

Entwurf und Implementierung eines Echtzeit-Strategie-Aufbauspiels (ESA) in Vogelperspektive

Seminarkurs 2017/2018

Gemacht von:

Robert Ryan

Christopher Schleppe

Tobias Schnitzler

Betreuer:

Herr Rolf Scheuermann

Stand 04.03.18

Inhaltsverzeichnis

[1 Einleitung 4](#_Toc508124701)

[1.1 Motivation 4](#_Toc508124702)

[1.2 Ideen 4](#_Toc508124703)

[1.3 Ziele und Spielprinzip 4](#_Toc508124704)

[2 Durchführungsteil 6](#_Toc508124705)

[2.1 Grundstruktur/Klassendiagramme 6](#_Toc508124706)

[2.1.1 Grundaufbau 6](#_Toc508124707)

[2.1.2 libGDX 6](#_Toc508124708)

[2.1.3 Android Studio 6](#_Toc508124709)

[2.1.4 Gesamtüberblick 6](#_Toc508124710)

[2.1.4.1 World 7](#_Toc508124711)

[2.1.4.2 Actor 7](#_Toc508124712)

[2.1.4.3 Player 7](#_Toc508124713)

[2.1.4.4 Item 7](#_Toc508124714)

[2.1.4.5 Saving 7](#_Toc508124715)

[2.1.4.6 Types 7](#_Toc508124716)

[2.1.4.7 GameLoop 8](#_Toc508124717)

[2.1.5 Übersicht Inventar 9](#_Toc508124718)

[2.2 World 9](#_Toc508124719)

[2.2.1 Grundprinzip 9](#_Toc508124720)

[2.2.2 Tile 10](#_Toc508124721)

[2.2.3 FLayers 10](#_Toc508124722)

[2.2.4 Actor 11](#_Toc508124723)

[2.2.4.1 Conveyor 12](#_Toc508124724)

[2.2.4.2 Oven 13](#_Toc508124725)

[2.2.4.3 Clutch 14](#_Toc508124726)

[2.2.4.4 ElectricOven 16](#_Toc508124727)

[2.2.4.5 Generator 17](#_Toc508124728)

[2.2.4.6 Powerline 18](#_Toc508124729)

[2.2.4.7 Solarpanel 18](#_Toc508124730)

[2.2.4.8 Miner 18](#_Toc508124731)

[2.2.5 SaveGame 20](#_Toc508124732)

[2.2.6 PlayerController 21](#_Toc508124733)

[2.2.6.1 Input 21](#_Toc508124734)

[2.2.6.2 Kollision 22](#_Toc508124735)

[2.2.6.3 Equipment 22](#_Toc508124736)

[2.3 PathFinding 23](#_Toc508124737)

[2.4 Inventar 24](#_Toc508124738)

[2.4.1 Items 26](#_Toc508124739)

[2.4.1.1 Waffen 26](#_Toc508124740)

[2.4.1.2 Werkzeuge 27](#_Toc508124741)

[2.4.1.3 Verbrauchsstoffe 27](#_Toc508124742)

[2.4.1.4 Liste 27](#_Toc508124743)

[2.4.2 Crafting 27](#_Toc508124744)

[2.5 Speichern 27](#_Toc508124745)

[2.5.1 Grundlegende Funktion 27](#_Toc508124746)

[2.6 Zukünftige Pläne 27](#_Toc508124747)

[3 Schlussteil 28](#_Toc508124748)

[4 Literaturverzeichnis 29](#_Toc508124749)

[5 Zeitplan 30](#_Toc508124750)

[6 Gruppensitzungsprotokolle 31](#_Toc508124751)

[6.1 1. Treffen 31](#_Toc508124752)

[6.2 2. Treffen 32](#_Toc508124753)

[6.3 3. Treffen 33](#_Toc508124754)

[6.4 4. Treffen 34](#_Toc508124755)

[6.5 5. Treffen 35](#_Toc508124756)

[6.6 6. Treffen 36](#_Toc508124757)

[6.7 7.Treffen 37](#_Toc508124758)

[6.8 8. Treffen 38](#_Toc508124759)

[6.9 9. Treffen 39](#_Toc508124760)

[6.10 10. Treffen 40](#_Toc508124761)

[6.11 11. Treffen 41](#_Toc508124762)

[6.12 12. Treffen 42](#_Toc508124763)

[6.13 13. Treffen 43](#_Toc508124764)

[6.14 14. Treffen 44](#_Toc508124765)

[6.15 15. Treffen 45](#_Toc508124766)

[6.16 16. Treffen 46](#_Toc508124767)

[7 Zeitnachweis 47](#_Toc508124768)

[7.1 Robert Ryan 47](#_Toc508124769)

[7.2 Christopher Schleppe 47](#_Toc508124770)

[7.3 Tobias Schnitzler 48](#_Toc508124771)

# Anmerkung

Zur besseren Übersicht sind im Folgenden Klassen orange, Methoden blau und Parameter sowie Attribute violett geschrieben.

# Einleitung

## Motivation

Da zusätzliches Wissen im Bereich Informatik im Berufsleben von hoher Bedeutung ist und seitens der Teilnehmer der Seminararbeit viel Interesse für ein genaueres Verständnis in der Thematik der Spieleprogrammierung besