintergründe die im Unterricht wichtig sind.

Laut [Winslow, 1996] kommen viele vorherige Studien zu der Feststellung, dass Anfänger im Programmieren (Eng: *novice programmers*) oft nach einiger Zeit die Syntax und Semantik einzelner Statements kennen, aber oft unfähig sind diese zu funktionstüchtigen Programmen zusammenzuführen. Dies ist ein Phänomen das ich ebenfalls oft bei den Schülern beobachten kann. Ebenfalls sagt [Winslow, 1996] aus, dass obwohl ein Anfänger in der Lage ist ein Problem

selbstständig auf Papier zu lösen, hat dieser oft Probleme diese Lösung in einem passenden Computerprogramm umzusetzen. Über Anfänger sagt [Winslow, 1996], basierend auf vorherigen Studien, folgendes aus:

- Anfängern fehlt ein angemessenes mentales Modell des Gebiets der Programmierung [Kessler and Anderson, 1986].
- Anfänger sind auf ein oberflächliches Wissen der Programmierung begrenzt.
- Anfänger haben nur flüchtiges Wissen der Programmierung, an das sie nicht unbedingt denken wenn sie es brauchen [Perkins and Martin, 1986].
- Anfänger benutzen allgemeine Problemlösungsstrategien anstelle von denen die für die Lösung eines Problems angemessen sind.
- Anfänger tendieren dazu sich der Lösung durch das benutzen ihnen bekannten Kontrollstrukturen zu nähern
- Anfänger versuchen sich der Lösung Zeile für Zeile zu nähern, anstatt das Problem global zu sehen [Anderson, 1985].

Ebenso stellen die Autoren fest, dass eines der Hauptprobleme, unabhängig vom Niveau des Lernenden, nicht die Syntax ist, sondern das Kombinieren der einzelnen Statements um die Lösung für ein Problem zu erreichen. Ebenso reicht das Studieren der Syntax und Semantik einer neuen Programmiersprache nicht aus um diese zu verstehen und effektiv einzusetzen. Zusätzlich bemerken die Autoren, dass die Progression einer Person die eine Programmiersprache lernt stark abhängig ist vom Lösen von Problemen. Dies bedeutet, dass die Schüler nur Fortschritte erreichen können, wenn diese motiviert werden eigenständig Probleme zu lösen.

Eine sehr interessante Schlussfolgerung zu der [Winslow, 1996] kommen ist, dass es ungefähr 10 Jahre dauert um aus einem Anfänger der Programmierung einen Experten zu machen. Da der Programmierunterricht in unserem Schulsystem auf drei Jahre begrenzt ist, muss, wie von [Winslow, 1996] vorgeschlagen, das beste unternommen werden um den Schülern die Kenntnisse mit auf den Weg zu geben um die restliche Praxis selbst zu machen.

[Linn and Dalbey, 1989] schlagen vor, dass es einen Vorgang gibt der ideal ist um eine neue Programmiersprache zu lernen:

1. Die Syntax und Semantik einer Sprache zu lernen, eine Eigenschaft nach der anderen
2. Lernen diese Eigenschaften zur Lösung eines Problems zu kombinieren, indem bestehende Kenntnisse verwendet werden. Dadurch werden diese Kenntnisse durch die Eigenschaft erweitert.
3. Die generellen Problemlösungsansätze verbessern.

So sollte idealerweise beim Ausarbeiten neuer Aufgaben, dieses Prozedere eingehalten werden. Ebenfalls trifft dies auf die meisten der bereits bestehenden Aufgaben die durch die CNPI zur Verfügung gestellt werden zu. So werden immer nach und nach einzelnen neue Features von *Java* eingeführt und dann in verschiedenen Aufgaben angewendet um so die Problemlösungsfähigkeiten der Schüler zu erweitern.

2.5 Unimozer

In diesem Abschnitt wird das bereits bestehende Tool Unimozer [Fisch, 2009] vorgestellt. Ich gehe auf die wichtigsten und grundlegenden Funktionen des Programms ein.

Dieses Tool wird auf den 3GTG Klassen benutzt um den Schülern einen leichteren Einstieg in die Programmierung zu ermöglichen. Das Programm ist an das bestehende Programm *BlueJ* [3] angelehnt, wurde aber für die Bedürfnisse des luxemburgischen Schulsystems angepasst. Es folgt eine kurze Übersicht über die Funktionen des Programms:

UML Diagramme: Das Programm ermöglicht es UML Diagramme (Unified Modeling Language) eines Prorgrammes anzuzeigen. Dies ermöglicht dem Schüler einen besseren Überblick über das Programm. Sobald etwas im Quellcode einer Klasse verändert wird, spiegelt sich diese Veränderung im UML Diagramm wieder. Quellcode und UML Diagramm bleiben jederzeit synchronisiert. Ebenfalls werden die Relationen (z.B. Vererbung, Komposition,...) zwischen den einzelnen Klassen im Diagramm dargestellt. Ein Beispiel ist in Abbildung 2.2 zu sehen.

Erleichtertes Erstellen von Klassen: Dem Benutzer wird es ermöglicht ganze Klassen, Attribute, Methoden und Konstruktoren auf vereinfachte Art und Weise zu erstellen. Anstatt selbst Quellcode schreiben zu müssen, werden die wichtigsten Elemente automatisiert erstellt. Dies

[3] `https://www.bluej.org`

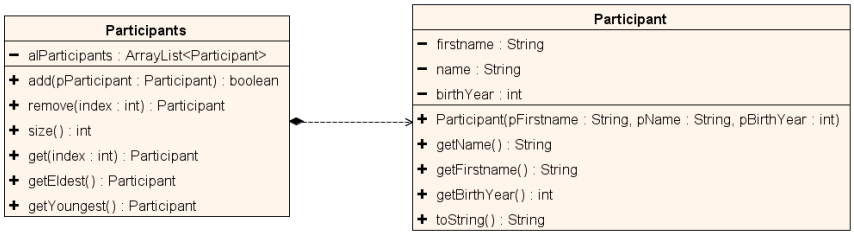
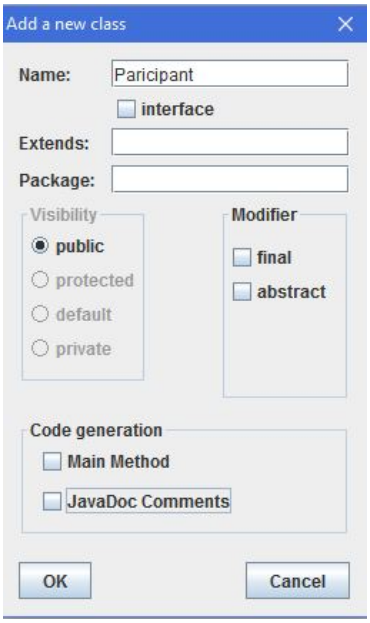
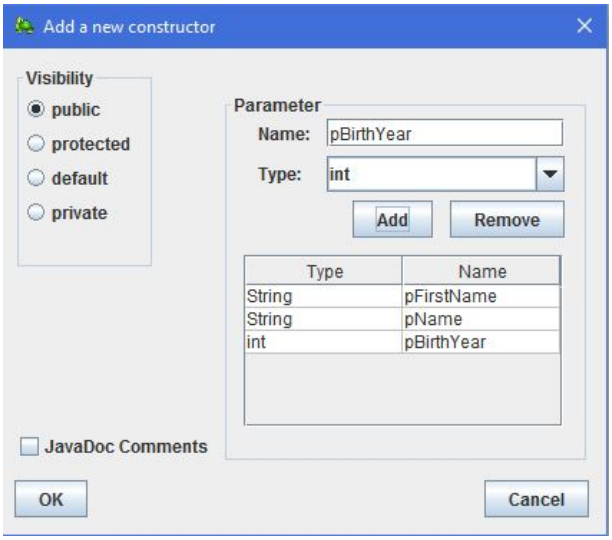


Abbildung 2.2: Ein Beispiel eines durch Unimozer erstellten UML Diagramms. Hier wird eine Komposition zwischen den beiden Klassen gezeigt.

ermöglicht es dem Benutzer den Quellcode generieren zu lassen, um sich diesen dann anzusehen und nachvollziehen zu können. Einige Beispiele dieser geführten Erstellung sind in Abbildung 2.3 zu sehen.



(a) Hier wird das Erstellen einer Klasse mit Unimozer gezeigt. Der Benutzer kann hier alle erforderlichen Optionen auswählen.



(b) Hier wird gezeigt wie der Konstruktor einer Klasse mit Unimozer erstellt werden kann. Der Benutzer kann die einzelnen Parameter durch das Menü hinzufügen.

Abbildung 2.3: Diese Bilder zeigen wie der Benutzer automatisiert Quellcode generieren lassen kann.

NS-Diagramme: In Unimozer ist das Programm *Structorizer*[4] eingebunden. Dadurch wird das Anzeigen von Nassi–Shneiderman Diagrammen (kurz. NS-Diagramme) ermöglicht [Nassi and Shneiderman, 1973]. Dies sind Diagramme die es ermöglichen den Verlauf einer Methode zu visualisieren. Die NS-Diagramme sind ein Bestandteil des Programmes der 3GTG, 2GGE und

[4] `http://structorizer.fisch.lu/`

1GGE. Diese treten dort unter dem Namen Struktogramm auf. Die Schüler sollen in der Lage sein, diese interpretieren zu können. Ein Beispiel eines solchen Struktogrammes ist in Abbildung 2.4 zu sehen.

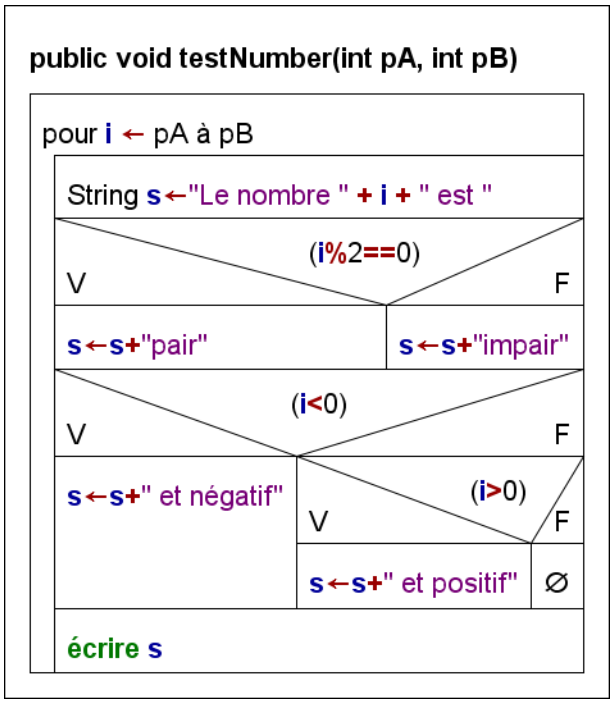


Abbildung 2.4: Ein Beispiel eines Nassi–Shneiderman Diagrammes, so wie es auf einer 3GTG benutzt wird.

Kompilieren und Objekte erstellen: Unimozer ermöglicht es die erstellten Klassen zu kompilieren und Objekte aus diesen zu erstellen. Dazu kann man einen der Konstruktoren der Klasse auswählen und mithilfe eines Menüs erstellen. Dies ist zu sehen in Abbildung 2.5.

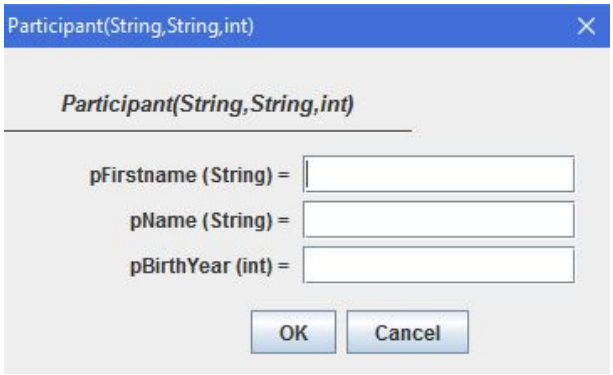


Abbildung 2.5: Hier wird gezeigt wie man mithilfe eines Menüs einfach ein Objekt erstellen kann.

Methoden aufrufen: Der Benutzer kann von aus Klassen erstellten Objekten alle öffentlichen Methoden aufrufen und ausführen. Hierzu werden alle erstellten Objekte im unteren Bereich des

Programms angezeigt. Hier sieht man den Namen und den Typ des Objekts, so wie all dessen Attribute. Die öffentlichen Methoden des Objekts werden bei einem Rechtsklick auf das Objekt angezeigt. Diese sind des weiteren unterteilt, je nach Vaterklasse von der sie jeweils abstammen. Der Benutzer kann diese Methoden dann ausführen. Auch Methoden die Parameter benötigen können so ausgeführt werden. Hier öffnet sich dann ein Fenster, das den Benutzer nach den Parametern befragt. Ein Beispiel des Aufrufens einer Methode ist in Abbildung 2.6 zu sehen.

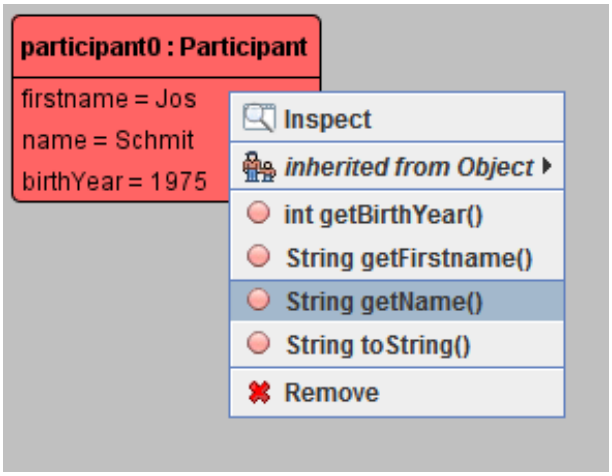


Abbildung 2.6: Hier wird gezeigt wie erstellte Objekte angezeigt werden. Diese ermöglichen es die öffentlichen Methoden eines Objekts aufzurufen.

2.6 Bestehende Arbeit in diesem Kontext

Zu der hier diskutierten Problematik gibt es bereits einen *Tavail de Candidature*:

> *Analyse de la valeur pédagogique ajoutée d'un outil de programmation Java en vue d'une introduction éventuelle au programme des classes du régime technique de la division technique générale.* *(*[Goetz, 2015]*)*

Dieser beschäftigt sich mit der Suche nach einem geeigneten Tool um den Programmierunterricht grafisch zu unterstützen und somit für Schüler leichter verständlich zu machen. Ebenso sollte die Motivation die die Schüler im Programmierunterricht an den Tag legen hierdurch gesteigert werden.

Konkret wurden die folgenden Programme unter die Lupe genommen und deren Funktionalitäten miteinander verglichen:

- Ceebot

- Jeroo
- CodeSpell
- JavasCool
- KaraJava
- Greenfoot
- Java-Hamster

Die Autorin hat jedes dieser Programme kurz ausprobiert und für jedes einzelne Vor- und Nachteile aufgestellt. Ebenfalls wurde deren Einsetzbarkeit in den einzelnen Klassen der *Division Générale* analysiert. So konnte die Autorin eine Tabelle mit den Stärken und Schwächen der einzelnen Tools aufstellen. Ein anderer Faktor der bei der Benutzung der Programme eine wichtige Rolle spielt ist der Preis. Das ideale Programm sollte frei zugänglich sein. Um einen Überblick zu erhalten hat die Autorin eine Matrix aufgestellt, welche die wichtigsten Informationen über die getesteten Programme enthält. Diese ist in Abbildung 2.7 zu finden.

OUTIL	Prix	Résumés des notions vues en 11e	Utilisable avec Scratch	Utilisable avec Structogramme HaSE	Utilisable après la 11e TG
Ceebot	75/750 CHF	X			
Jeroo	Gratuit	X			
CodeSpell	19,99 $				
JavasCool	Gratuit	X			
KaraJava	Gratuit		X	X	X
Greenfoot	Gratuit				
Java-Hamster	Gratuit		X	X	X

Abbildung 2.7: Auszug von [Goetz, 2015], wo die verschiedenen Tools miteinander verglichen und deren Funktionalitäten gezeigt werden.

Dann suchte die Autorin sich eines dieser Tools aus um es genauer unter die Lupe zu nehmen. Die Kriterien für das Tool waren:

- Die Einsetzbarkeit sollte durch das *Horaire et Programmes* gewährleistet sein und auf den 3 Klassen 11TG, 12GE und 13GE einsetzbar sein.

- Das Tool sollte gratis sein und kompatibel mit allen gängigen Betriebssystemen sein, so dass jeder Schüler den gleichen Zugang zum Tool hat.

Im Endeffekt entschied sich die Autorin, aufgrund der Kriterien, das Tool Java-Hamster näher zu analysieren. In diesem Kontext entwarf sie Aufgaben für Java-Hamster die in den Kontext des Programmes der 11TG Klassen passen. Allerdings sind die Aufgaben bislang nur durch die Arbeit selbst zugänglich und nicht auf der offiziellen Webseite der CNPI zur Verfügung gestellt. Somit sind sie für die Meisten Sekundarschullehrer unzugänglich, beziehungsweise diese wissen nicht einmal dass die Arbeit existiert. Ein Ziel meiner eigenen Arbeit wäre die Erstellten Aufgaben zur Verfügung zu stellen.

Die Autorin befasst sich mit zwei Hauptfragen:

- Wie wirkt sich der Einsatz des Java-Hamster Programmes auf die Motivation der Schüler aus?
- Wie wirkt sich der Einsatz des Java-Hamster Programmes auf die schulischen Resultate des Informatikunterrichts aus?

Um diese Fragen zu beantworten, wurden die Aufgaben auf mehreren Klassen durchgeführt um die Reaktion der Schüler, so wie deren Motivationssteigerung zu messen. Ebenso gab es eine Kontrollklasse, die ohne dieses Tool und die erstellten Aufgaben arbeitete, die dazu diente eine Referenz zur Einstufung der Motivation der anderen Klassen darzustellen.

Die Autorin kommt zu dem Schluss, dass es aktuell kein Tool auf dem Markt gibt, das den Erwartungen des luxemburgischen Schulsystems entspricht, außer das aktuell benutzte Tool *Unimozer*, welches spezifisch hierfür entwickelt wurde. Deshalb empfiehlt die Autorin das Ausarbeiten eines eigens für unser Schulsystem entworfenen Tools, oder das Einbinden in *Unimozer* des gewünschten Funktionen. Das will ich in meiner Arbeit realisieren.

2.7 Bestehende Programme

In diesem Abschnitt werde ich eine Analyse einiger bereits existierender Tools machen und sehen ob diese geeignet sind im Unterricht benutzt zu werden. Wie bei [Goetz, 2015], werde ich darauf achten ob diese Programme kostenfrei zur Verfügung stehen. Den Hauptschwerpunkt lege ich hier allerdings auf die Umsetzbarkeit bei der Klassen 3GTG. Da es in dieser Arbeit darum geht die Motivation durch eine bessere Visualisierung zu erreichen, beschränke ich mich auf diese

Klassen. In den folgenden Jahren, kreieren die Schüler ihre eigenen grafischen Interfaces. Ebenfalls werde ich nicht alle von [Goetz, 2015] analysierten Programme noch einmal analysieren, sondern mich auf einige beschränken die für mich wichtig sind. Ebenfalls werden noch weitere Programme analysiert, welche nicht in [Goetz, 2015] vorkommen.

2.7.1 Code Combat

Eine Webseite in diesem Kontext die ich erwähnen möchte ist *CodeCombat*[5]. Diese Webseite ermöglicht es Nutzern spielerisch eine Programmiersprache zu lernen. Diese unterstützt leider bislang Java noch nicht als Programmiersprache und kommt daher nicht in Frage für den Einsatz auf einer *3GTG*, aber ich möchte sie trotzdem erwähnen, da sie meiner Meinung nach einen gelungenen Ansatz bietet um einem jungen Publikum eine Programmiersprache beizubringen. Einige der Funktionen und Ideen des Programms könnten für eine spätere Implementation in *Unimozer* von Nutzen sein.

Aus eigener Erfahrung kann ich sagen, dass *CodeCombat* gut bei den Schülern ankommt. Ich habe diese Webseite selbst mit Schülern des *cycle inférieur* im *Code Club* des *Lycée des Arts et Métiers*, einer nebenschulischen Aktivität die den Schülern das Programmieren näher bringen soll, benutzt. Die Schüler wirken sehr interessiert und scheinen ebenfalls sehr selbstständig mit den ihnen gestellten Aufgaben klar zu kommen. Das Feedback, das durch die Schüler erhalten wurde war weitestgehend positiv.

Zu Beginn hat der Schüler hier eine Auswahl an verschiedenen Welten, die je nach Schwierigkeitsgrad verschiedene Aufgaben bieten, zu sehen auf Abbildung 2.8. Jede dieser Welten ist in mehrere Level eingeteilt, die der Schüler lösen muss um weiter zu kommen. Jeder der Welten deckt verschiedene Elemente einer Programmiersprache ab.

Die Level werden progressiv immer schwieriger und bringen immer wieder neue Elemente der Programmierung ein. Anfangs muss sich der Held nur bewegen um die Edelsteine einzusammeln, später muss er zum Beispiel auch noch Gegner besiegen. Eines dieser Level ist in Abbildung 2.9 zu sehen.

[5] `https://codecombat.com/`



Abbildung 2.8: In dem Lernspiel *CodeCombat* stehen verschiedene Welten zur Verfügung, die jeweils neue Elemente einer Programmiersprache einbringen.

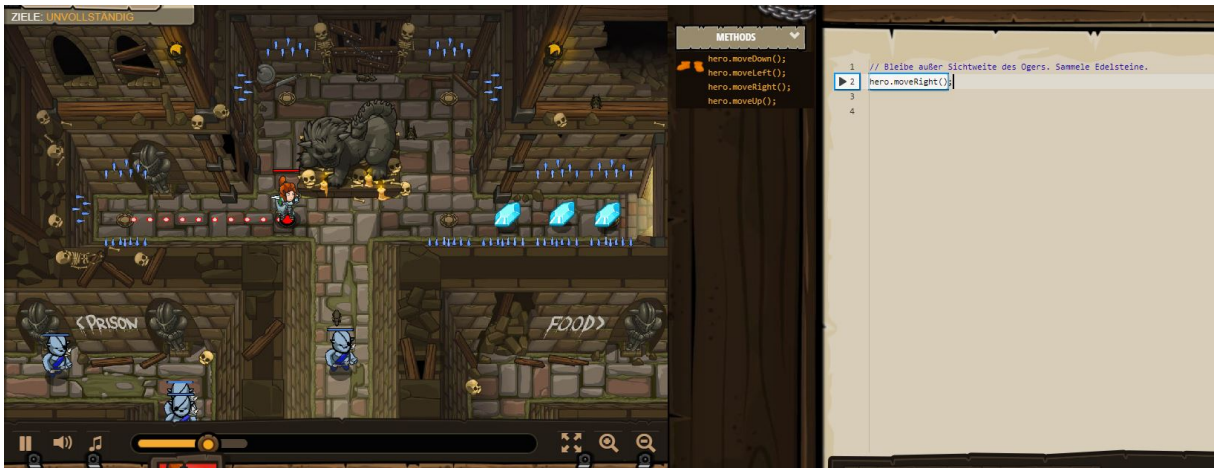


Abbildung 2.9: Ein Level aus *CodeCombat*. Der Schüler muss hier alle Edelsteine sammeln und den Gegner besiegen.

CodeCombat unterstützt mehrere Programmiersprachen, unter anderem *JavaScript* und *Python*. Die Programmiersprache *Java* wird allerdings nicht unterstützt, sonst hätte *CodeCombat* eventuell für den Programmierunterricht zum Einsatz kommen können. Hinzu kommt, dass *CodeCombat* nicht gratis für Schulen zur Verfügung steht, sondern dem Kauf einer Lizenz bedarf. Ein Vorteil ist, dass *CodeCombat* leicht für alle Schüler zugänglich ist, da es keiner Installation bedarf. Es kann einfach im Browser aufgerufen werden.

Zusätzlich hat *CodeCombat* ein System in dem der Schüler einen Avatar, ein Charakter der für den Spieler selbst steht, besitzt. Dieser kann vom Schüler personalisiert werden. Ebenso kann

er verschiedene Ausrüstungsgegenstände sammeln um seinen Charakter auszurüsten. Die einzelnen Spieler können sich außerdem in verschiedene Clans zusammenschließen. Alles in allem stellt *CodeCombat* eine Art von *Gamification* dar, die näher in Abschnitt 2.3 beschrieben wird.

Trotzdem gibt es eine Eigenschaft von *CodeCombat*, die es zum Großteil unbrauchbar macht. Es gibt hier ein Bezahlsystem, wo sich der Schüler mit Geld Gegenstände kaufen kann, um sich so bessere Gegenstände für seinen Charakter zuzulegen. Dies ist etwas was ich als ungeeignet für die Benutzung in unserem Schulsystem halte.

2.7.2 Java-Hamster

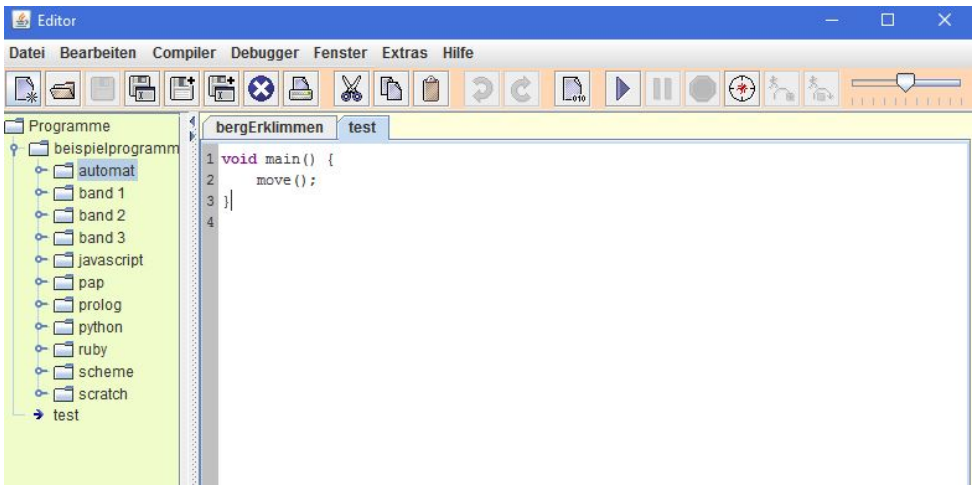
Einen ebenfalls interessanten Ansatz an die Programmierung stellt für mich das Programm *Java-Hamster* dar, das ebenfalls von [Goetz, 2015] unter die Lupe genommen wurde. In diesem Programm soll der Schüler einen Hamster durch ein Level steuern, ähnlich wie es bei *CodeCombat* der Fall ist. Für den *Java-Hamster* existieren bereits mehrere Bücher[6], die Aufgaben zu den verschiedenen Elementen einer Programmiersprache enthält.

Das Interface des Programms ist in zwei Fenster aufgeteilt. Eines der Fenster, zu sehen in Abbildung 2.10a zeigt einen Code Editor, der es dem Schüler ermöglicht direkt seine Befehle für den Hamster einzugeben. Hier werden auch Kontrollstrukturen von Java unterstützt. Ebenfalls gibt es hier die meisten Funktionen die ein Code-Editor unterstützt, allerdings leider keine Auto-Vervollständigung. Dies ist allerdings ein Feature das sinnvoll wäre.

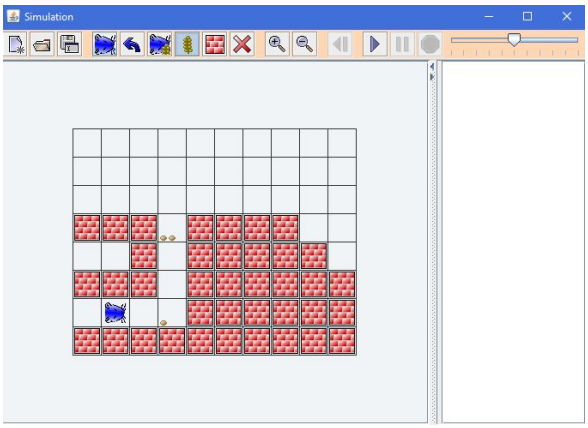
In Abbildung 2.10b ist das Spielfeld zu sehen. Dies kann vom Benutzer jederzeit verändert werden. So können Mauern, der Hamster selbst und Körner platziert werden, die es einzusammeln gilt. Diese Level können gespeichert und geladen werden. So ist es möglich, dass der Lehrer ein Level vorbereitet und es seinen Schülern zur Verfügung stellt. So können spezifische Aufgaben erstellt werden.

In der Arbeit von [Goetz, 2015] sind einige erstellte Aufgaben zu finden, die auf die Klassen der *Division Générale* passen. Ebenfalls wurde von [Goetz, 2015] eine pädagogische Hilfe für die Lehrer erstellt, die das Programm *Java-Hamster* im Unterricht verwenden wollen. Die erstellten Aufgaben reichen von einfachen sequenziellen Befehlen bis hin zum Sortieren von Elementen. Somit sind die Aufgaben sowohl auf der 3GTG, als auch auf einer 2GGE einsetzbar.

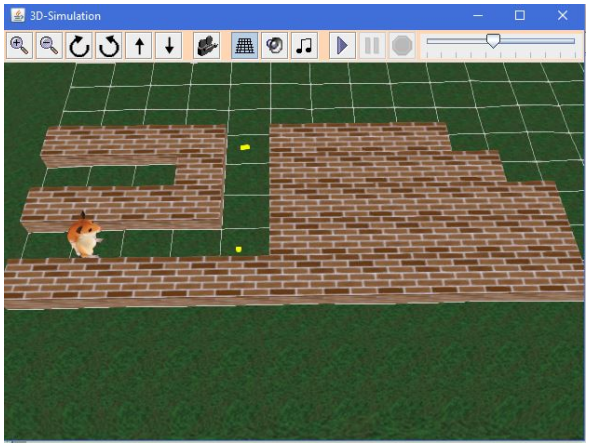
[6] `http://www.java-hamster-modell.de/buecher.html`



(a) Hier sieht man den Code Editor von *Java-Hamster*



(b) Hier wird das Spielfeld von *Java-Hamster* gezeigt. Der Hamster kann durch den Code des Schülers bewegt werden. Ebenfalls kann das Spielfeld vom Benutzer verändert werden.



(c) Hier wird das Spielfeld von *Java-Hamster* gezeigt in 3D gezeigt. Es zeigt das gleiche Level wie das 2D-Spielfeld.

Abbildung 2.10: Diese Bilder zeigen wie das Interface von *Java-Hamster* dargestellt wird.

Ein großer Nachteil des Programms ist, dass die Befehle, beziehungsweise die Methoden, durch die der Hamster nicht sofort für den Benutzer sichtbar sind. Diese sind ausschließlich in der Dokumentation zu *Java-Hamster* zu finden. Dies finde ich nicht gut, da der Schüler auf diese Art und Weise keinen guten Überblick erhält. Besser wäre es wenn die Methoden direkt für den Benutzer sichtbar wären.

Die erstellten Dokumente von [Goetz, 2015] enthalten unter anderem eine Installationsanleitung des Programmes für die Schüler, so wie eine Dokumentation zum Programm. Diese enthält eine Übersicht über die wichtigsten Methoden für die Schüler, so dass diese nicht in der offizielle Dokumentation nachzusehen brauchen. Trotzdem bin ich der Meinung, dass es besser wäre diese

Methoden direkt im Programm auf einem Blick zu haben. Eine Methode die ich mir hier vorstellen könnte, wäre die Methoden wie in *Unimozer*, zu sehen auf Abbildung 2.6, direkt anzuzeigen.

2.7.3 Greenfoot

Ein Programm das ebenfalls von [Goetz, 2015] analysiert wurde ist *Greenfoot*[7]. Dieses Programm wurde an der *University of Kent* entwickelt und wird von *Google* und *Oracle* finanziell unterstützt. Das Programm benutzt *Java* als Programmiersprache und wäre somit geeignet für den Einsatz auf einer 3GTG. In diesem Programm gibt es sogenannte Akteure, mit denen der Benutzer mittels *Java-Code* interagieren kann.

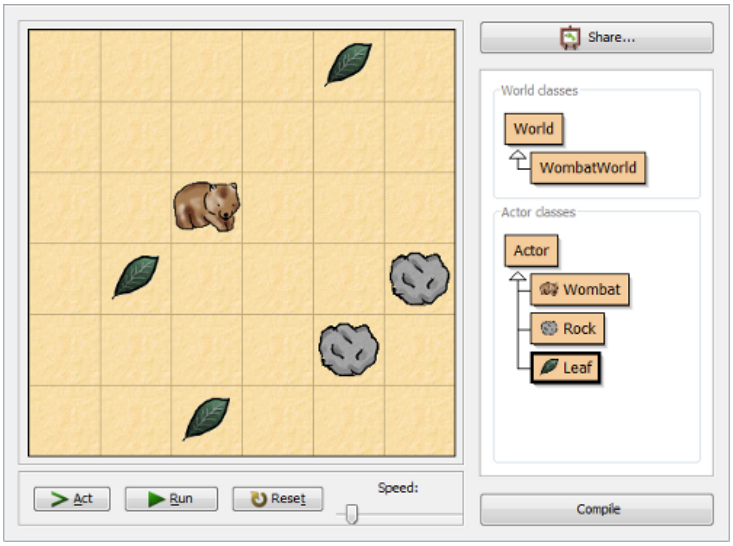


Abbildung 2.11: Das Interface des Programms *Greenfoot*, so wie es auf der offiziellen Webseite `https://www.greenfoot.org` vorgestellt wird.

In Abbildung 2.11 kann man das Interface des Programms *Greenfoot* sehen. Ein Programm in *Greenfoot* wird Szenario genannt. Ein solches Szenario besteht aus verschiedenen Akteuren und mindestens einer Klasse die die bestehende Klasse *Welt* erweitert. Diese eigenen Klassen können vom Benutzer komplett eigenständig programmiert werden.

Auf der offiziellen Webseite findet man eine vollständige Dokumentation, so wie Ressourcen für Lehrer, die *Greenfoot* im Unterricht benutzen wollen. Ebenfalls findet man dort eine große Anzahl an Szenarien die von Benutzern hochgeladen wurden. Das Programm wird sehr viel benutzt um damit kleinere Spiele zu erstellen, so findet man eine große Anzahl solcher auf der

[7] `https://www.greenfoot.org`

offiziellen Webseite.

Laut [Goetz, 2015] ist das Programm nur auf der Klasse 3GTG einsetzbar, da alle anderen Funktionen die in den Folgejahren gelernt werden sollen nicht mit *Greenfoot* umsetzbar sind. Ein Problem das ich hier sehe ist, dass auf einer 3GTG bereits das Programm *Unimozer* benutzt wird. So müssten hier beide Programme abwechselnd benutzt werden, was für den Schüler eine Unbequemheit darstellt und diese eventuell überfordern würde. Die Schüler lernen eine für sie unbekannte Programmiersprache, es wäre für mich sinnvoll dass diese nicht noch verschiedene Computerprogramme bedienen müssten.

Ebenfalls fehlen *Greenfoot* einige essentielle Features, die *Unimozer* unterstützt. So ist es in *Greenfoot* zwar möglich wenn ein Objekt einer Klasse erstellt wurde, deren Methoden zu visualisieren, jedoch gibt es keine Möglichkeit in *Greenfoot* UML Diagramme anzeigen zu lassen. Diese stehen jedoch auf dem offiziellen Rahmenlehrplan der Klasse und kommen ebenfalls in den Folgejahren wieder.

Ebenfalls bietet *Greenfoot* keine Möglichkeit Struktogramme von einzelnen Methoden zu visualisieren, im Gegensatz zu *Unimozer*. Auch diese sind fest im Rahmenlehrplan einer 3GTG verankert und müssen behandelt werden. Zusätzlich sind die Attribute eines erstellten Objekts nicht direkt sichtbar, so wie in *Unimozer*. Diese können zwar angezeigt werden, jedoch nur durch ein bestimmtes Menü, das für diesen Zweck vorgesehen ist.

Einen Vorteil den *Greenfoot* besitzt ist, dass es hier, laut der Dokumentation, ebenfalls Schnittstellen zu bestimmten Hardwaregeräten gibt. So kann *Greenfoot* unter anderem mit der *Microsoft Kinect* oder auch Gamepads benutzt werden. *Greenfoot* ist kostenlos und ist kompatibel mit allen gängigen Betriebssystemen.

Ebenfalls bemängelt [Goetz, 2015], dass es ähnlich wie bei *Java-Hamster*, viele verschiedenen Methoden gibt, die der Schüler kennen muss um eigene Projekte erstellen zu können. Diese sind eventuell überfordernd für einen Schüler und lenken ihn eventuell vom Wesentlichen ab. Ebenfalls muss der Benutzer darauf achten, wenn er nach einem Parameter vom Typ *String* gefragt wird, die Gänsefüßchen mit einzugeben, eine Arbeit die *Unimozer* automatisch übernimmt.

Ein großer Nachteil von *Greenfoot* ist, dass dieses nicht für den Einsatz in unserem Schul-

system angepasst werden kann, im Gegensatz zu dem Programm *Unimozer*, welches komplett *open-source* ist.

2.7.4 Moenagade

Das Programm *Moengade*[Fisch, 2017], ebenfalls von Robert Fisch, dem Entwickler von *Unimozer*, wurde speziell für den Einsatz auf der Klasse 3CI konzipiert. Das Programm soll den Schülern die Logik und Konzepte des Programmierens beibringen, ohne dass diese fähig sein müssen Quellcode zu schreiben. Das Prinzip des Programmes ist angelehnt an *Scratch*[8]. Den Schülern wird es ermöglicht sehr schnell visuelle Resultate zu erhalten indem sie mit einer Block-Programmiersprache einzelne Elemente steuern.

Der Einsatz dieses Programmes auf einer 3CI ist sinnvoll, da dies das erste Mal ist wo Schüler des *Enseignement Secondaire* mit Programmierung konfrontiert werden, im Gegensatz zu den Schülern des *Enseignement Secondaire Général*, die bereits durch *Scratch* einen Einstieg erhalten. Später wird dann der Übergang auf die Programme *Unimozer* und *Netbeans* gemacht.

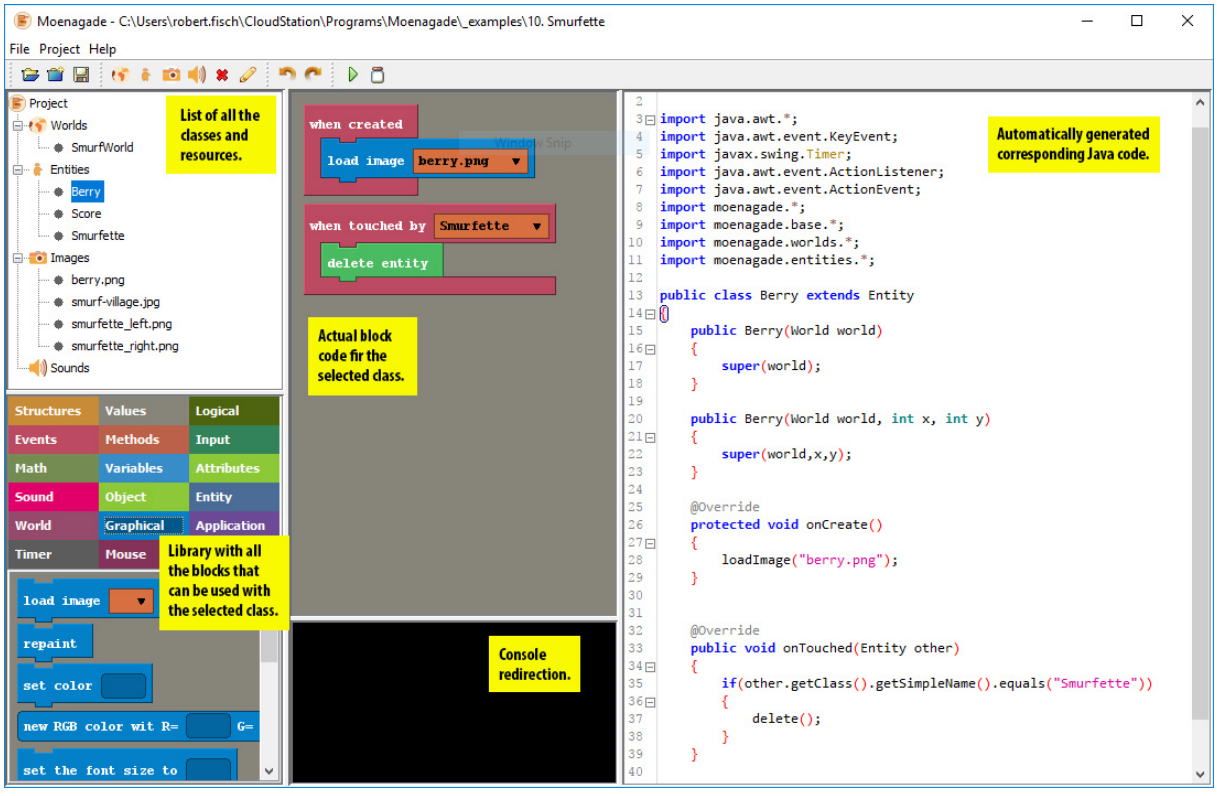


Abbildung 2.12: Das Interface des Programms *Moenagade*, so wie es auf der offiziellen Webseite [Fisch, 2017] vorgestellt wird.

[8] `https://scratch.mit.edu/`

Einen Vorteil den das Programm hat, ist dass der Schüler sein Programm durch das Benutzen von Blockprogrammierung erstellen kann. Dann wird automatisch der dazugehörige *Java-Code* erstellt. Dies ermöglicht einen schnellen Übergang von der Blockprogrammierung zum Programmieren mit Quelltext. Außerdem nimmt es dem Schüler die anfangs verursachten Probleme der Syntax weg und er kann sich ganz auf die Logik des Programmierens konzentrieren.

Für mich stellt das Programm einen sehr guten Ansatz dar um dem Schüler einen Einstieg in die Programmierung zu bieten, ohne dass dieser mit Quellcode konfrontiert wird. Dies ist ähnlich wie es im *Enseignement Secondaire Général* mit *Scratch* geschieht. Meiner Meinung nach wäre das Programm *Moenagade*, das eigentlich für den Einsatz auf einer 3CI konzipiert wurde, ebenfalls Sinnvoll für den Einsatz an Stelle von *Scratch*. So könnte einfacher der Übergang zum Programmieren mit Quellcode gewährleistet sein. Jedoch stell nach wie for für mich das Schuljahr 4GTG dar, welches ein Jahr Unterbrechung zwischen dem Lernen der Logik des Programmierens und des eigentlichen Programmierens darstellt.

Das Programm *Moenagade* ist dadurch, dass es unter *Java* geschrieben wurde, kompatibel mit allen gängigen Betriebssystemen und kann so problemlos von den Schülern auf deren Computer benutzt werden.

Kapitel 3

Methoden

3.1 Geplante Erweiterung von Unimozer

3.1.1 Motivation zur Erweiterung

Das bestehende Tool *Unimozer* soll durch eine Möglichkeit erweitert werden, die es den Schülern erlaubt direkt ein grafisches Feedback zu den von ihnen erstellten Klassen zu bekommen. Da das Erstellen eines grafischen Interfaces erst auf der Klasse 2GGE erlernt wird, soll hier eine andere Möglichkeit geschaffen werden, die es den Schülern erlaubt grafische Interfaces zu benutzen.

Ebenso soll die Erweiterung den Schülern ermöglichen ihre einzelnen erstellten Klassen und Methoden als Teil eines Größeren zu sehen. Oft beschwerten sich die Schüler, dass sie keinen Nutzen oder Kontext in den zur Verfügung stehenden Aufgaben finden. Dieses Problem soll hier gelöst werden.

Die direkte Visualisierung ihrer Klassen, soll die Motivation der Schüler gesteigert werden und die Wertschätzung des Fachs gefördert werden. Die Schüler sollen ihre Arbeit als Teil eines Ganzen erkennen und so die Notwendigkeit besser nachvollziehen können.

3.1.2 Geplante Umsetzung

Es soll ein System erstellt werden, das den Schülern visuelle Interaktion und Resultate schneller zugänglich macht. Hierzu wird eine Möglichkeit in *Unimozer* hinzugefügt, welche es ermöglicht sogenannte *interaktive Projekte* in *Unimozer* zu laden.

Der Unterschied von diesen Projekten zu normalen Projekten ist, dass es bereits bestehende Klassen im Hintergrund gibt, mit denen die Schüler interagieren können. Diese Klassen dienen dazu die Arbeit der Schüler zu visualisieren. Ein Beispiel hierfür wäre der *Space Simulator*, welcher eine abgewandelte Version des Java-Hamsters ist darstellt, welcher in Abschnitt 2.7.2 genauer beschrieben wird. Hier gibt es mehrere Level die existieren, inklusive sammelbarer Diamanten und Gegner die besiegt werden können, sowie einen Raumschiff mit dem der Schüler interagieren kann. Der Schüler soll dieses Raumschiff durch eine Controller-Klasse steuern können. Es existieren zwei verschiedene Typen von Projekten, die genauer in Abschnitt 3.2 beschrieben werden.

Der Schüler muss bei diesem Projekt weder die Klassen der Spielfläche, noch alle anderen Hintergrundklassen sehen. Damit der Schüler einfacher interagieren kann, wird hier eine Interface Klasse erstellt, die die Schnittstelle für den Schüler darstellt. So wird kein überflüssiger Quellcode, oder überflüssige Hilfsmethoden gezeigt. Diese könnten den Schüler verwirren, oder ihm sogar zu viel helfen um die Aufgabe zu lösen.

Die bestehenden Projekte werden direkt in *Unimozer* integriert und sind somit landesweit sofort nach der Integration verfügbar, über den *Webstart* von *Unimozer*. Sollten diese Projekte aktualisiert werden, so sind diese Änderungen ebenfalls sofort verfügbar.

So muss folglich in *Unimozer* ein System zum Erstellen, Speichern und Laden von interaktiven Projekten integriert werden. Diese Projekte sollen immer rückwärtskompatibel bleiben um bestehende Abspeicherungen von Projekten nicht negativ zu beeinträchtigen.

3.2 Geplante Aufgaben und Projekte

Zusätzlich zum Erstellen einer Methode zur Integration von interaktiven Projekten, werden auch letztere erstellt. Ebenfalls sollen dazugehörige Aufgaben erstellt werden.

Es werden zwei Arten von Projekten erstellt:

- Projekte wo der Schüler einen Controller implementieren muss
- Projekte wo der Schüler die Modellklasse erstellen muss

Bei beiden Typen existieren bereits Java Klassen im Hintergrund, welche größtenteils un-

sichtbar sind für den Schüler. Dies ist der Fall um den Schüler nicht durch komplexen Code zu verwirren, oder aber ihm zu helfen dadurch dass er bereits Beispielcode hat. Welche Elemente sichtbar oder unsichtbar für den Schüler sind wird in den folgenden Abschnitten erklärt.

In den folgenden Abschnitten werden beide Typen von Projekten genauer erklärt.

3.2.1 Projekte die einen Controller erfordern

Erklärungen und Anforderungen

Bei dieser Art von Projekten soll der Schüler eine Controller-Klasse entwerfen, die eine bestimmte bereits vorhandene Klasse steuern kann.

Ein Beispiel hierfür wäre zum Beispiel der Java-Hamster. Der Schüler sieht nur den Hamster den er steuern kann, alle anderen Hintergrundklassen die zur Anzeige benutzt werden sieht der Schüler nicht. Hinzu kommt, dass ebenfalls eine leere Controller-Klasse erstellt wird, die der Schüler vervollständigen muss.

Die Klasse mit der der Schüler interagieren kann, enthält eventuell Hilfsmethoden so wie Attribute die der Schüler nicht sehen soll. Um dieses Problem zu lösen wird eine Interface Klasse erstellt, die nur die Methoden enthält, die für den Schüler zugänglich sein sollen. Somit sieht der Schüler keine Hilfsmethoden der Klasse mit der er interagiert.

Die Controller-Klasse, die der Schüler vervollständigen soll, wird bereits für ihn erstellt. Diese enthält bereits ein Attribut vom Typ der Interface-Klasse, so wie einen Setter für dieses Attribut. Diese wurden für den Schüler erstellt, da sie existieren müssen, dass das Projekt ohne Probleme gestartet werden kann. Ebenfalls wird das Attribut das die Interface-Klasse enthält, beim Starten des Programmes automatisch initialisiert und zeigt auf das im Hintergrund erstellte Objekt.

Ein konkretes Beispiel

Hier wird der vorherige Abschnitt anhand eines konkreten Beispiels dargestellt, dem *Space Simulator*. Dessen genaue Funktionsweise, so wie die dazugehörigen Aufgaben werden im Kapitel 4.2.3 genauer beschrieben.

In diesem Projekt soll der Schüler ein Raumschiff durch verschiedenen Level steuern. Das Raumschiff selbst, so wie alle anderen Hintergrundklassen existieren bereits, aber deren Quellcode ist für den Schüler nicht zugänglich. Das Raumschiff besitzt bestimmte Methoden, die der Schüler in seiner Controller-Klasse benutzen kann, um das Raumschiff zu steuern.

3.2.2 Projekte die ein Modell erfordern

Erklärungen und Anforderungen

Bei dieser Art von Projekten, soll der Schüler eine Modell-Klasse vervollständigen, die das Modell im MVC-Modell darstellt. Es existiert bereits ein grafisches Interface zum Visualisieren dieser Klasse. Alle Hilfsklassen, zur Anzeige, sind für den Schüler unsichtbar. Er sieht nur die Klasse die er vervollständigen soll.

Um diese Klasse zu testen, hat der Schüler noch immer die Möglichkeit diese manuell zu kompilieren und die Methoden einzeln aufzurufen und auszuprobieren, so wie sie es bereits durch den Rest des Kurses gewohnt sind. Durch das grafische Interface kommt jetzt allerdings die Möglichkeit das Projekt zu starten und die Klasse dann mittels eines grafischen Interfaces zu probieren. Das Interface bietet für den Schüler eine einfache Schnittstelle um mit der von ihnen erstellten Klasse zu interagieren.

Durch die CNPI wurden bereits im Jahre 2017 versuche unternommen um in diese Richtung zu denken. So wurden den Schülern Projekte zur Verfügung gestellt, die bereits ein grafisches Interface enthalten und vom Schüler die Implementation einer bestimmten Klasse voraussetzen. Hier entstehen für mich allerdings einige Probleme:

1. Der Schüler soll dieses Projekt durch das Programm *Netbeans* fertigstellen, das sie bis dahin noch nicht benutzt haben. Dies könnte jedoch als Übergang auf das Folgejahr dienen, wo dieses benutzt wird.

2. Die Projekte sind für das Ende der 3GTG gedacht und werden bislang nicht früher von den Schülern benutzt.

3. Beim Öffnen des Projekts wird der Schüler direkt mit Fehlermeldungen konfrontiert. Das Projekt enthält direkt Fehler, da die Klasse und Methoden die der Schüler erstellen soll noch nicht existieren. Dies könnte meiner Meinung nach frustrierend für die Schüler sein. Die Projekte können dadurch ebenfalls nicht sofort ausgeführt werden, sondern erst wenn

der Schüler alle benötigten Methoden implementiert hat. Es würde jedoch Sinn machen wenn der Schüler diese einzelnen Methoden direkt beim Erstellen testen könnte.

Trotz dieser Kritiken finde ich, dass diese Projekte eine gute Initiative sind und einen Schritt in die richtige Richtung darstellen. Durch meine Implementierung der interaktiven Projekte versuche ich diese Probleme zu umgehen:

1. Dadurch, dass meine Implementation in *Unimozer* vorgenommen wird, können die Schüler das ihnen vertraute Programm weiterhin benutzen.
2. Die Projekte können eventuell schon früher benutzt werden. So versuche ich Aufgaben für die unterschiedlichsten Teile des Programms zu erstellen.
3. Die vom Schüler zu erstellenden Methoden existieren nicht von Anfang an, womit das Problem entsteht, dass diese nicht direkt in den Hilfsklassen benutzt werden können ohne dass Fehlermeldungen entstehen. Diese soll der Schüler jedoch nicht sehen, da sie in Klassen entstehen die für den Schüler unverständlich sind. So sollen die Hilfsklassen für den Schüler unsichtbar sein, wie bei den Controller-basierten Projekten. Außerdem soll Reflexion benutzt werden um die Methoden der Schüler zu benutzen. Dann werden dem Schüler auf die Situation angepasste Meldungen angezeigt.

Ein weiteres Problem das hierdurch entsteht ist, dass es sein kann, dass Methoden, die zum einwandfreien Funktionieren des Projektes gebraucht werden, noch nicht vom Schüler erstellt wurden. Oder der Schüler hat diese eventuell anders genannt, oder sich im Typ geirrt.

Um dieses Problem zu lösen werden diese Fehlermeldungen abgefangen und dem Schüler auf verständliche Art und Weise dargelegt. Dies passiert zum Beispiel in der Form: *Die Methode void xy() wurde in deiner Klasse noch nicht erstellt!.*

Ein konkretes Beispiel

Hier wird der vorherige Abschnitt anhand eines konkreten Beispiels dargestellt. Ein Beispiel ist ein Geldautomat, der angelehnt ist an eine bereits bestehende Aufgabe. Das grafische Interface zum Interagieren mit dem Geldautomaten existiert bereits, der Schüler muss allerdings die Klasse vervollständigen die das Konto des Benutzers darstellt, inklusive der Methoden um Geld abzuheben und einzuzahlen, so wie um den PIN-Code des Benutzers zu überprüfen. Eine genaue Beschreibung dieses Projekts wird in Abschnitt 4.2.1 gegeben.

Kapitel 4

Praktische Umsetzung

In diesem Kapitel wird die Erweiterung des Programmes *Unimozer* durch die im vorherigen Kapitel dargestellten Funktionen erklärt. In Abschnitt 4.1 gehe ich genauer auf die Implementierung ein. In Abschnitt 4.2 gehe ich dann auf die erstellten interaktiven Projekte ein.

4.1 Implementation in Unimozer



Abbildung 4.1: Dies ist ein QR-Code, der zu der GitHub Seite des Projektes *Unimozer* führt.

Unimozer ist ein *open-source* Projekt, das von Robert Fisch gestartet wurde und auch gewartet wird. Der Quell-Code dieses Programms ist offen zugänglich und kann auf `https://github.com/fesch/Unimozer` gefunden werden. Der QR-Code in Abbildung 4.1 führt zu dieser Seite. Das Projekt wird regelmäßig von verschiedenen Lehrern aktualisiert. Eine genauere Beschreibung von *Unimozer* ist in abschnitt 2.5 zu finden.

Die in dieser Arbeit hinzugefügten Funktionen wurden in einer separaten Abzweigung (engl. branch) des originalen Projektes erstellt. Diese wurden dann später in den Hauptzweig eingefügt. Somit blieb *Unimozer* bis zur kompletten Fertigstellung unberührt, ohne dass auf die Vorteile des Benutzens eines Git-Repositories verzichtet wird. Einmal fertiggestellt, konnte so leicht die veränderte Version von *Unimozer* in Betrieb genommen werden.

4.1.1 Erstellte Klassen

Die meisten neuen Klassen, die zu dem bestehenden Programm *Unimozer* hinzugefügt werden, wurden in ein Paket mit dem Namen *lu.fisch.unimozer.interactiveproject* hinzugefügt. Hiermit bleiben diese gruppiert und bleiben einfacher zu erkennen. Die Dialoge wurden zu einem bereits für diesen Zweck vorgesehenem Paket *lu.fisch.unimozer.dialogs* hinzugefügt.

Zusätzlich zu den erstellten Klassen, mussten allerdings einige Änderungen im bereits bestehenden Codes vorgenommen werden, was sich für eine saubere Einbindung nicht vermeiden lies. Diese können im *GitHub* Projekt von *Unimozer* eingesehen werden.

projects.xml

Dies ist eine XML-Datei, welche alle Informationen über die in *Unimozer* integrierten interaktive Projekte enthält. Jedes neue interaktive Projekt muss in dieser Datei registriert werden. Beim Öffnen eines neuen interaktiven Projekts, werden die nötigen Informationen aus dieser Datei geladen.
Die Struktur dieser XML-Datei ist exakt festgelegt und muss eingehalten werden, um ein Funktionieren eines Projekts zu gewährleisten. Ein Beispiel für diese Datei kann im Anhang A eingesehen werden. Hier folgt eine Erklärung der einzelnen Elemente eines Eintrags:

- **<type>:** Enthält den Typen des Projekts. Hier muss der Wert *model-based* oder *controller-based* enthalten sein, je nach Typ des zu ladenden Projekts.
- **<name>:** Enthält den Namen des Projekts.
- **<student-class>:** Enthält den Namen der Klasse die der Schüler editieren soll. Diese Klasse wird dem Schüler als einzige editierbare Klasse bereitgestellt. Alle anderen Klassen, außer der Interface-Klasse sind für den Schüler unsichtbar.
- **<path>:** Enthält den Pfad zu dem Projekt. Dieser wird vom *Root* von *Unimozer* aus angegeben.
- **<main>:** Enthält den Namen der Main-Klasse des Projekts.
- **<package>:** Enthält den Namen des Pakets in dem sich die Klassen befinden.
- **<files>:** Enthält alle Klassen die zum Projekt gehören. Diese werden unter der Form **<file>** KlassenName **</file>** eingefügt.

InteractiveProjekt

Dies ist die Hauptklasse, die alle wichtigen Attribute und Methoden für ein interaktives Projekt enthält. Die wichtigsten Methoden dieser Klasse sind:

- **Konstruktor:** Dies ist die Methode die das Projekt erstellt. Sie bekommt als Parameter den Namen des Projekts so wie eine Referenz zu einem Objekt mit dem Typ *Diagram* des bereits bestehenden Teils von *Unimozer*.
- **loadFromXML:** Diese Methode benutzt *XPath* [Clark et al., 1999] um die nötigen Informationen aus der Datei **projects.xml** zu laden. Die Methode nimmt einen booleschen Parameter entgegen, der angibt ob ein Projekt neu geladen wird, oder ein bestehendes gespeichertes Projekt geöffnet wird. Hiervon hängt ab, ob die Datei die der Schüler erstellen soll mitgeladen wird oder nicht. Der Name des zu ladenden Projekts befindet sich in dem Attribut **name** der Klasse. Beim Laden achtet die Methode ebenfalls darauf, welche Klassen im UML-Diagramm angezeigt werden sollen und welche Klassen der Schüler bearbeiten kann. Diese Methode unterscheidet ebenfalls zwischen fest eingebauten interaktiven Projekten, oder von anderen Lehrern erstellten Projekten. Ist ein Projekt fest in *Unimozer* eingebaut, so werden die Hintergrundklassen alle aus den in *Unimozer* mitgelieferten Textdateien geladen. Wird ein interaktives Projekt geladen das nicht fest in *Unimozer* eingebaut ist, so werden die Hilfsklassen alle aus dem Projektordner mitgeladen.
- **runProject:** Diese Methode ist für das Ausführen der interaktiven Projekte zuständig. Sie ist angelehnt an die bereits bestehende Methode *run* der bereits existierenden Klasse *Diagram*. Zuerst werden alle Klassen kompiliert und dann wird ein Objekt aus der in der Datei *projects.xml* angegebenen Klasse erstellt. Dann werden über Reflexion die wichtigsten Informationen aus diesem Objekt extrahiert, unter anderem das Objekt mit dem der Schüler interagieren soll.
- **save:** Dies ist eine Methode die hilft interaktive Projekte zu speichern. Da das Laden und Speichern komplizierter ist, wird dies noch in Abschnitt 4.1.2 genauer erklärt.

MyInteractiveObject

Diese Klasse ist eine Kind-Klasse der bereits in *Unimozer* enthaltenen Klasse *MyObject*. Letztere stellt ein konkretes Objekt grafisch dar, das aus einer Klasse erstellt wird. Diese Objekte werden in *Unimozer* im unteren Bereich, dargestellt durch die Klasse *Objectizer*, angezeigt. Dies kann in Abbildung 4.2 gesehen werden. Die roten Objekte sind die ursprünglichen Objekte wie sie

normalerweise in *Unimozer* dargestellt werden. Die Klasse *MyInteractiveProject* stellt eines der grünen Objekte dar. Dies sind die Objekte mit denen der Schüler interagieren können soll, die sogenannten Interface-Objekte. Hier werden beispielsweise die Attribute nicht angezeigt, weil diese den Schüler nicht ablenken sollen. Der Schüler sieht nur die für ihn bestimmten öffentlichen Methoden, die seine Klassen ausführen können.

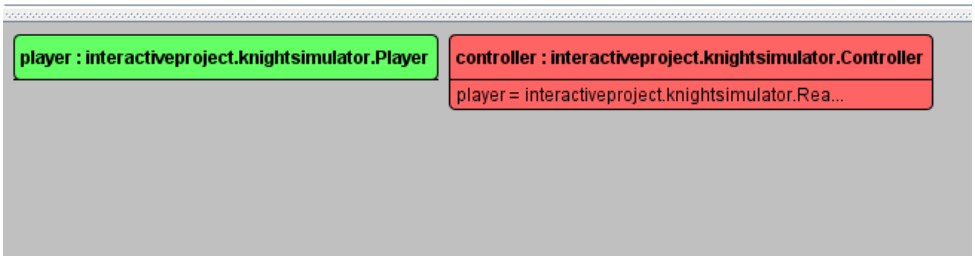


Abbildung 4.2: Hier wird der Bereich namens Objectizer von *Unimozer* gezeigt. Die Objekte der kompilierten Klassen werden in rot dargestellt (MyObject). Das grüne Objekt stellt ein Objekt der Interface-Klasse dar.

4.1.2 Speichern und Laden eines Projekts

Zum Speichern und Laden von interaktiven Projekten mussten bestehende Methoden der bereits bestehenden Klasse von *Unimozer* namens *Diagram* verändert werden. Hier musste jetzt der Fall unterschieden werden ob es sich um ein interaktives Projekt handelt oder nicht. Deshalb wurde ein Attribut des Typs *InteractiveProject* erstellt, das dazu dient diese Fälle zu unterscheiden. Hat dieses Attribut den Wert *null*, so handelt es sich nicht um ein interaktives Projekt, andernfalls handelt es sich um ein solches. Diese Art und Weise der Umsetzung wurde gewählt, da der bestehende Teil von *Unimozer* nicht beeinträchtigt werden durfte.

Wenn jetzt ein Projekt gespeichert wird, muss ebenfalls irgendwie unterschieden werden um welche Art von Projekt es sich handelt, ohne bereits bestehende gespeicherte Projekte zu stören. Da *Unimozer* auf vielen Klassen landesweit benutzt wird, müssen bestehende Projekte weiterhin problemlos geladen werden können. Sonst würden Probleme beim alltäglichen Einsatz der Software entstehen, was unbedingt vermieden werden musste. Es existiert bereits eine Datei *unimozer.pck* im Projektordner eines existierenden gespeicherten Projekts. Diese enthält mehrere Informationen über das Projekt. Anfangs wollte ich diese Datei benutzen um die Information zu speichern ob es sich um ein interaktives Projekt handelt oder nicht. Dies war allerdings nicht möglich. Die Datei hat mehrere Informationen hintereinander in einer festgelegten Reihenfolge. Ebenfalls kann nicht einfach eine Information dahinter gesetzt werden, da sie keine festgelegte

Größe hat und am Ende der Datei alle Klassen aufgezählt werden, ohne festgelegte Anzahl. Deshalb wurde, um den Unterschied zwischen normalen und interaktiven Projekten machen zu können, festgelegt, dass bei interaktiven Projekten zusätzlich eine Datei namens *interactiveproject.pck* vorhanden sein wird. Diese wird beim Speichern eines interaktiven Projekts angelegt und beim Laden wird überprüft ob diese Datei existiert. Der genaue Inhalt dieser Datei wird später erklärt.

Anfangs habe ich nur ein System vorgesehen um interaktive Projekte zu benutzen die im Quellcode von *Unimozer* fest eingebaut sind. Der Vorteil hiervon wäre, dass die Projekte direkt landesweit in *Unimozer* verfügbar sind, ohne dass zusätzliche Dateien heruntergeladen und eingebunden werden müssen. Später wurde dann allerdings die Überlegung angestellt ob es nicht Sinn machen würde, dass Lehrer ihre eigenen interaktiven Projekte erstellen können, ohne diese in den Quellcode einbauen zu müssen. Hierzu sollte dann ein System erstellt werden das dies ermöglicht. Hier gibt es beim Speichern oder Laden also zwei Fälle:

- **Fest eingebaute Projekte:** Diese sind in den Quellcode von *Unimozer* eingebaut und über das Menü zugänglich, das das Auswählen solcher Projekte ermöglicht. In diesem Fall dürfen die bestehenden Klassen beim Speichern nicht mitgespeichert werden, um so Aktualisierungen oder Fehlerbehebungen der Projekte nicht zu blockieren. Da die Hintergrundklassen in *Unimozer* eingebunden sind, wird ein Speichern nicht benötigt. Außerdem sind diese Klassen für den Schüler statisch, beziehungsweise werden nicht durch diesen verändert. Nur die Klassen die der Schüler selbst editiert werden in diesem Fall auf der Festplatte gespeichert.
- **Eigene erstellte Projekte:** Hier befinden sich die Hintergrundklassen nicht um Quellcode von *Unimozer*. Deshalb müssen diese zusammen mit dem Projekt auf der Festplatte gespeichert werden.

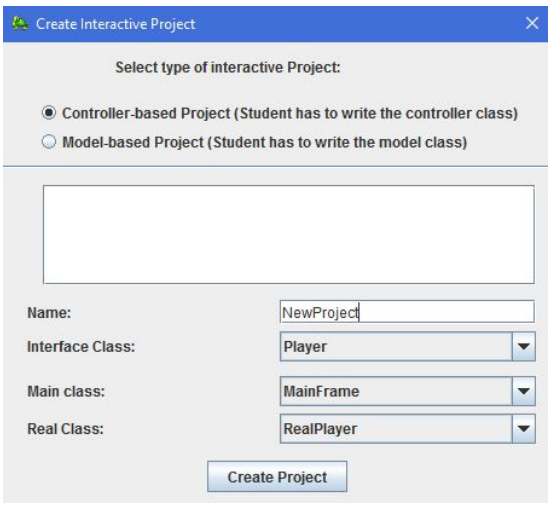
Im letzteren Fall musste ebenfalls bedacht werden, dass die benötigten Informationen über die verschiedenen Klassen nicht in der in *Unimozer* enthaltenen Datei *projects.xml* enthalten sein können. So musste eine Methode gefunden werden diese Informationen mit den Projektdateien im Projektordner zu speichern. Ebenfalls sollte das laden eines interaktiven Projekts nicht komplett neu geschrieben werden. Deshalb wurde der Entschluss genommen die Datei *interactiveproject.pck* zu benutzen die mit den interaktiven Projekten gespeichert wird. Hier werden dann, exakt gleich wie in der Datei *projects.xml* die Informationen über das gespeicherte interaktive Projekt geschrieben. So kann die Methode *loadFromXML* der Klasse *InteractiveProject*,

beschrieben in Abschnitt 4.1.1, weiter benutzt werden. Sie musste nur aus einer anderen als der eingebauten XML-Datei lesen.

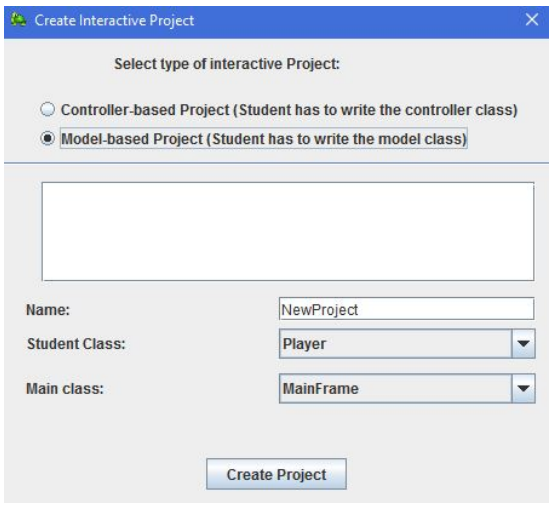
4.1.3 Erstellen neuer interaktiver Projekte

Wie schon in Abschnitt 4.1.2 beschrieben, wurde eine Funktion eingebaut, die es ermöglicht eigenen interaktive Projekte zu erstellen, anstatt nur die fest eingebauten benutzen zu können. Um dies zu ermöglichen muss der Ersteller alle benötigten Klassen in *Unimozer* erstellen, oder ein Projekt aus *Netbeans* importieren, das alle Klassen enthält. Diese Können dann verwendet werden um daraus ein interaktives Projekt zu erstellen.

Zu diesem Zweck wurde eine neue Klasse *CreateInteractiveProjectDialog* erstellt, die die Klasse *JDialog* erweitert. Durch diese Klasse wird dem Benutzer ein Dialogfeld präsentiert, das es ihm ermöglicht aus dem aktuell geladenen Projekt ein interaktives Projekt zu erstellen.



(a) In diesem Menü können Controller-basierte Projekte erstellt werden. Dazu müssen alle wichtigen Klassen ausgewählt werden.



(b) In diesem Menü können Modell-basierte Projekte erstellt werden. Man muss hier die Klasse auswählen die der Schüler vervollständigen muss.

Abbildung 4.3: Dies ist das Menü zum Erstellen von neuen interaktiven Projekten.

Das Dialogfeld ändert sein Aussehen, je nach Auswahl des Projekttypen. So kann der Benutzer interaktive Projekte erstellen die Controller-basiert sind und solche die Modell-basiert sind. Je nach Art des Projekts muss der Benutzer verschiedene Angaben machen. Für all diese Angaben durchsucht *Unimozer* alle Klassen des aktuellen Projekts.

Für beide Typen

Für beide Typen von Projekten muss der Benutzer folgende Angaben machen:

- **Name:** Der Name den er dem interaktiven Projekt geben möchte.
- **Main Class:** Hier muss der Benutzer angeben welche die Hauptklasse des Programms ist. Diese muss eine Erweiterung der Klasse *JFrame* sein. Es werden hier automatisch alle Klassen angegeben die darauf zutreffen.

Controller-basierte Projekte

Das Dialogfeld zum Erstellen eines Controller-basierten Projektes kann in Abbildung 4.3a gesehen werden. Folgende Angaben muss der Benutzer angeben:

- **Interface Class:** Der Name der Interface-Klasse durch die der Schüler mit dem Programm interagieren kann. Das Dialogfeld zeigt automatisch alle Klassen an, die ein *Interface* sind.
- **Real Class:** Die Implementation der Interface-Klasse. Die Klassen die in Frage kommen werden all angezeigt.

Bei beiden Angaben ist es möglich, dass nur eine einzige Klasse zur Auswahl steht, jedoch wurde die Möglichkeit offen gelassen dass mehrere dieser Klassen existieren.

Modell-basierte Projekte

Das Dialogfeld zum Erstellen eines Modell-basierten Projektes kann in Abbildung 4.3b gesehen werden. Folgende Angaben muss der Benutzer angeben:

- **Student Class:** Der Name der Klasse die der Schüler vervollständigen muss. Diese muss bereits existieren, der Schüler muss sie nur vervollständigen.

Nach der Auswahl aller benötigten Angaben kann ein Projekt erstellt werden. Danach kann dieses normal gespeichert werden und den Schülern zur Verfügung gestellt werden. Sollten einige der benötigten Klassen fehlen, oder kein Name angegeben werden wird dies dem Benutzer mitgeteilt und das Projekt kann nicht erstellt werden bevor diese Probleme behoben wurden.

4.2 Erstellte Projekte und Aufgaben

Hier werde ich kurz darlegen welche interaktiven Projekte und die dazugehörigen Aufgaben erstellt wurden. Die kompletten Angaben sind in Anhang B zu finden.

4.2.1 Bank Account

Dies ist ein Modell-basiertes Projekt, das erfordert, dass der Schüler das Projekt vervollständigt indem er die Modellklasse implementiert. Der bereits erstellte Teil des Projekts zeigt ein Interface, das an einen Geldautomaten angelehnt ist, zu sehen in Abbildung 4.4.

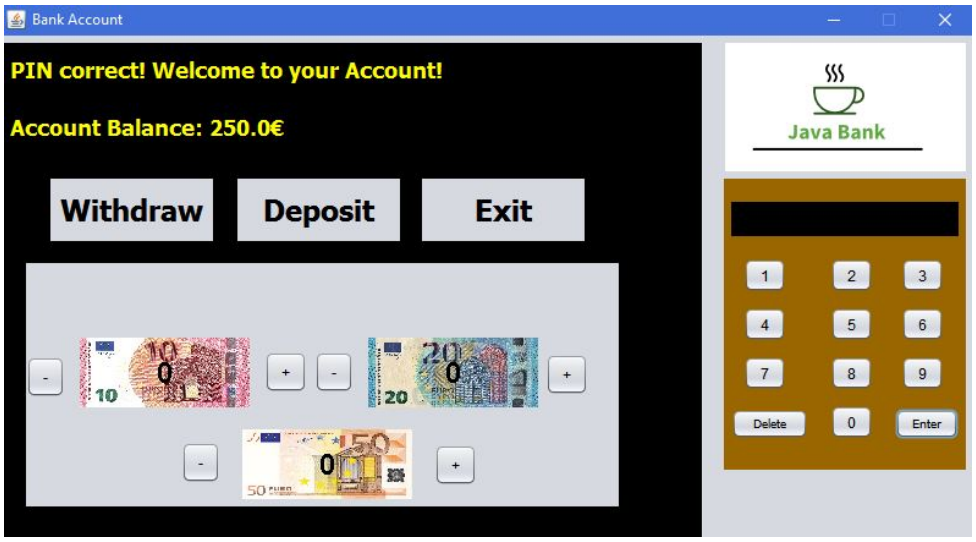


Abbildung 4.4: Das Interface der Aufgabe Bank Account. Es ist angelehnt an das Aussehen eines Geldautomaten. Rechts kann eine PIN-Nummer eingegeben werden, im linken Bereich kann der Benutzer Geld abheben oder zum Konto hinzufügen.

Diese Aufgabe ist für das erste Trimester einer 3GTG Klasse gedacht. Der Schüler muss hierzu ganze Zufallszahlen erstellen können, so wie die Grundlagen der Programmierung kennen. Ebenso muss der Schüler einfache If/Else Abzweigungen erstellen können. Der Schüler muss hier im Projekt eine Klasse namens *Account*, wie im UML Diagramm, in Abbildung 4.5 gezeigt, vervollständigen.

Anfangs ist der linke Teil des Interfaces versteckt und nur das Nummernfeld zur Eingabe des Passworts ist verfügbar. Das Ziel ist, dass der Benutzer zuerst seine PIN eingeben muss, damit der Rest des Interfaces freigeschaltet wird. Hierzu muss der Schüler als erstes ein Attribut namens **pin** erstellen, das eine ganze Zahl ist. Später wird noch eine Methode erstellt zum Überprüfen der PIN. Ebenfalls soll ein Attribut das eine Zeichenkette(String) ist namens **message** erstellt werden, so wie eine *Getter* Methode für dieses. (Dies könnte über Reflexion auch auf eine andere Art geschehen, in der auch direkt auf ein privates Attribut zugegriffen werden kann, aber dies würde nur wenig Sinn machen, da *Getter* und *Setter* Methoden für die Schüler auf dem Rahmenlehrplan stehen und eine korrektere Art darauf zuzugreifen darstellen.) Dieses Attribut stellt den Text dar, den der Geldautomat anzeigen wird.

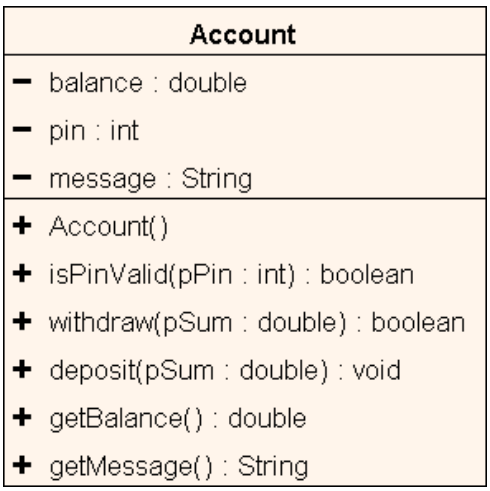


Abbildung 4.5: Dies ist das UML Diagramm der Klasse Account.

Die PIN des Benutzers soll anschließend im *Constructor* der Klasse *Account* mit einer Zufallszahl zwischen 1000 und 9999 initialisiert werden. Dies wurde so definiert, dass die Zahl vierstellig bleibt, ohne, dass die Schüler hier mit String-Operationen konfrontiert werden. Ebenfalls wurde zu diesem Zeitpunkt der Vergleich von zwei Strings noch nicht im Kurs behandelt. Ebenfalls muss hier das Attribut **message** initialisiert werden.

Der Benutzer kann hier eigentlich das Programm schon testen und nachsehen ob die in **message** gespeicherte Mitteilung auch angezeigt wird. Außerdem kann er im *Objectizer* von *Unimozer* bereits überprüfen ob die Zufallszahl initialisiert wird. In einem nächsten Schritt soll der Schüler die Methode **isPinValid** hinzufügen, die eine Zahl als Parameter nimmt und überprüft ob sie mit der PIN übereinstimmt. Hier muss der Schüler eine *If/Else* Abzweigung benutzen. Jetzt wird der Schüler angewiesen das Programm zu starten und eine PIN einzugeben. Hat er seine Klasse korrekt erstellt, so kann er sich im Geldautomaten mit der richtigen PIN anmelden. Sollte er bei den Parametern, Typen oder Namen der Methoden einen Fehler gemacht, so wird er über eine Meldung darauf hingewiesen. Einer der Vorteile dieser Projekte, wie bereits in Abschnitt 3.2.2 erwähnt, ist, dass er Schüler die Projekte von Anfang an starten kann, da diese durch die fehlenden Methoden keine Fehlermeldungen generieren. Ebenfalls kann der Schüler sein Programm etappenweise Ausprobieren und muss somit nicht bis zum Schluss hiermit warten.

In einem nächsten Schritt muss der Schüler die Methoden zum Abheben und Einzahlen von Geld auf dem Konto implementieren. Zunächst wird die Methode **deposit** erstellt, die es

ermöglicht eine als Parameter übergebene Anzahl an Geld auf das Konto zu setzen. Hier wird das Benutzen des Typs *double*, so wie das Benutzen von Parametern in Methoden wiederholt. Sobald diese Methode implementiert ist kann der Schüler sie bereits testen. Er kann mittels des gegebenen Interfaces Geldscheine hinzufügen und dann die Summe auf das Konto einzahlen durch einen Klick auf die Schaltfläche *Deposit*. Das Interface wird dann automatisch aktualisiert und zeigt auch direkt den neuen Kontostand an.

Als letztes soll dann eine Methode zum Abheben von Geldbeträgen implementiert werden. Diese nimmt einen Betrag entgegen und versucht dass den entsprechenden Betrag vom Konto abzuheben. Dies soll aber nur geschehen wenn genug Geld auf dem Konto verfügbar ist. Dies muss der Schüler in der Methode überprüfen. Sollte nicht genug Geld auf dem Konto sein, so wird dies durch ein verändern des Attributs **message** im Interface angezeigt. Hier muss der Schüler eine *If/Else* Abzweigung benutzen. Danach kann das Programm wiederum getestet werden und sollte jetzt voll funktionstüchtig sein.

Die Aufgabe ist vorgesehen für den Einsatz gegen Ende des ersten Trimesters, wo alle benötigten Elemente behandelt sein müssten. So werden in dieser Aufgabe folgende Elemente des Rahmenlehrplans der 3GTG wiederholt:

- Erstellen von Attributen und den dazugehörigen *Getter* und *Setter* Methoden.
- Erstellen eines *Constructors* für eine Klasse.
- Generieren von ganzen Zufallszahlen in einem vorgegebenen Intervall.
- Die Typen *boolean*, *int*, *double* und *String*.
- Einfache *If/Else* Abzweigungen.
- Parameter bei einer Methode benutzen.

4.2.2 Slot Machine

Bei diesem Projekt soll der Schüler einen einarmigen Banditen erstellen. Hier handelt es sich um ein Modell-basiertes Projekt bei dem das grafische Interface bereits existiert. Wie bei den anderen Aufgaben ist dieses Projekt von Anfang an ausführbar für den Schüler und generiert keine Fehlermeldungen die ein Kompilieren verhindern. Auch hier ist das Ziel, dass der Schüler das Programm Schritt für Schritt, und nicht erst am Ende, ausprobieren kann. Beim Starten

wird dem Schüler als Interface das Bild eines einarmigen Bandits gezeigt.

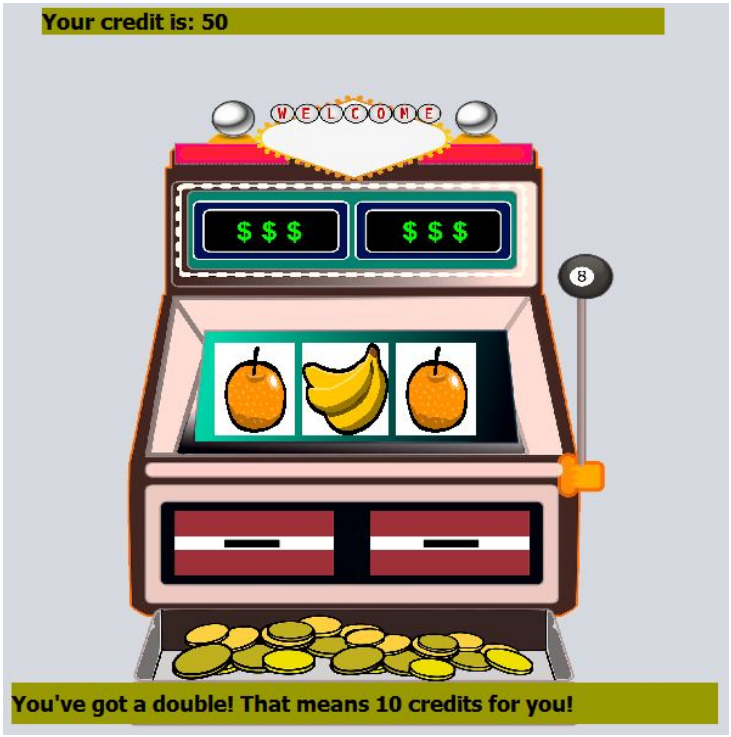


Abbildung 4.6: Das Interface der Aufgabe Slot Machine. Dem Schüler wird hier ein einarmiger Bandit präsentiert, den er benutzen kann um damit zu spielen. Durch ein Klicken auf den Arm des Banditen kann dieser gestartet werden.

Der Schüler muss bei dieser Aufgabe die Klasse die den einarmigen Banditen darstellt implementieren. Hierzu müssen die benötigten Attribute und Methoden erstellt werden, die zur Logik der Klasse beisteuern. Das genaue UML Diagramm der Klasse ist in Abbildung 4.7 zu sehen.

Der Schüler muss in einer ersten Phase die 3 Rollen des einarmigen Banditen als Attribute erstellen. Diese werden **reel1**, **reel2** und **reel3** genannt, aus dem Englischen Wort *Reel* das diese Rollen darstellt. Diese Attribute sind ganze Zahlen aus dem Intervall $[0; 5]$ um die verschiedenen Symbole darzustellen. Neben den drei Attributen muss der Schüler *Getter* Methoden für diese erstellen. Bereits jetzt wird der Schüler aufgefordert um sein Programm auszuprobieren. Er kann das Erstellte Objekt im *Objectizer* von *Unimozer* benutzen um den einzelnen Rollen des Bandits manuell Werte zu geben. Diese werden dann sofort im Interface durch ein bestimmtes Bild angezeigt. Durch diese frühen Erfolge soll der Schüler ermutigt werden. Er bekommt hier direktes Feedback zu den von ihnen erstellten Methoden. Im bereits bestehenden Teil des Programmes wird ebenfalls überprüft, ob der Schüler eine Zahl im richtigen Intervall zuordnet, ansonsten bekommt er eine maßgeschneiderte Fehlermeldung.

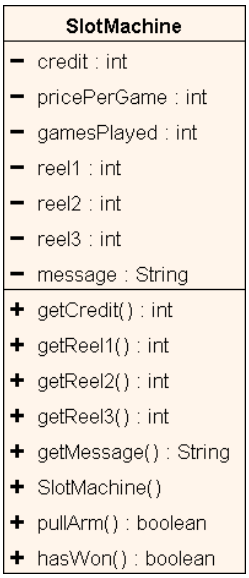


Abbildung 4.7: Das UML Diagramm der Klasse *SlotMachine*, die der Schüler vervollständigen soll.

In der zweiten Phase wird begonnen die Methode **pullArm** zu implementieren, die das eigentliche betätigen des Armes des Banditen darstellt. Dies ist eine boolesche Methode, denn sie soll angeben ob der Spieler über genügend Geld verfügt oder nicht, diese Logik wird allerdings später eingebaut. Zuerst soll der Schüler die Methode so schreiben, dass den drei Rollen eine Zufallszahl im Intervall $[0; 5]$ zugeordnet wird und immer der Wert **true** zurückgegeben wird. Dann wird der Schüler wieder aufgefordert das Programm zu starten. Durch die generierte Zufallszahl, die der Schüler einer Rolle zuordnet wird dann eine Animation gestartet, bei der die Rolle dann am Ende auf dem der Zahl zugeordneten Symbol stehen bleibt.

Anschließend wird der Schüler gebeten die Attribute **credit**, **gamesPlayed** und **pricePerGame** zu erstellen, welche ganze Zahlen sind. Hier repräsentiert **credit** das verbleibende Guthaben eines Spielers, welches auf 250 initialisiert wird. Zu diesem Attribut muss ebenfalls eine *Getter* Methode erstellt werden, damit das verbleibende Guthaben im Interface angezeigt werden kann. Das Attribut **gamesPlayed** zählt die Anzahl der gespielten Runden und soll auf Null initialisiert werden. Das Attribut **pricePerGame** stellt den Preis einer Runde des Spiels dar und soll auf 5 initialisiert werden. Zusätzlich soll noch das Attribut **message** erstellt werden, inklusive einer *Getter* Methode, welches ähnlich wie in der Aufgabe *Bank Account* eine Nachricht darstellt die dem Benutzer im Interface angezeigt wird.

Als nächstes soll die Logik der Methode **pullArm** vervollständigt werden. Der Schüler soll überprüfen ob der Spieler über genügend Guthaben verfügt, damit er eine Runde spielen kann. Ist dies nicht der Fall, so muss in das Attribut **message** eine entsprechende Nachricht gespeichert werden und die Methode gibt **false** zurück. Andernfalls werden den 3 Rollen Zufallszahlen zugeeignet, wie schon vorhin erstellt, der Preis eines Spiels vom Guthaben abgezogen und dir Anzahl der Spiele erhöht. Hierdurch werden einfache *If/Else* Abzweigungen wiederholt, so wie

der boolesche Type bei einer Methode. Der Schüler kann jetzt sein Programm wieder ausprobieren.

Als letztes soll dann eine Methode **hasWon** vom Typ boolean erstellt werden, die überprüft ob der Spieler gewonnen hat. Sie wird vom Interface automatisch aufgerufen nachdem eine Runde gespielt wurde. Es gibt zwei Fälle in denen der Schüler gewonnen hat, wenn zwei Rollen das gleiche Symbol zeigen und wenn 3 Rollen das gleiche Symbol zeigen. In beiden Fällen gewinnt der Spieler unterschiedliche Geldbeträge. Ebenfalls wird in diesen Fällen, oder beim Verlieren des Spiels, eine entsprechende Nachricht in das Attribut **message** gespeichert. Hier muss der Schüler schon verschachtelte *If/Else* Abzweigungen benutzen können, oder aber mit dem verknüpfen von Konditionen arbeiten. Anschließend ist der einarmige Bandit vollkommen funktionstüchtig und kann ausprobiert werden.

So werden bei dieser die folgenden Elemente des Rahmenlehrplans wiederholt:

- Erstellen von Attributen und den dazugehörigen *Getter* Methoden.
- Erstellen eines *Constructors* für eine Klasse.
- Generieren von ganzen Zufallszahlen in einem vorgegebenen Intervall.
- Die Typen *boolean*, *int* und *String*.
- Einfache *If/Else* Abzweigungen.
- Verschachtelte *If/Else* Abzweigungen oder verknüpfte Konditionen.

4.2.3 Space Simulator

Das interaktive Projekt *Space Simulator* wurde erstellt um das flexibelste Projekt zu sein. Es ist angelehnt an die in Abschnitt 2.6 beschriebenen Programme *CodeCombat* und *Java-Hamster*. Der Schüler soll hier in der Lage sein einen Spielcharakter durch ein Spielfeld zu steuern und verschiedene Aufgaben zu lösen. Es ist ein Controller-basiertes Projekt, da der Schüler ein Objekt durch einen Controller steuern muss. Hier wurde kein Hamster verwendet, sondern ein Raumschiff-Thema um zu versuchen die Schüler dadurch mehr zu motivieren. Es ist ein versuch durch *Gamification* die Schüler zu motivieren, indem beim Programmieren spielerische Elemente eingebracht werden.

Wie bei den Java-Hamster, versucht der Schüler hier ein bestimmtes Objekt über das Spielfeld navigieren, in diesem Fall ein Raumschiff. Dieses Raumschiff soll so gesteuert werden, dass es alle Diamanten im Spielfeld einsammelt und dabei nicht in die Asteroiden fliegt, da es sonst zerstört wird. Ebenfalls muss der Schüler versuchen alle Gegner zu umfliegen, oder zu besiegen. Ein Level ist erst geschafft, wenn der Schüler alle Diamanten gesammelt hat.

Für den Schüler ist bei diesem Controller-basierten Projekt nur ein Interface namens *Player* sichtbar. Hier sind nur Methoden eingetragen, die für den Schüler sichtbar sein sollen: die Methoden also, die für das Steuern des Raumschiffs wichtig sind. Im Hintergrund befindet sich noch eine Klasse namens *RealPlayer*, die weitere Attribute und Methoden enthält, um das Programm umzusetzen. Durch die Interface-Klasse *Player* bleiben diese aber für den Schüler nicht sichtbar.

Anfangs wurde erst eine Liste mit fest eingebauten Level erstellt, die an die ebenfalls erstellten Aufgaben, in Anhang B zu finden, angepasst sind. Die Level sind als einfache Textdateien in *Unimozer* eingebunden, in denen sie auf einfache Art und Weise gespeichert sind, durch das Benutzen verschiedener Symbole für die verschiedenen Objekte auf dem Spielfeld. Die Schüler sind in der Lage die Level durch eine einfache *Dropdown-Liste* auszuwählen. Dies ist in Abbildung 4.8 zu sehen.

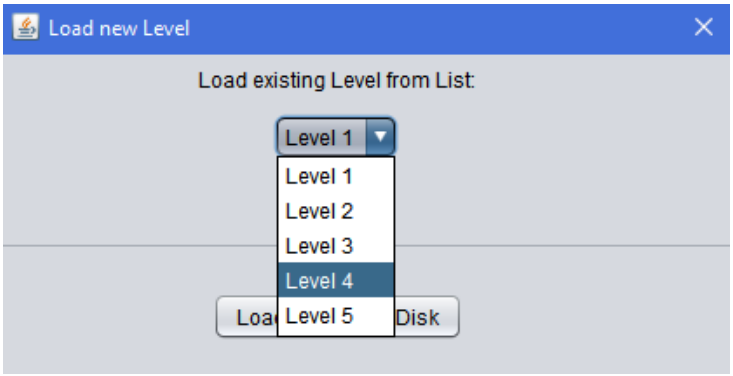


Abbildung 4.8: Der Schüler kann mittels der Liste eines der fest eingebauten Level auswählen.



Abbildung 4.9: Diese Toolbar ermöglicht es dem Benutzer Elemente auszuwählen um diese auf dem Spielfeld zu platzieren.

In einer zweiten Phase implementierte ich noch eine Methode um während des laufenden Programmes neue Level zu erstellen, oder Level zu bearbeiten. Dies erschien mir sinnvoll, da sonst andere Lehrer nur auf die eingebauten Level reduziert wären. Hierfür habe ich mich am Java-Hamster inspiriert. Dieser ermöglicht es dem Benutzer direkt Objekte

im Spielfeld zu platzieren, während das Programm läuft. Zu diesem Zweck habe ich eine Toolbar erstellt, zu sehen in Abbildung 4.9, die es dem Benutzer ermöglicht Elemente des Spiels auszuwählen und direkt im Spielfeld, durch ein Klicken, einzufügen. Man muss erst ein Objekt in dieser Toolbar auswählen, dann können diese platziert werden indem der Benutzer auf ein Feld innerhalb des Spielfeldes klickt.

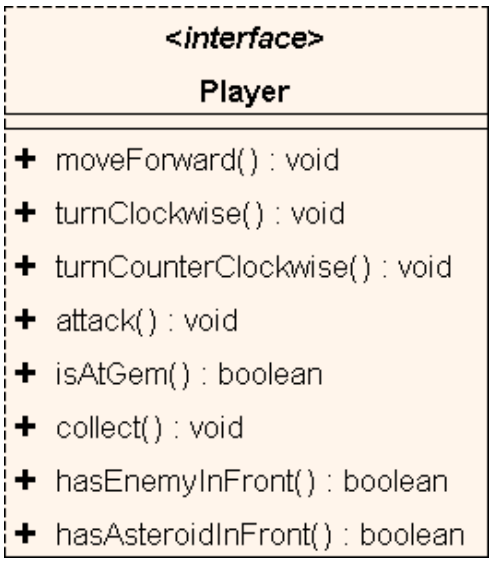


Abbildung 4.10: Hier ist das UML Diagramm der Klasse *Player* zu sehen.

Diese Projekt habe ich erstellt um mehr auf den Aspekt der *Gamification*, beschrieben in Abschnitt 2.3, einzugehen. Der Schüler soll durch dieses Projekt unter anderem direktes visuelles Feedback zu den von ihm implementierten Methoden erhalten. Außerdem erhoffe ich mir durch das einbringen von spielerischen Elementen, einer *Gamification*, eine Steigerung der Motivation der Schüler.

Das Raumschiff, das der Schüler steuern kann, verfügt über verschiedene Methoden mit denen der Schüler interagieren kann. Diese stellen keine komplette Liste dar und die Klasse kann problemlos mit weiteren erweitert werden, falls diese benötigt werden. Ein UML Diagramm der Klasse *Player* ist in Abbildung 4.10 zu sehen. Die erstellten Methoden sind die folgenden:

- **moveForward:** Diese Methode bewegt das Raumschiff ein Feld nach vorne. Kommt es dabei auf ein Feld mit einem Asteroiden oder einem Gegner, so explodiert es.
- **turnClockwise:** Dreht das Raumschiff im Uhrzeigersinn.
- **turnCounterClockwise:** Dreht das Raumschiff gegen den Uhrzeigersinn.
- **attack:** Attackiert einen Gegner, wenn dieser genau ein Feld vor dem Raumschiff ist.
- **isAtGem:** Gibt **true** zurück wenn das Raumschiff auf einem Diamanten ist, andernfalls gibt die Methode **false** zurück.
- **collect:** Sammelt einen Diamanten auf, wenn sich das Raumschiff darauf befindet.

- **hasEmenyInFront:** Gibt einen booleschen Wert zurück der aussagt ob sich ein Gegner auf dem Feld vor dem Raumschiff befindet.
- **hasAsteroidInFront:** Gibt einen booleschen Wert zurück der aussagt ob sich ein Asteroid auf dem Feld vor dem Raumschiff befindet.

Falls noch andere Lehrer Wünsche für Methoden für diese Klasse haben, kann die Klasse problemlos erweitert werden. So lange nur Methoden hinzukommen und keine entwendet werden, oder umbenannt werden, so ist Rückwärtskompatibilität gewährleistet. Die erstellten Level sind in der Schwierigkeit steigend und bringen nach einander neue Elemente ein. So beginnt der Schüler mit einem einfachen Level, zu sehen in Abbildung 4.11, wo er das Raumschiff nur einige Felder nach vorne bewegen muss um dann einen Edelstein einzusammeln. In einer ersten Aufgabe wird der Schüler aufgefordert dieses Level manuell zu lösen, ohne eine Methode zu implementieren. Der Schüler soll bei dem erstellten Objekt des Raumschiffs im *Objectizer* die Methoden manuell durch Klicken aufrufen um das Level zu lösen. Dies soll den Schülern einen einfachen Einstieg geben und ihm helfen die Funktionsweise der Methoden besser nachzuvollziehen.

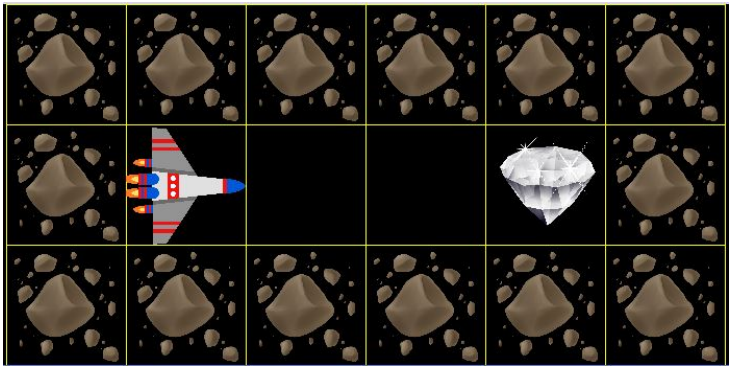


Abbildung 4.11: Das erste Level von *Space Simulator*

Im Gegensatz zum *Java-Hamster*, sind alle Methoden mit denen das Raumschiff gesteuert werden kann permanent für den Schüler sichtbar, ohne dass dieser in einer Dokumentation nachsehen muss. Ebenfalls können sie manuell vom Schüler ausgeführt werden, ohne dass diese in einer Methode benutzt werden müssen. Dies macht dem Schüler den Einstieg und das Ausprobieren einfacher.

Nachdem der Schüler das erste Level manuell gelöst hat, wird ihm anhand einer bestehenden Methode gezeigt, wie man das erste Level auch durch das implementieren einer Methode lösen

kann. Der Schüler wird aufgefordert diese Methode auszuprobieren. Dieses erlangte Wissen soll er nun einsetzen um ein zweites Level, zu sehen in Abbildung 4.12, zu lösen, in dem es gilt mehrere Diamanten zu sammeln.

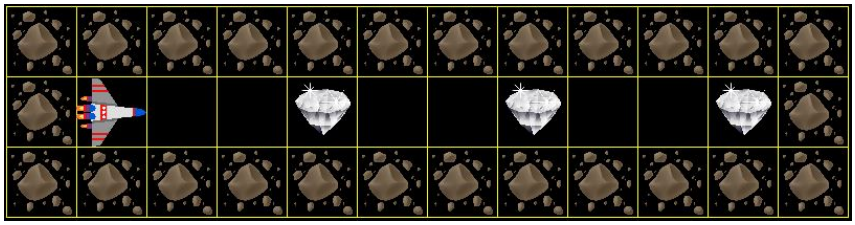


Abbildung 4.12: Das zweite Level von *Space Simulator*. Dieses kann mit einer Schleife gelöst werden.

In einem ersten Versuch wird der Schüler aufgefordert dieses Level auf eine sequentielle Art zu lösen, indem er alle benötigten Befehle nacheinander aufruft. In einem zweiten Versuch können dann Schüler die bereits den Umgang mit Schleifen gelernt haben, versuchen dieses Level anhand von einer Schleife zu lösen. Dies ist hier möglich, da es sich drei Mal um die gleiche Folge von Befehlen handelt.

In dem dritten Level wird dann das Drehen des Raumschiffs eingeführt. Um dieses Level zu lösen muss der Schüler in der Lage sein, das Raumschiff im und gegen den Uhrzeigersinn zu drehen.



Abbildung 4.13: Das dritte Level von *Space Simulator*. Hier muss das Raumschiff gedreht werden.

Auch dieses Level kann mittels einer Schleife gelöst werden. Durch das Benutzen der Me-

thode **hasAsteroidInFront** ist es sogar möglich das Level mit sehr wenigen Befehlen zu lösen, welches den Schülern als Herausforderung gegeben werden kann.

In einem vierten Level werden dann Gegner eingeführt. Hier muss der Schüler das Ufo besiegen um weiterfliegen zu können. Anhand der Methode **hasEnemyInFront** kann überprüft werden, ob sich ein Gegner genau vor dem Raumschiff befindet. Ist dies der Fall, so kann die Methode **attack** benutzt werden um das Ufo zu besiegen. Dieses Level ist in Abbildung 4.14 zu sehen.

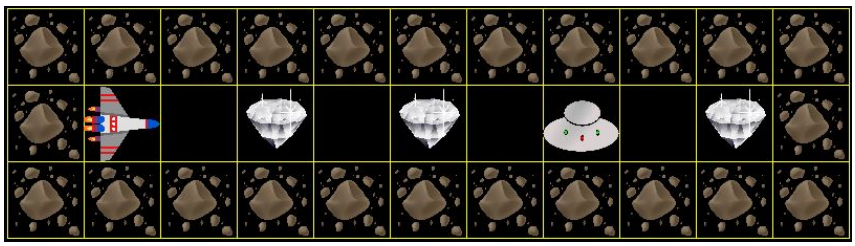


Abbildung 4.14: Das vierte Level von *Space Simulator*. Hier muss das Raumschiff den Gegner besiegen um fortfahren zu können.

Für alle Level gibt es mehrere Lösungen. So können diese alle durch das einfache Anreihen von Instruktionen gelöst werden, oder durch das Benutzen von Schleifen optimiert werden. So kann es für die Schüler eine Herausforderung sein die kürzest Lösung für ein Level zu finden.

4.2.4 TurtleBox

Das Projekt *TurtleBox* ist eine bereits bestehende Aufgabe für die 3GTG Klassen. Es ist angelehnt an das die Programmiersprache Logo, für die es mehrere Umsetzungen gibt. In diesem Projekt hat der Schüler eine Schildkröte zur Verfügung, die er über eine Fläche steuern muss. Das Ziel ist es hiermit Bilder zu malen. Da der Schüler eine Klasse schreibt, die die Schildkröte steuert ist es ein Controller-basiertes Projekt.

Bisher war es so, dass das bestehende Projekt, auf dem der Schüler sein Programm aufbaut, über einen Link[1] heruntergeladen werden musste. Jetzt findet sich das Projekt fest eingebaut in *Unimozer* wieder, als interaktives Projekt. Der Schüler muss es nicht mehr herunterladen, sondern kann es auf die gleiche Methode direkt auswählen, so wie bei anderen interaktiven Projekten. Die Controller Klasse wird ebenfalls automatisch erstellt. Ebenfalls wird beim Ausführen des Projekts automatisch ein Objekt der Klasse *TurtleBox* erstellt und mittels *Setter* dem Ob-

[1] `http://unimozer.fisch.lu/Files/TurtleBox.zip`

jekt der Klasse *Controller* hinzugefügt.

Zu diesem Projekt wurden auch einige Aufgaben erstellt in denen der Schüler bestimmte Formen und Figuren nachstellen muss. Diese befinden sich in Anhang B. Die Aufgaben erfordern ein logisches Überlegen, darüber wie diese Formen durch Fortbewegung der Schildkröte erstellt werden können. Die Aufgaben wurden zum Teil übernommen aus dem offiziellen *receuil d'exercices* der Klasse 3GTG.

Kapitel 5

Resultate

In diesem Kapitel werden die Ergebnisse dieses Projekts dargestellt. Ebenfalls wird versucht diese zu interpretieren.

Es wurden mehrere Versuche mit den erstellten interaktiven Projekten auf verschiedenen Schulklassen durchgeführt. Dies waren 3GTG Klassen des *Lycée des Arts et Métiers*. Die Versuche wurden unternommen um eine Motivationssteigerung bei den Schülern, im Vergleich zu den normalen Aufgaben und Projekten, festzustellen.

In Abschnitt 5.1 gehe ich auf die Meinungen der Schüler, so wie die Ergebnisse der Umfragen ein. Dann werde ich in Abschnitt 5.2 auf meine persönlichen Beobachtungen während der Versuche eingehen. In Abschnitt 5.3 werden die Meinungen anderer Lehrer dargestellt.

5.1 Meinung der Schüler

Die Meinung der Schüler wurde vor und nach dem Benutzen von interaktiven Projekten genommen um so einen Vergleich herstellen zu können. Es wurden einfache Fragen gestellt um feststellen zu können ob die Schüler durch die interaktiven Projekte motiviert wurden. Es wurden den Schülern jeweils verschiedene Aussagen vorgelegt zu denen es jeweils vier Antwortmöglichkeiten, basierend auf einer *Likert-Skala* [Likert, 1932], gab: *Ich stimme zu*, *Ich stimme teils zu*, *Ich stimme eher nicht zu*, *Ich stimme gar nicht zu*. Die Fragen wurden auf zwei 3GTG Klassen meiner Arbeitskollegen gestellt, da ich selbst keine solche Klasse in diesem Schuljahr habe. Es wurden jeweils die Projekte *Bank Account* und *Slot Machine* ausprobiert, so wie einige der erstellten Aufgaben für den *Space Simulator* für die schnelleren Schüler. Insgesamt nahmen 41 Schüler an dieser Umfrage teil.

Ich bin mir bewusst, dass dies keine repräsentative Umfrage darstellt, da die Größe der Stichprobe nicht ausreicht um eine eindeutige Aussage machen zu können. Allerdings kann die Umfrage dazu benutzt werden um ein Indikator zu sein auf welche Weise sich die interaktiven Projekte bei den Schülern auswirken. Ebenso ist die Einsatzdauer dieser Projekte zu kurz, so könnte eine plötzliche Motivationssteigerung nach Längerer Benutzung abflachen. Die ist eine Beobachtung, die ich selbst in [Osweiler, 2015] gemacht habe, wo die digitalen Abzeichen anfangs für Euphorie sorgten, aber die Begeisterung nach einiger Zeit abnahm. Eine klare Aussage kann erst nach einer Langzeitstudie gemacht werden.

Fragen vor den interaktiven Projekten:

1. Mir macht das Programmieren mit Java Spaß.
2. Ich verstehe wozu die von mir erstellten Klassen dienen.
3. Die Aufgaben finde ich interessant.
4. Ich bin im Unterricht motiviert.

Diese Fragen wurden den Schülern gestellt, bevor sie mit den interaktiven Projekten begonnen haben. Die Ergebnisse dieser Umfrage sind in Abbildung 5.1 zu sehen.

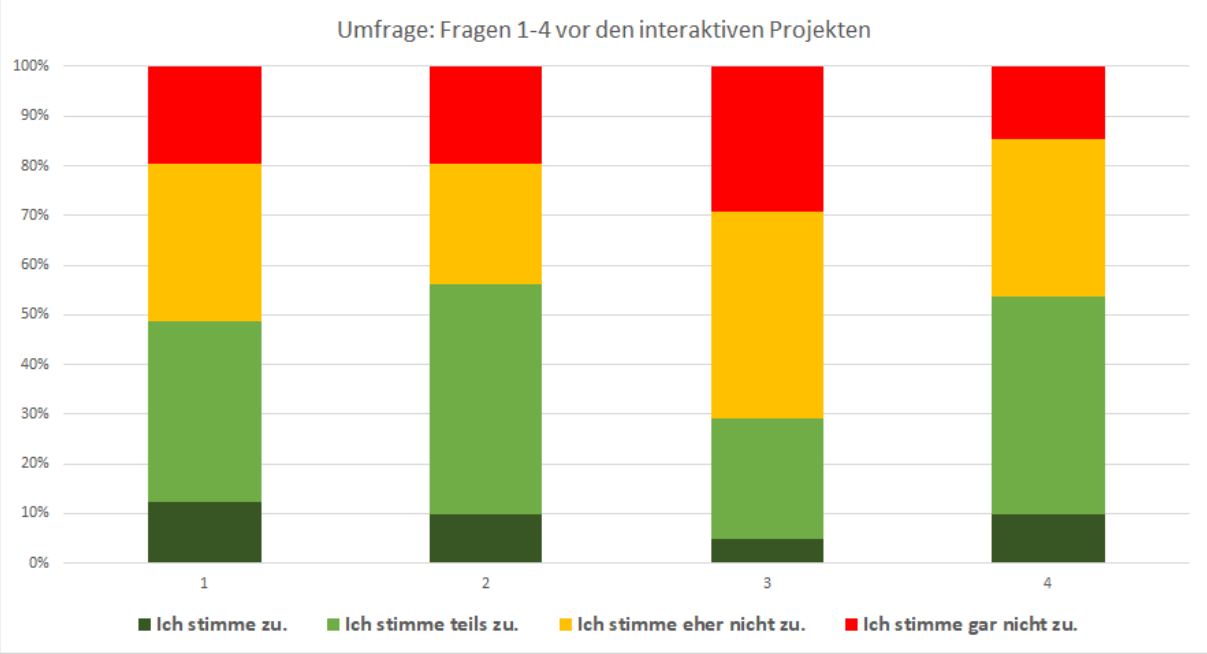


Abbildung 5.1: Ergebnisse der Umfrage vor den interaktiven Projekten.

Man sieht bei der ersten Frage deutlich, dass ungefähr die Hälfte der Schüler Spaß am Programmieren hat und die andere Hälfte nicht. Dies ist ein Phänomen, das sich mit persönlichen Beobachtungen deck, die ich selbst in den vergangenen Jahren gemacht habe. Das Fach Informatik hat die Tendenz bei den Schülern sehr stark zu polarisieren, so hat man oft zwei Lager an Schüler: Die einen die das Fach überhaupt nicht mögen, so wie die anderen, denen das Fach Spaß macht. Für meine Umfrage finde ich dies eine gute Mischung, da die beiden Lager ungefähr gleich stark vertreten sind, und so besser eine allgemeine Aussage gemacht werden kann.

Bei der zweiten Frage kann man beobachten, dass sich die Meinung der Schüler ähnlich aufteilt als bei der ersten Frage. So haben scheinbar die Hälfte der Schüler Probleme damit den Sinn hinter den von ihnen erstellten Klassen zu sehen. Interessant wäre hier zu wissen, ob dies die gleichen Schüler sind, die bei der ersten Frage angegeben haben, dass ihnen das Programmieren keinen Spaß macht. Dies kann jedoch im Nachhinein nicht herausgefunden werden, da die Umfrage komplett anonym gemacht wurde. Wenn dort aber eine Korrelation bestünde, so könnte dies erklären warum die Schüler das Programmieren nicht mögen.

Die dritte Frage bietet ebenfalls eine mögliche Erklärung warum manche Schüler keinen Spaß am Fach haben. Hier zeigt sich, dass die Mehrheit der Schüler der 3GTG die aktuellen Aufgaben uninteressant finden. Wenn man dies mit der ersten Frage vergleicht kann man feststellen, dass ebenfalls Schüler die Spaß am Programmieren haben die gestellten Aufgaben uninteressant finden. Dies könnte daran liegen, dass sie auf einer 3GTG noch keine grafischen Oberflächen erstellen, die ihre Programme anschaulicher machen.

Bei der vierten Frage zeigt sich, dass annähernd die Hälfte der Schüler aussagen, dass sie nicht motiviert im Programmierunterricht sind. Es ist schwer zu sagen, ob die an der Natur des Faches selbst liegt, oder aber eventuell eine allgemeine Einstellung in der Schule widerspiegelt. Ebenfalls könnte diese Aussage sehr von der Tagesform der Schüler abhängig sein. Eine mögliche Erklärung könnte das *Erwartung-mal-Wert-Modell* von [Atkinson, 1957] und [Wigfield and Eccles, 2000] bringen, die aussagt, dass die Motivation für ein bestimmtes Schulfach von dessen Wertschätzung abhängt. Wenn die Schüler keinen Spaß am Fach haben, wie in der ersten Frage ausgedrückt, kann sich das negativ auf die allgemeine Motivation und Lernbereitschaft im Unterricht auswirken.

Fragen nach den interaktiven Projekten:

1. Ich bin motivierter durch die interaktiven Projekte.
2. Ich verstehe besser wozu die von mir erstellten Klassen dienen.
3. Die interaktiven Projekte visualisieren meine Klasse besser.
4. Die interaktiven Projekte machen mehr Spaß als normale Aufgaben.

Diese Fragen wurden den Schülern gestellt, nachdem sie mit den interaktiven Projekten gearbeitet haben. Die Ergebnisse dieser Umfrage sind in Abbildung 5.2 zu sehen.

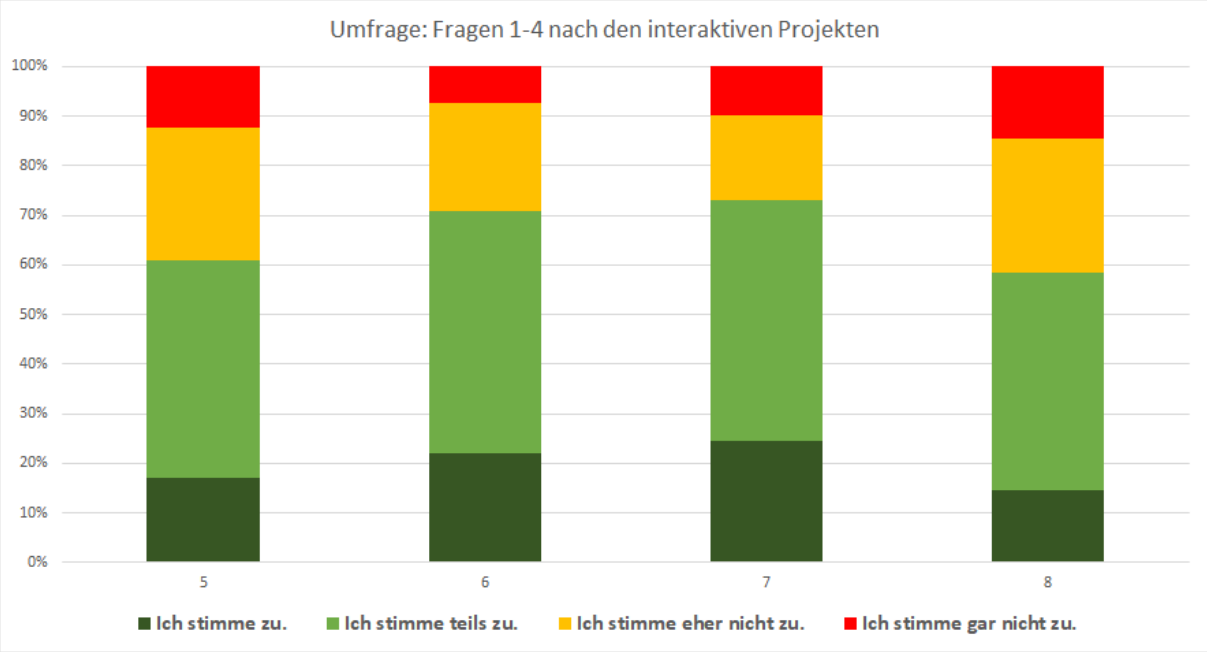


Abbildung 5.2: Ergebnisse der Umfrage nach den interaktiven Projekten.

Man kann sehen, dass über 61% der Schüler aussagen, dass sie durch die interaktiven Projekte motivierter sein. Dies könnte daran liegen, dass es eine neue Erfahrung für die Schüler ist und es deshalb eine positive Wirkung auf sie hat. Es könnte allerdings sein, dass dies nur der kurzfristige Effekt ist und längerfristig diese anfängliche Begeisterung abnimmt. Um festzustellen, ob hier eine länger Anhaltende Wirkung erzielt wurde, müssten die Projekte über einen längeren Zeitraum eingesetzt werden.

Laut den Ergebnissen der zweiten Frage, sagen 71% der Schüler ebenfalls aus, dass sie durch die interaktiven Projekte besser sehen wozu die von ihnen erstellten Klassen dienen. Dies könnte

daran liegen dass die Schüler durch die interaktiven Projekte leichter den Kontext sehen in den ihre Klassen passen. Ebenfalls könnte dies mit den Ergebnissen der dritten Frage zusammenhängen, wo ebenfalls 73% der Schüler aussagen, dass sie meinen, dass ihre Klasse durch die interaktiven Projekte besser visualisiert wird. Die Schüler sehen schneller einen Sinn hinter den gestellten Aufgaben.

Zusätzlich sagen 58,5% der Schüler aus, dass die Aufgaben der interaktiven Projekte ihnen mehr Spaß machen als die normalen Aufgaben des Kurses. Dies könnte ebenfalls daran liegen, dass es sich um etwas neues für sie handelt und sie deshalb kurzfristig motiviert werden, aber kein anhaltender Effekt messbar wäre, so wie ich es in [Osweiler, 2015] feststellen musste.

Mit den Ergebnissen zeigt sich eine Tendenz zur tatsächlichen Motivationssteigerung bei den Schülern. Diese könnte sich erklären lassen, durch eine Steigerung der Wertschätzung des Fachs, laut des *Erwartung-mal-Wert-Modell* von [Atkinson, 1957] und [Wigfield and Eccles, 2000]. Ebenso könnte dies ein Fall sein durch einen Versuch von *Gamification*. Jedoch sind diese Ergebnisse mit Vorsicht zu betrachten. Die Schüler könnten zum Beispiel durch meine eigene Anwesenheit beeinflusst worden sein, da sie mir eventuell einen Gefallen erweisen wollten, durch ein positives Feedback. Ebenso könnte diese Steigerung auch nur von kurzer Dauer sein, so dass auf längere Sicht die Motivation abnehmen würde. Außerdem war die Stichprobe mit nur 41 teilnehmenden Schülern zu klein um eine wissenschaftlich klare und haltbare Aussage machen zu können. Trotzdem bin ich mit den Ergebnissen zufrieden, da diese eine Tendenz zur Motivationssteigerung aufzeigen.

5.2 Persönliche Beobachtungen

Die gemessenen Werte der Umfragen decken sich auch mit denen von mir gemachten Beobachtungen. So konnte ich zum Beispiel sehen dass die Schüler sehr motiviert an diese neuen Aufgaben herangingen. Dies könnte jedoch auch der Fall sein, weil es ein Ausbruch aus dem normalen Schulalltag war, dadurch dass, durch meine Person, noch ein fremder Lehrer anwesend war und dass die Schüler wussten, dass sie dazu auserwählt wurden bei einer Studie mitzumachen.

Nach einer kurzen Einführung in die Funktionsweise der interaktiven Projekte, schienen die Schüler gut zurecht zu kommen und arbeiteten relativ selbstständig. Trotzdem schien es für sie Anfangs etwas ungewohnt zu sein, dass sie die Projekte ausführen können und diese dann mit

einem grafischen Interface testen zu können. Oft mussten die Schüler darauf hingewiesen werden.

Trotzdem waren die ersten Eindrücke der Schüler sehr positiv als sie zum ersten Mal das grafische Interface sahen das zu ihrem Programm gehörte. Man konnte sehen, dass sie Gefallen daran fanden, dass das Interface die Methoden benutzt die sie selbst geschrieben haben. Dies könnte daran liegen, dass das grafische Interface den Stellenwert den die Klassen der Schüler haben erhöht.

5.3 Meinung der Lehrer

Das Experiment war nicht nur für die Schüler eine neue Erfahrung, sondern auch für die teilnehmenden Lehrer. Diesen ist es nach persönlichen Aussagen einfach gefallen mit den interaktiven Projekten zu arbeiten, da bereits alles nötige vorhanden war. Absichtlich habe ich den Lehrern nicht erklärt wie die Projekte genau funktionieren, sondern bin ich nur kurz auf den Sinn der Projekte eingegangen. Dies habe ich getan um besser beobachten zu können wie ein Lehrer mit den Projekten umgeht, ohne dass er eine mündliche Erklärung erhalten hat. Dies tat ich bereits in Ausblick auf eine landesweite Benutzung, wo es auch nicht möglich ist jedem einzeln zu erklären wie die Projekte funktionieren. Dank einer genauen Erklärung, die sich am Anfang der Aufgaben befindet haben die Lehrer sich sehr schnell zurecht gefunden. Sie haben den Schülern eine kurze Zusammenfassung von dieser gemacht bevor diese loslegen konnten.

Die interaktiven Projekte konnten problemlos im Kurs benutzt werden, ohne dass der Lehrer viel Vorbereitung hineinstecken musste, da sowohl eine erklärenden Einführung, als auch die Projekte un deren dazugehörigen Aufgaben bereits vorhanden waren. Die bestehende Art und Weise Unterricht zu halten wird durch die interaktiven Projekte auf keine Weise gestört oder beeinflusst, sondern gliedern sich diese Problemlos in den Unterricht ein.

Auf einer 3CI wurden die interaktiven Projekte von einem der Lehrer sogar ohne die Angaben getestet. Dies wurde getan um zu überprüfen ob die Schüler allein durch die Logik des Interfaces und der Meldungen, die sie erhalten, auf die von ihnen zu erstellenden Methoden und Attribute selbstständig kommen können. Dies hat auch größtenteils geklappt und die Schüler sind ohne Angaben mit dem interaktiven Projekt zurecht gekommen.

Kapitel 6

Schlussfolgerung und Ausblick

Bislang gab es solche Projekte die fest in *Unimozer* eingebaut sind und wie sie in dieser Arbeit erstellt wurden noch nicht um im Unterricht benutzt zu werden. Meiner Meinung nach ist es extrem wichtig dem Schüler visuelles Feedback zu seiner Arbeit zu geben, so wie einen höheren Stellenwert der von ihm implementierten Klassen. Die hier in der Arbeit vorgestellten Projekte ermöglichen es den Schülern schneller zu Ergebnissen zu kommen. Dies motiviert die Schüler mehr als einzelne Klassen zu schreiben, die ihm nur textuell ein Feedback geben oder bei denen er den Kontext nicht versteht.

Dadurch, dass in dieser Arbeit nicht nur einzelne Projekte erstellt wurden, sondern vielmehr eine Plattform um solche umzusetzen, ist es möglich neue Projekte zu erstellen und diese in *Unimozer* zu integrieren. Ein Lehrer kann ganz einfach ein Projekt erstellen und anhand des integrierten Menüs das Projekt in ein interaktives Projekt für die Schüler umwandeln. Dies kann dann sofort in der Klasse benutzt werden. Ebenfalls kann er dieses Projekt dann weitergeben an die Verwalter von *Unimozer*, so dass dieses dann fest eingebaut werden kann und so landesweit verfügbar ist.

Bei den Versuchen mit den Schülern hat sich eine kleine Motivationssteigerung gezeigt, auch wenn diese Versuche nur mit wenig Schülern gemacht wurden. So bin ich doch positiv gestimmt was die Wirkung der interaktiven Projekte anbelangt. Es wird versucht den Schülern eine größere Wertschätzung ihrer eigenen Arbeit zu geben was meiner Meinung nach bei dem Testlauf auch gelungen ist. Ob dieser positive Effekt jedoch anhaltend ist muss sich noch zeigen.

In einer nächsten Phase soll die hier vorgestellte Arbeit in der CNPI vorgestellt werden um

eine landesweite Bereitstellung der Projekte und Aufgaben anzustreben. Das Ziel ist, dass diese fest in den Kursus integriert werden können. Ebenso werde ich die Funktionsweise der bereitgestellten Plattform erklären und vorstellen wie ein Lehrer ein solches Projekt erstellen kann, um so eine größere Basis an Lehrern, die solche Projekte benutzen und erstellen, zu schaffen.

So hoffe ich, dass die interaktiven Projekte, durch ihre Integration in *Unimozer* und dadurch landesweite Verfügbarkeit, in naher Zukunft auf nationaler Ebene benutzt werden. Ebenfalls erhoffe ich mir, dass noch weitere Projekte durch andere Lehrer, so wie durch mich selbst, folgen werden um so eine große Basis an Unterrichtsmaterial zu bilden.

Literaturverzeichnis

[Anderson, 1985] Anderson, J. R. (1985). *Cognitive psychology and its implications*. WH Freeman/Times Books/Henry Holt & Co.

[Atkinson, 1957] Atkinson, J. W. (1957). Motivational determinants of risk-taking behavior. *Psychological review*, 64(6p1):359.

[Biggs, 2011] Biggs, J. B. (2011). *Teaching for quality learning at university: What the student does*. McGraw-Hill Education (UK).

[Clark et al., 1999] Clark, J., DeRose, S., et al. (1999). Xml path language (xpath).

[Deci, 1971] Deci, E. L. (1971). Effects of externally mediated rewards on intrinsic motivation. *Journal of personality and Social Psychology*, 18(1):105.

[Deci, 1972] Deci, E. L. (1972). Intrinsic motivation, extrinsic reinforcement, and inequity. *Journal of personality and social psychology*, 22(1):113.

[Deterding et al., 2011] Deterding, S., Dixon, D., Khaled, R., and Nacke, L. (2011). From game design elements to gamefulness: defining gamification. In *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, pages 9–15. ACM.

[Entwistle, 1998] Entwistle, N. (1998). Motivation and approaches to learning: Motivating and conceptions of teaching. *Motivating students*, pages 15–23.

[Fallows and Ahmet, 1999] Fallows, S. J. and Ahmet, K. (1999). *Inspiring students: Case studies in motivating the learner*. Psychology Press.

[Fisch, 2009] Fisch, R. (2009). Unimozer. `http://unimozer.fisch.lu`. [Online; zugänglich am 12. Dezember 2017].

[Fisch, 2017] Fisch, R. (2017). Moengade. `http://moenagade.fisch.lu`. [Online; zugänglich am 12. Dezember 2017].

[Goetz, 2015] Goetz, J. (2015). Analyse de la valeur pédagogique ajoutée d'un outil de programmation java en vue d'une introduction éventuelle au programme des classes du régime technique de la division technique générale.

[Greasley, 1998] Greasley, K. (1998). *Does gender affect students' approaches to learning*. London: Kogan Page.

[Huotari and Hamari, 2011] Huotari, K. and Hamari, J. (2011). Gamification from the perspective of service marketing. In *Proc. CHI 2011 Workshop Gamification*.

[Jenkins, 2001] Jenkins, T. (2001). The motivation of students of programming. In *ACM SIGCSE Bulletin*, volume 33, pages 53–56. ACM.

[Kessler and Anderson, 1986] Kessler, C. M. and Anderson, J. R. (1986). Learning flow of control: Recursive and iterative procedures. *Human-Computer Interaction*, 2(2):135–166.

[Likert, 1932] Likert, R. (1932). A technique for the measurement of attitudes. *Archives of psychology*.

[Linn and Dalbey, 1989] Linn, M. C. and Dalbey, J. (1989). Cognitive consequences of programming instruction. *Studying the novice programmer*, pages 57–81.

[Nassi and Shneiderman, 1973] Nassi, I. and Shneiderman, B. (1973). Flowchart techniques for structured programming. *ACM Sigplan Notices*, 8(8):12–26.

[Osweiler, 2015] Osweiler, R. (2015). Empirische Machbarkeitsstudie über den erfolgreichen Einsatz von digitalen Abzeichen in einem Scratch Projekt.

[Perkins and Martin, 1986] Perkins, D. and Martin, F. (1986). Fragile knowledge and neglected strategies in novice programmers. In *first workshop on empirical studies of programmers on Empirical studies of programmers*, pages 213–229.

[White, 1959] White, R. W. (1959). Motivation reconsidered: the concept of competence. *Psychological review*, 66(5):297.

[Wigfield and Eccles, 2000] Wigfield, A. and Eccles, J. S. (2000). Expectancy–value theory of achievement motivation. *Contemporary educational psychology*, 25(1):68–81.

[Winslow, 1996] Winslow, L. E. (1996). Programming pedagogy - a psychological overview. *ACM Sigcse Bulletin*, 28(3):17–22.

Anhang A

Inhalt der Datei projects.xml

Hier wird ein Beispiel für den Inhalt der Datei *projects.xml* gezeigt, allerdings nur mit nur zwei Projekten, einem Modell-basierten und einem Controller-basierten Projekt.

```
<?xml version="1.0" encoding="UTF-8"?>
<projects>
        <project id="Space Simulator">
                <type>controller-based</type>
                <name>Space Simulator</name>
                <interface-class>Player</interface-class>
                <real-class>RealPlayer</real-class>
                <main>MainFrame</main>
                <interface-attribute>player</interface-attribute>
                <package>interactiveproject.spacesimulator</package>
                <files>
                        <file>Player</file>
                        <file>RealPlayer</file>
                        <file>Board</file>
                        <file>MainFrame</file>
                        <file>DrawPanel</file>
                        <file>LevelSelectDialog</file>
                        <file>NewLevelDialog</file>
                        <file>OpenLevelDialog</file>
                        <file type="student-class">Controller</file>
                </files>
        </project>
```

```
        <project id="Bank Account">
                <type>model-based</type>
                <name>Bank Account</name>
                <main>MainFrame</main>
                <package>interactiveproject.bankaccount</package>
                <files>
                        <file>MainFrame</file>
                        <file>bankNotePanel</file>
                        <file type="student-class">Account</file>
                </files>
        </project>
</projects>
```

ANHANG B

Erstellte Aufgaben

Exercices interactifs Java

Table des matières

1.1. Qu'est-ce qu'un projet interacftif ?....2
1.2. Ouvrir un nouveau projet interactif....2
1.3. Sauvegarder et ouvrir un projet interactif....2
1.4. Exécuter un projet interactif....2
1.5. Les deux types de projets....3
1.5.1. Projets basés sur un contrôleur....3
1.5.2. Projets basés sur un modèle....3
Exercice 1 : Bank Account....5
Exercice 2 : Slot Machine....7
Exercice 3 : Premiers pas avec le Space Simulator....10
Exercice 4 : Résoudre un niveau en implémentant un méthode (introduction)....11
Exercice 5 : Résoudre un niveau en implémentant une méthode (Level 2)....11
Exercice 6 : Solution à l'aide d'une boucle (Level 2)....12
Exercice 7 : Level 3....12
Exercice 8 : Ennemis (Level 4)....12
Exercice 9 : Dessiner un carré....14
Exercice 10 : Dessiner d'autres figures....15
Exercice 11 : Utiliser des méthodes pour dessiner des figures....15
Exercice 12 : Figures plus complexes....15

Explications générales

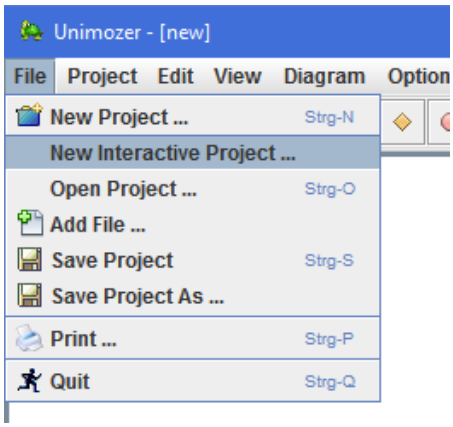
1.1. Qu'est-ce qu'un projet interacftif ?

Un projet interactif est un projet avec des classes prédéfinies. Ces projets peuvent être ouverts et édités dans le logiciel « Unimozer ».

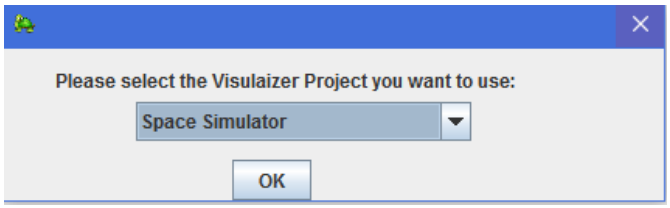
Il existe deux types de projets qui sont expliqués plus loin dans ce document.

1.2. Ouvrir un nouveau projet interactif

Pour ouvrir un nouveau projet interactif, il faut aller sur « File → New Interactive Project » dans le logiciel « Unimozer » .



Ensuite une fenêtre s'ouvre, qui permet de sélectionner le projet à ouvrir. Pour plus d'informations sur les différents projets, consultez les exercices dans ce document.

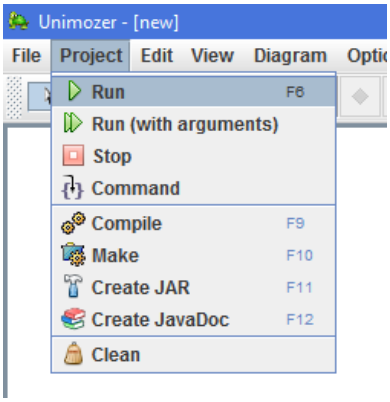


1.3. Sauvegarder et ouvrir un projet interactif

Les projets interactifs peuvent être sauvegardés et ouverts de la même manière que tous les autres projets crées dans « Unimozer ». **Attention** : Ces projets ne sont pas compatibles avec le logiciel « Netbeans ».

1.4. Exécuter un projet interactif

Pour exécuter un projet interactif, il faut le lancer par le menu « Project → Run ».

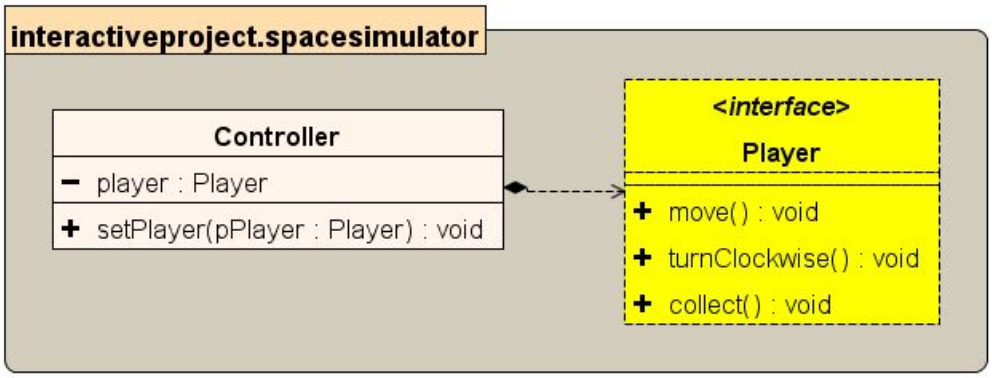


1.5. Les deux types de projets

Il existe deux types différents de projets, qui sont expliqués en détail par la suite.

1.5.1. Projets basés sur un contrôleur

Dans ce type de projet, l'élève doit compléter une classe « Controller ». Celle-ci peut interagir avec une classe interface préexistante pour manipuler un objet du même type dans le projet. Cette classe est créée automatiquement lors de l'ouverture d'un nouveau projet interactif.



Lors de la création d'un projet, deux classes sont automatiquement visibles.

La classe interface, « Player » dans cet exemple, représente la classe avec laquelle l'élève peut interagir pour manipuler le programme. Toutes les méthodes que le contrôleur peut appeler sont montrées dans l'UML de cette classe. Le code source de cette classe est inaccessible.

La classe « Controller » est la classe qui doit être complétée par l'élève. Dans cette classe, un attribut du type de la classe interface est créé, ainsi qu'un manipulateur pour celui-ci. Ceux-ci ne doivent pas êtres supprimés pour assurer le bon fonctionnement du projet interactif.

1.5.2. Projets basés sur un modèle

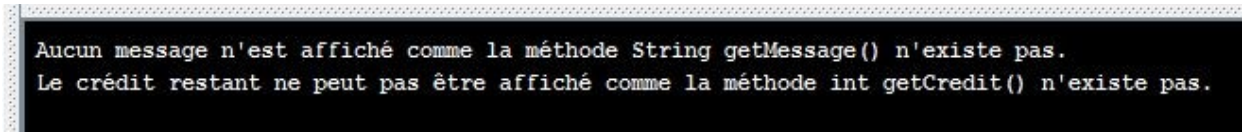
Dans ce type de projet, l'élève doit compléter la classe modèle du projet. Des classes préexistantes, qui ne sont pas visibles, font appel à cette classe modèle pour pouvoir exécuter le programme.

Les méthodes dont l'implémentation est nécessaire sont référencées dans les exercices

respectifs d'un projet concret. Ces méthodes varient d'un projet à l'autre.

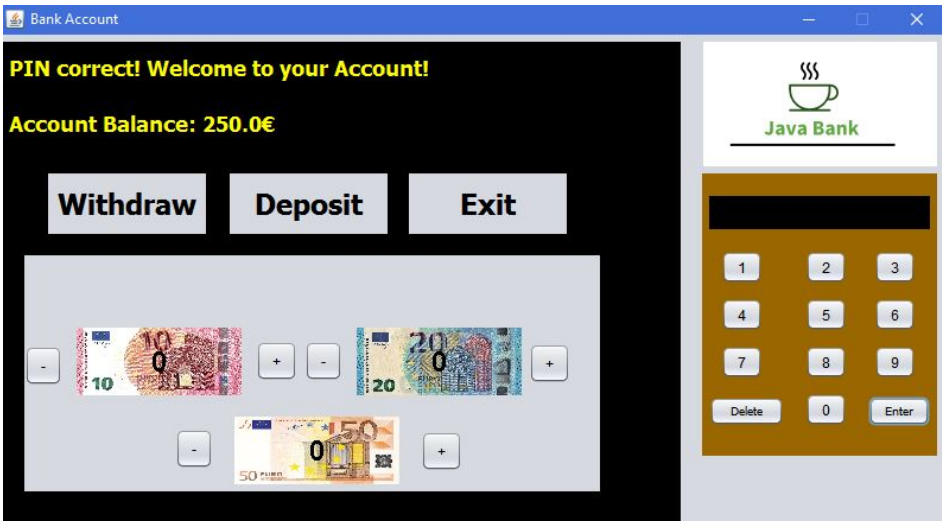
La classe modèle est automatiquement crée lors de la création d'un nouveau projet. Elle ne doit pas, mais peut déjà contenir des attributs et des méthodes.

Comme il se peut qu'une méthode nécessaire au bon fonctionnement du projet n'est pas encore implémentée, « Unimozer » affiche un message dans la console qui indique de qu'elle(s) méthode(s) il s'agit.



Bank Account

Exercice 1 : Bank Account



Dans cet exercice, vous allez compléter la classe «Account» qui représente un compte bancaire. Pour ceci, créez un nouveau projet interactif (Project → New interactive project…) et sélectionnez l'exercice « Bank Account ». Par la suite vous devez compléter la classe **Account**.

Account
− balance : double − pin : int − message : String
+ Account( ) + isPinValid(pPin : int) : boolean + withdraw(pSum : double) : boolean + deposit(pSum : double) : void + getBalance( ) : double + getMessage( ) : String

1. Ajoutez l'attribut réel **balance** qui représente le montant d'argent en € présent sur le compte. Ajoutez l'accesseur **getBalance** à la classe.
2. Ajoutez l'attribut entier **pin** qui représente le code pour accéder au compte bancaire.
3. Ajoutez l'attribut **message** qui représente une chaîne de caractères. Ajoutez un accesseur **getMessage** à la classe.
4. Ajoutez un constructeur qui :
   - Initialise **balance** à la valeur 250.
   - Initialise **pin** à un nombre entier aléatoire dans l'intervalle [1000;9999].

- Initialise **message** à « Welcome! Please enter your PIN. »

5. Ajoutez la méthode **isPinValid**, qui retourne une valeur booléenne. La méthode prend un entier **pPin** en paramètre et vérifie si le code pin passé en paramètre est valide.
   - Si le code **pPin** donné correspond au code **pin** du compte, message est mis à « PIN correct! Welcome to your account. » et la méthode retourne **true**.
   - Sinon, message est mis à « Incorrect PIN! Please try again. ».

   Exécutez le programme (Project → Run…) et vérifiez si vous pouvez accéder au compte bancaire. Vous pouvez voir le code pin généré automatiquement dans l'objet **account** créé dans Unimozer.



6. Ajoutez la méthode **deposit**, qui prend un entier **pSum** en paramètre. Le montant spécifié par **pSum** est ajouté au montant d'argent du compte bancaire. En plus, **message** est mis à « You added *xy* € to your account. », ou xy est remplacé par la valeur de **pSum**.



   Exécutez le programme et essayez de déposer de l'argent au compte. Vous pouvez changer le montant à ajouter en utilisant les boutons « + » et « - » à côté des billets. Le montant sélectionné peut être ajouté en utilisant le bouton « Deposit ».

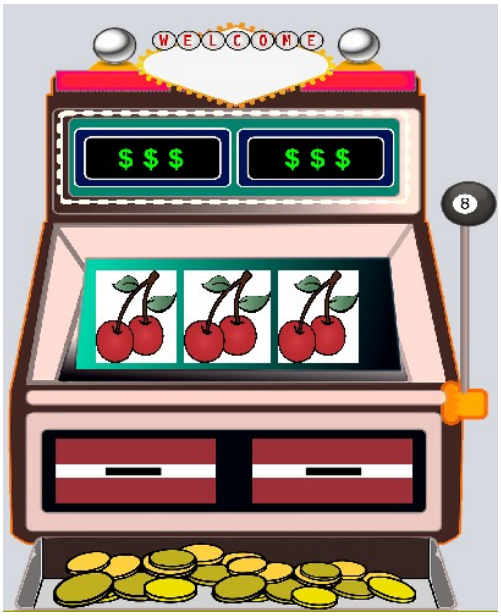
7. Ajoutez la méthode **withdraw**, qui prend un entier **pSum** en paramètre et retourne une valeur booléenne. Le montant spécifié par **pSum** est retiré du montant d'argent du compte bancaire. Pour ceci il faut d'abord vérifier s'il y a assez d'argent sur le compte. Il existe deux cas :
   - Si le compte a assez d'argent, le montant spécifié par **pSum** est retiré du compte. En plus, message est mis à « Here are your *xy* €. », ou xy est remplacé par la valeur de **pSum**. Ensuite la méthode retourne **true**.
   - Sinon, aucune valeur est retirée du compte et **message** est mis à « You don't have enough money. ». Ensuite la méthode retourne **false**.

Notions requises :	classe, objet, méthode, paramètre, type, attribut, void, boolean constructeur, Math.random()
Structures requises :	If (simple)
Niveau destiné :	3GTG

Slot Machine

Exercice 2 : Slot Machine

Dans cet exercice, vous allez compléter la classe « SlotMachine » qui représente un bandit manchot (DE : einarmiger Bandit). Pour ceci, créez un nouveau projet interactif (Project → New interactive project…) et sélectionnez l'exercice « Slot Machine ».



SlotMachine
− credit : int
− pricePerGame : int
− gamesPlayed : int
− reel1 : int
− reel2 : int
− reel3 : int
− message : String
+ getCredit( ) : int
+ getReel1( ) : int
+ getReel2( ) : int
+ getReel3( ) : int
+ getMessage( ) : String
+ SlotMachine( )
+ pullArm( ) : boolean
+ hasWon( ) : boolean

1. Ajoutez les attributs entiers **reel1**, **reel2**, **reel3** qui représentent les 3 rouleaux (EN:reel) du bandit manchot. Ceux-ci sont représentés par des nombres entiers, le projet interactif va automatiquement les représenter par des images.
2. Ajoutez les accesseurs **getReel1**, **getReel2** et **getReel3** et manipulateurs **setReel1**, **setReel2** et **setReel3** des attributs respectifs.
3. Exécutez votre programme (Project→ Run) et essayez d'exécuter manuellement les méthodes **setReel1**, **setReel2** et **setReel3** avec des nombres dans l'intervalle [0;5].
4. Implémentez la méthode **boolean pullArm**, qui attribue un nombre aléatoire dans l'intervalle [0;5] à **reel1**, **reel2** et **reel3**. Ensuite elle retourne la valeur **true**. Ensuite essayez votre programme et cliquez sur le levier du bandit manchot.
5. Ajoutez l'attribut entier **credit** qui représente le crédit actuel du joueur et initialisez-le à 50. Ajoutez aussi un accesseur **getCredit** à la classe.
6. Ajoutez un attribut entier **gamesPlayed** qui est initialisé à 0. Cet attribut est utilisé pour compter le nombre de jeux joués.
7. Ajoutez l'attribut entier **pricePerGame** et initialisez-le à 5. Ce nombre représente le prix à payer pour pouvoir jouer une partie.
8. Ajoutez l'attribut **message** qui représente une chaîne de caractères, ainsi que

l'accesseur **getMessage**. L'attribut **message** est initialisé à « Welcome ».

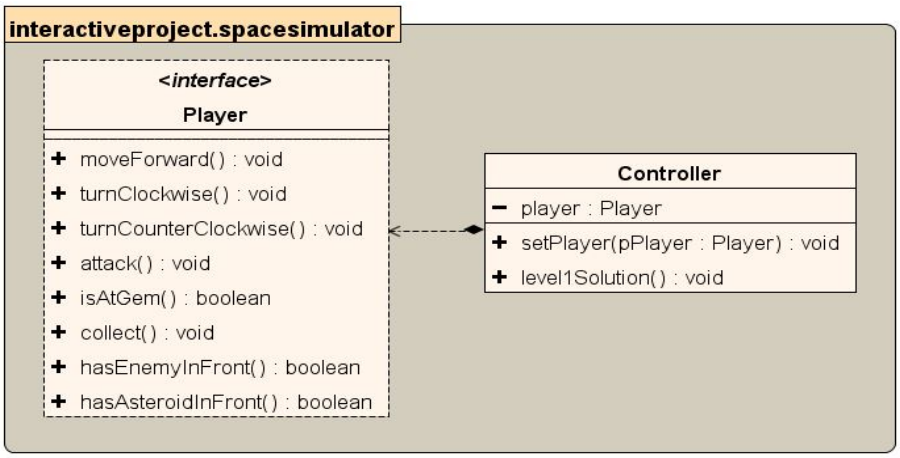
9. Modifiez la méthode **pullArm** comme suit :
   - Vérifiez si le joueur a assez de crédit pour pouvoir jouer un jeu.
   - Si c'est le cas :
     - incrémentez le nombre de jeux joués.
     - Décrémentez le crédit par le prix d'un jeu.
     - Attribuez un nombre entier aléatoire dans l'intervalle [0;5] à **reel1**, **reel2** et **reel3** et retournez la valeur **true**.
   - Sinon,le joueur ne peut plus jouer, attribuez un texte approprié à **message** et retournez la valeur **false**.

   Essayez votre programme en cliquant sur le levier du bandit manchot.
10. Ajoutez la méthode **boolean hasWon**, qui vérifie si le jouer a gagné :
    - Si les 3 attributs reel1, reel2 et reel3 sont egaux :
      - Le joueur gagne 100 crédits.
      - Le message est mis à « You win the Jackpot of 100 credits! »
      - La méthode retourne **true**.
    - Sinon, si au moins deux des attributs sont égaux :
      - Le joueur gagne 10 crédits.
      - Le message est mis à « You've got a double! You win 10 credits! ».
      - La méthode retourne **true**.
    - Sinon le jouer perd :
      - Le message est mis à « You lose! »
      - La méthode retourne **false**.

Notions requises :	classe, objet, méthode, paramètre, type, attribut, void, boolean constructeur, Math.random()
Structures requises :	If, conditions composées
Niveau destiné :	3GTG

Space Simulator

Ce projet contient plusieurs niveaux d'un jeu. Le but est d'implémenter des méthodes dans la classe « Controller » qui servent à contrôler le vaisseau spatial et de résoudre les différents niveaux du jeu.



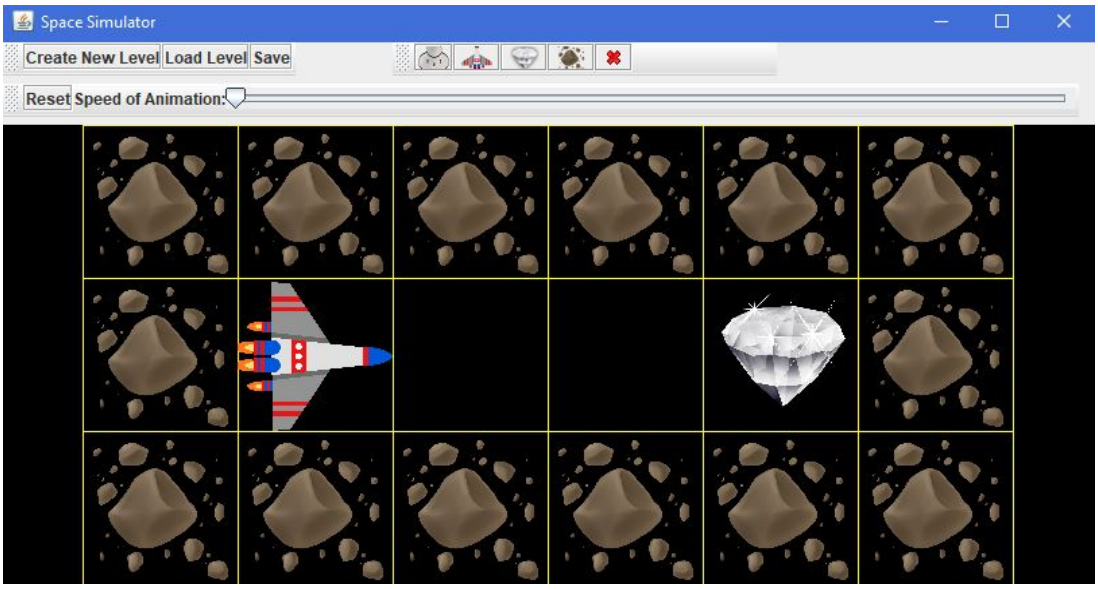
Seule la classe « Controller » peut être changée. Celle-ci contient déjà un attribut du type « Player » qui sert à interagir avec le vaisseau spatial. L'UML ci-dessus montre toutes les méthodes accessibles de la classe « Player ».

Le but du jeu est de recueillir tous les diamants dans un niveau et d'éviter tous les astéroïdes et ennemis. Les ennemis peuvent aussi êtres détruits à l'aide de la méthode **attack()** .

Exercice 3 : Premiers pas avec le Space Simulator

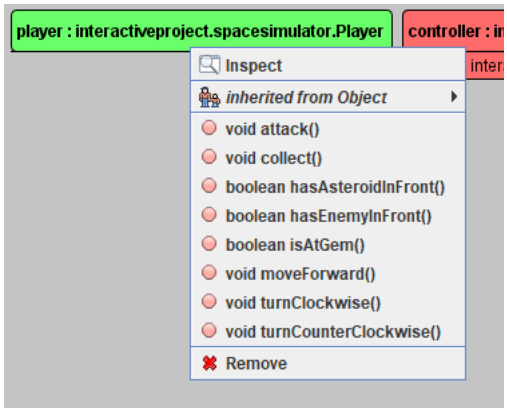
Le but de cet exercice est de pouvoir interagir manuellement avec la classe **Player**.

Pour ceci, exécutez votre projet (Project → Run). Lors de l'exécution, une fenêtre s'ouvre qui montre le jeu en question.



En bas, dans « Unimozer », vous voyez deux nouveaux objets qui ont été créés, un objet de votre classe « Controller » et un objet de la classe « Player ».

Essayez maintenant manuellement de résoudre ce niveau 1 du jeu, en faisant appel aux méthodes de l'objet « Player ». Le but est de recueillir tous les diamants.

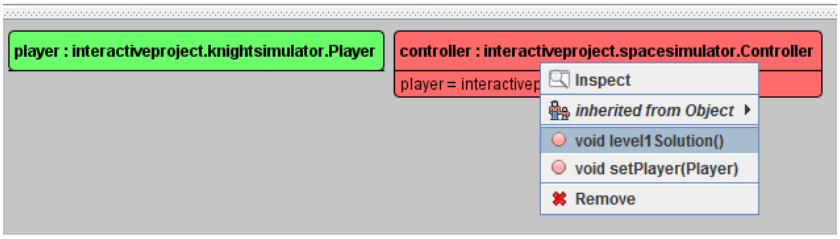


Exercice 4 : Résoudre un niveau en implémentant un méthode (introduction)

Un exemple d'une méthode est déjà implémentée. La méthode **level1Solution** permet de résoudre le niveau 1 du jeu.

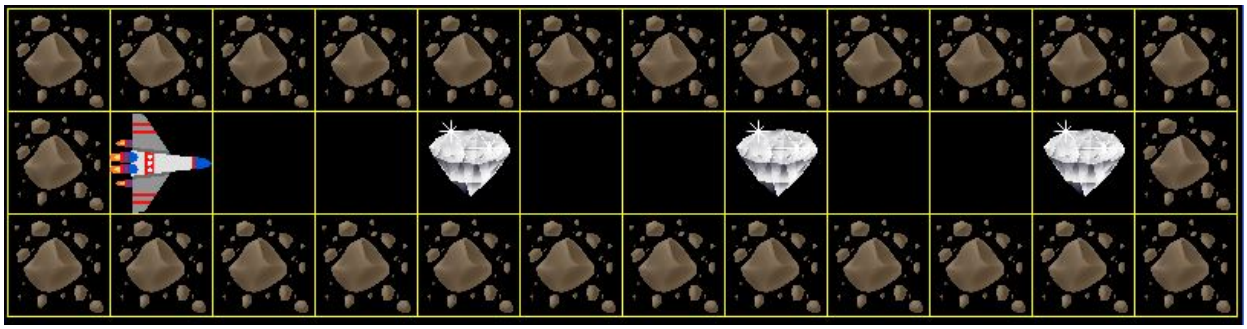
Regardez le code de cette méthode et essayez de prévoir ce qui va se passer lors de l'exécution de cette méthode.

Ensuite exécutez votre projet et faites appel à cette méthode de l'objet « controller ».



Exercice 5 : Résoudre un niveau en implémentant une méthode (Level 2)

Implémentez la méthode **level2Solution** qui permet de résoudre le niveau 2 du jeu. Pour ouvrir un nouveau niveau du jeu, vous devez cliquer sur « Load Level » dans l'interface du jeu.

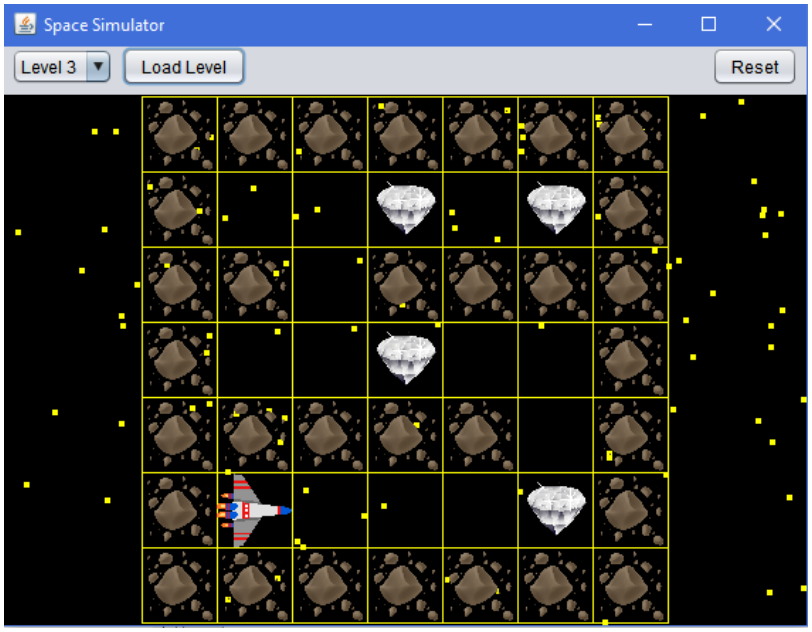


Exercice 6 : Solution à l'aide d'une boucle (Level 2)

Essayez à nouveau de résoudre le niveau 2 du jeu, mais cette fois-ci en utilisant un structure répétitive « pour ».

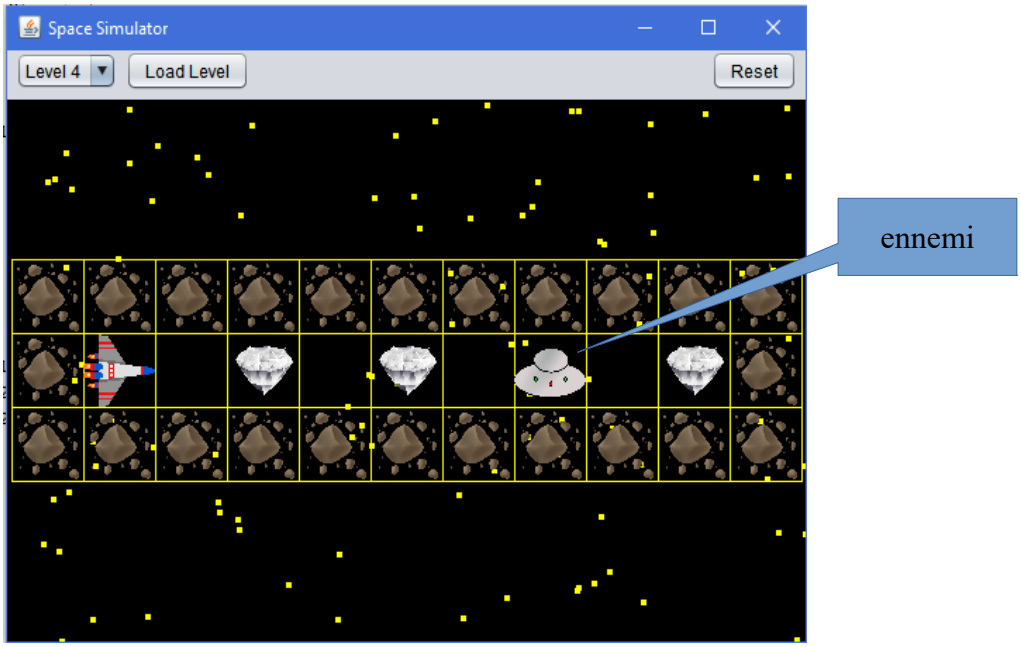
Exercice 7 : Level 3

Essayez de résoudre le niveau 3 du jeu! Pour ceci, vous pouvez utiliser les méthodes **turnClockwise** et **turnCounterClockwise** pour faire tourner le vaisseau spatial.



Exercice 8 : Ennemis (Level 4)

Dans ce niveau de jeu, il y a une nouveauté, des ennemis.



Vous pouvez utiliser la méthode **hasEnemyInFront**, qui retourne **true** si un ennemi se

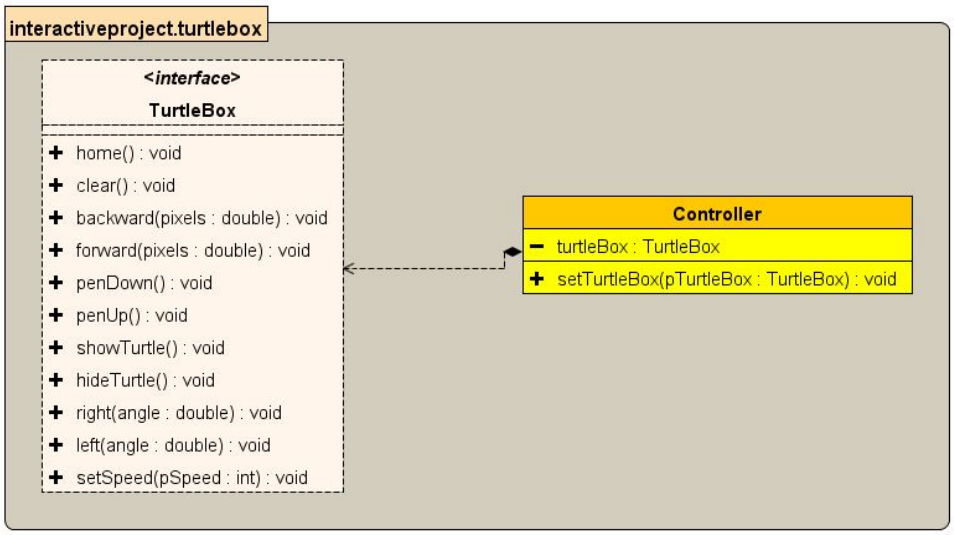
trouve devant le vaisseau.

En plus vous disposez de la méthode **attack** qui permet de détruire un ennemi si celui-ci se trouve dans la case directement devant le vaisseau.

Écrivez un méthode level4Solution, qui permet de résoudre ce niveau du jeu. Pour avancés : Essayez d'utiliser une structure répétitive « pour ».

TurtleBox

Ce projet contient une tortue qui peut être utilisée pour tracer des figures. Le but est d'implémenter des méthodes dans la classe « Controller » qui servent à contrôler la tortue pour tracer différentes figures. Voici un diagramme UML qui montre les méthodes utilisables de la classe « TurtleBox » :



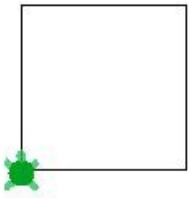
Il existe déjà un attribut du type « TurtleBox » dans la classe « Controller » qui peut être utilisée pour interagir avec la tortue.

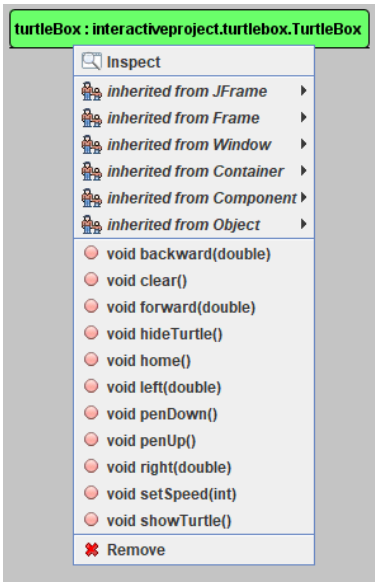
Utilisez à chaque fois les méthodes **clear(…)** et **home(…)** au début d'une méthode, pour effacer la surface de dessin et pour remettre la tortue au point de départ.

Exercice 9 : Dessiner un carré

Exécutez le projet (Project→Run) et essayez de manuellement dessiner un carré. La longueur d'un côté est à votre choix. Utilisez pour ceci les méthodes **forward(…)**, **backward(…)**, **right(…)** et **left(…)** de la classe TurtleBox. Les angles des méthode **right(…)** et **left(…)** sont exprimés en degrés.

Faites appel au méthodes dans Unimozer quand le projet a été exécuté :





Exercice 10 : Dessiner d'autres figures

Essayez maintenant de dessiner d'autres figures manuellement :

- un triangle équilatéral
- un triangle rectangle et isocèle
- pour avances : un triangle isocèle

Exercice 11 : Utiliser des méthodes pour dessiner des figures

- `drawSquare(double pSideLength)`
- `drawEquilateralTriangle(double pSideLength)`
- `drawIsocelesRightangledTriangle(double pBase)`
- `drawIsocelesTriangle(double pBase, double pHeight)` `//p.avancés`
- `drawHouse(double pBase)`

Exercice 12 : Figures plus complexes

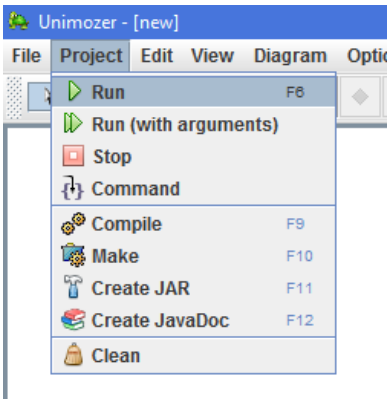
- `drawPolygon(double pSideLength, int pN)` qui dessine un polynôme équilatéral et équiangle de n cotes
- `drawCircle(double pRadius)` qui dessine un cercle de rayon `pRadius` trace de façon aussi précise que possible
- `drawCircleCenter(double pRadius)` qui dessine un cercle de rayon `pRadius` trace de façon aussi précise que possible. Au début et a la fin, la tortue se trouve au milieu du cercle
- `drawCar(double pLength)` qui dessine une voiture

ANHANG C

Erstellte Dokumentation für Lehrer

Comment créer un projet interactif

1.1. Qu'est-ce qu'un projet interacftif ?



Un projet interactif est un projet avec des classes prédéfinies. Ces projets peuvent être ouverts et édités dans le logiciel « Unimozer ».

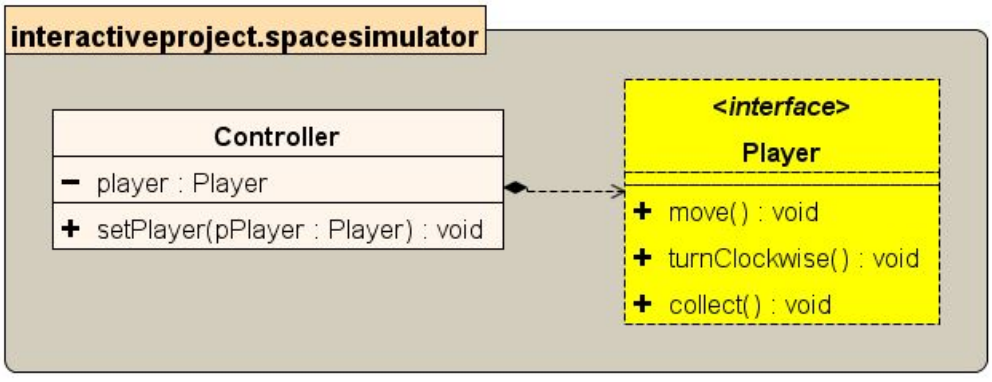
Le but de ces projets est que les classes des élèves sont visualisées dans Unimozer à l'aide d'un interface graphique.

1.2. Les deux types de projets

Il existe deux types différents de projets, qui sont expliqués en détail par la suite.

1.2.1. Projets basés sur un contrôleur

Dans ce type de projet, l'élève doit compléter une classe « Controller ». Celle-ci peut interagir avec une classe interface préexistante pour manipuler un objet du même type dans le projet. Cette classe est créée automatiquement lors de l'ouverture d'un nouveau projet interactif.



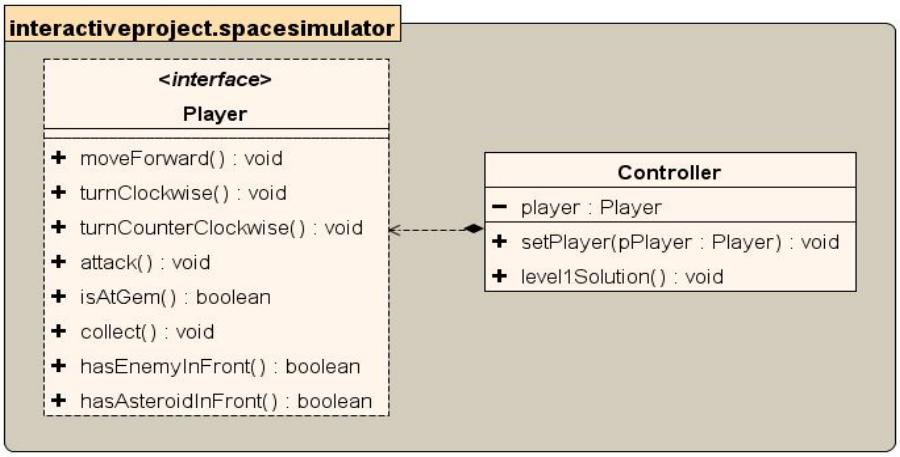
Lors de la création d'un projet, deux classes sont automatiquement visibles.

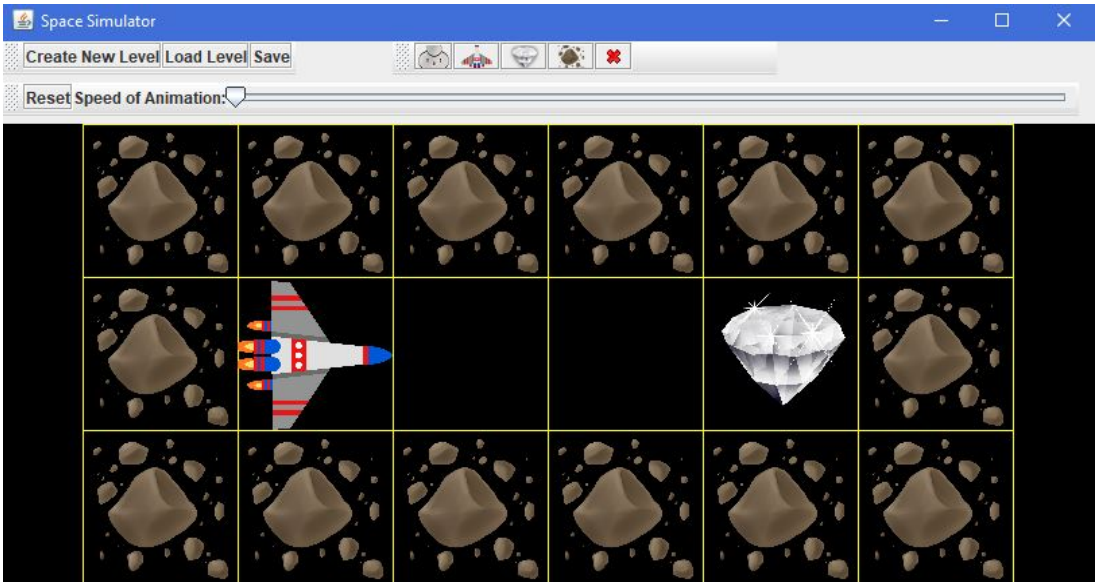
La classe interface, « Player » dans cet exemple, représente la classe avec laquelle l'élève peut interagir pour manipuler le programme. Toutes les méthodes que le contrôleur peut appeler sont montrées dans l'UML de cette classe. Le code source de cette classe est inaccessible.

La classe « Controller » est la classe qui doit être complétée par l'élève. Dans cette classe, un attribut du type de la classe interface est créé, ainsi qu'un manipulateur pour celui-ci.

Ceux-ci ne doivent pas êtres supprimés pour assurer le bon fonctionnement du projet interactif.

Un exemple de ce type de projet est « Space Simulator » qui peut être trouvé dans le document avec les exercices sur les projets interactifs. Voici le diagramme UML de ce projet. L'élève doit implémenter la classe « Controller » pour naviguer le vaisseau spatial, représenté par la classe « Player ». Le diagramme montre toutes les méthodes accessibles de cette classe.





1.2.2. Projets basés sur un modèle

Dans ce type de projet, l'élève doit compléter la classe modèle du projet. Des classes préexistantes, qui ne sont pas visibles, font appel à cette classe modèle pour pouvoir exécuter le programme.

Les méthodes dont l'implémentation est nécessaire sont référencées dans les exercices respectifs d'un projet concret. Ces méthodes varient d'un projet à l'autre.

La classe modèle est automatiquement crée lors de la création d'un nouveau projet. Elle ne doit pas, mais peut déjà contenir des attributs et des méthodes.

Comme il se peut qu'une méthode nécessaire au bon fonctionnement du projet n'est pas encore implémentée, « Unimozer » affiche un message dans la console qui indique de qu'elle(s) méthode(s) il s'agit.

```
Aucun message n'est affiché comme la méthode String getMessage() n'existe pas.
Le crédit restant ne peut pas être affiché comme la méthode int getCredit() n'existe pas.
```

Un exemple de ce type de projet est « Bank Account » qui peut être trouvé dans le document avec les exercices sur les projets interactifs. Dans ce projet, l'élève doit compléter la classe « Account » qui est utilisé par l'interface graphique de ce projet.

1.3. Comment créer un projet interactif ?

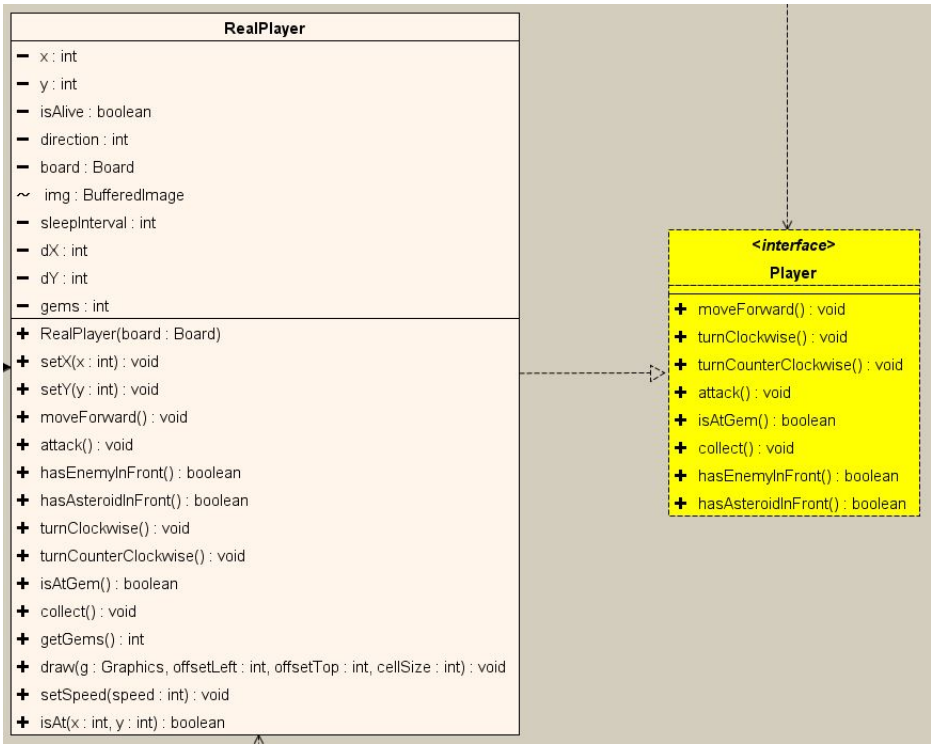
Pour créer un projet interactif, il faut respecter certaines contraintes, qui sont dépendantes du type de projet choisi, ainsi que quelques contraintes communes.

1.3.1. Contraintes communes

- Toutes les classes qui sont utilisées par le projet doivent se trouver dans Unimozer. Si le projet est créé en Netbeans, il faut ouvrir celui-ci en Unimozer.
- Le projet doit avoir une classe qui est la classe « main » du projet. Celle-ci doit être une extension de la classe « JFrame ». Principalement il est une bonne pratique de garder l'appeler « MainFrame » comme dans les exercices du cours. Celle-ci doit posséder le constructeur par défaut « MainFame() ».
- Toutes les classes doivent se trouver dans un même paquet (package). Le paquet par défaut peut être utilisé.
- Le projet doit contenir une classe qui doit être complété par l'étudiant. Celle-ci peut être vide, mais doit au moins contenir la structure qui définit la classe.

1.3.2. Contraintes pour un projet interactif basé sur un contrôleur

Dans le diagramme UML suivant, on voit deux classes du projet interactif « Space Simulator ». La classe « RealPlayer » est l'implémentation réelle du vaisseau spatial qui est dirigé par la classe « Controller » de l'élève. Elle contient des attributs et méthodes qui sont inutiles pour l'élève, mais importants pour le fonctionnement interne. Pour masquer ces attributs et méthodes, une classe « Interface » nommée « Player » est implémentée, qui ne contient que les méthodes utiles pour l'élève. La classe « RealPlayer » est une implémentation de cet interface.



- Il faut avoir une classe du type « Interface » qui sert à l'élève pour interagir avec la classe réelle qui se trouve dans le projet, mais qui est cachée pour l'élève.
- Cette classe « Interface » contient toutes le méthodes de la classe réelle auxquelles l'élève doit avoir accès.
- La classe que l'élève doit compléter doit être appelée « Controller ».
- La classe « MainFrame » doit avoir un attribut public de la classe réelle avec laquelle l'élève peut interagir. (Dans l'exemple un attribut du type « RealPlayer »)

1.3.3. Contraintes pour un projet interactif basé sur un modèle

Dans ce type de projet, l'élève doit compléter une classe qui sert comme modèle dans le programme (selon le modèle MVC). Le problème principal est que les méthode que l'élève doit écrire n'existent pas lors de la création du projet. Ce problème va générer des problèmes à la compilation dans des classes invisibles pour l'élève et le projet ne peut pas être exécuté jusque toutes les méthodes ont été implémentées.

Pour éviter ce problème, les autres classes dans le projet doivent faire appel aux méthodes de l'élève en utilisant de la réflexion. Dans l'exemple suivant je montre comment un objet de la classe modèle est initialisé dans le « MainFrame » :

```
private Object studentObject ;

...
public MainFrame() {
...
  try {
      Class studentClass =
            Class.forName("interactiveproject.bankaccount.Account");
       studentObject = studentClass.newInstance();

  }catch(ClassNotFoundException|InstantiationException|
    IllegalAccessException ex) {
      System.out.println("Le constructeur Account() n'a pas ete trouve");
      }
}
```

Toutes les exceptions qui peuvent avoir lieu, doivent être traitées et un message compréhensible est écrit dans la console pour l'élève.

En plus, la classe doit avoir un accesseur à l'attribut qui représente la classe de l'élève, qui est appelé getStudentObject(). Ceci doit être le cas, pour qu'une instance de cet attribut peut être affiche pour l'élève dans Unimozer

```
public Object getStudentObject() {
        return studentObject;
    }
```

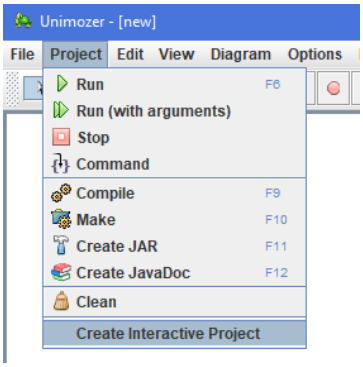
Finalement, à chaque fois que la classe « MainFrame » fait appel à une des méthodes de la classe de l'étudiant, cette méthode doit être appelée par réflexion et les exceptions doivent être traitées :

```
 try {
           //Appel de la méthode getMessage de la classe de l'étudiant
           Method method = studentClass.getMethod("getMessage");
           Object o = method.invoke(studentObject);
           messagePanel.setText((String) o);
     } catch (NoSuchMethodException | SecurityException |
           IllegalAccessException | IllegalArgumentException |
           InvocationTargetException ex) {
             System.out.println("Aucun message ne peut etre affiche,
comme la methode String getMessage n'est pas implementee");
             messageError = true;
     }
```

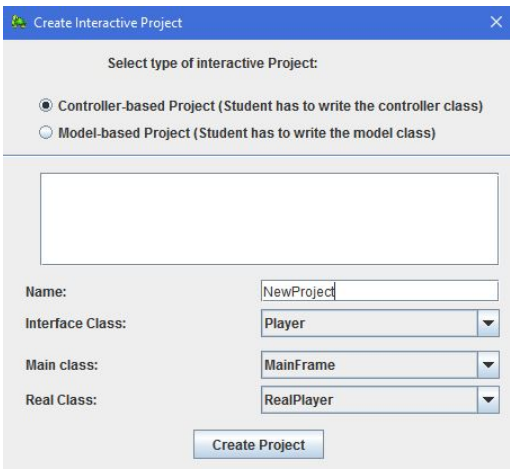
L'implémentation de ce type de projet est un peu plus complexe que l'implémentation d'un projet qui utilise un contrôleur, comme toutes les exceptions doivent être traitées et car aucun appel direct des méthodes de la classe de l'élève est possible.

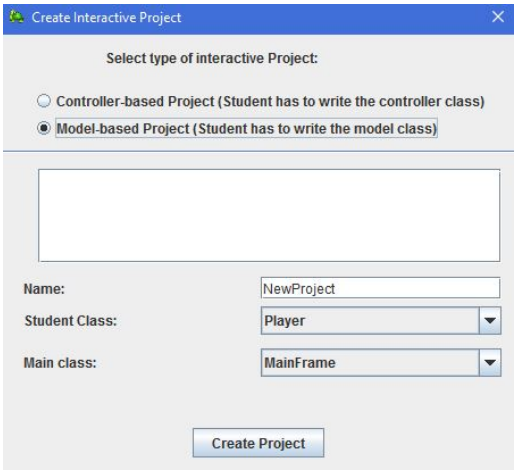
1.3.4. Conversion d'un projet en un projet interactifs

Un système de conversion d'un projet normal en un projet interactif est disponible dans Unimozer. Pour faire ceci, il faut d'abord ouvrir un projet qui contient toutes les classes nécessaires et qui remplit les contraintes respectives. Ensuite le projet peut être convertit en faisant appel au menu suivant :



Ceci fait apparaître un menu qui sert à créer les deux types de projets :





Pour les deux types de Projet on doit donner des informations différentes, qui sont dépendantes des classes présentes dans le projet :

- Name : Le nom du projet interactif.
- Main Class : La classe principale du projet, qui doit être une extension de la classe JFrame.

Pour les projets interactifs basés sur un contrôleur :

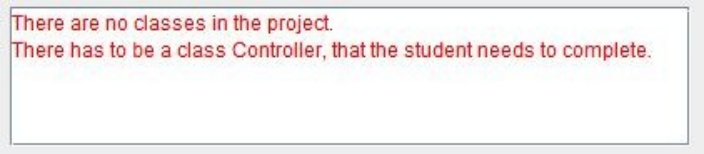
- Interface Class : La classe du type « Interface » qui sert à l'interaction de l'élève.
- Real Class : La classe réelle qui implémente la classe interface.

Pour les projets interactifs basés sur un modèle:

- Student Class : La classe que l'élève doit compléter et qui est visible et éditable pour celui-ci.

Unimozer fait déjà une présélection des classes possibles à l'aide des contraintes données.

Si jamais une contrainte n'est pas atteinte, Unimozer signale le problème :



Après la conversion, le projet peu être sauvegardé comme un projet normal et peut ainsi être donné aux élèves. Lors de l'ouverture du projet, Unimozer va automatiquement reconnaître celui-ci comme un projet interactif.

1.3.5. Questions ou problèmes / Intégration dans Unimozer

Si vous avez d'autres questions sur les projets interactifs, respectivement vous voulez que votre projet est intégré fixe dans Unimozer, vous pouvez toujours envoyer un courriel à ronny.osweiler@education.lu.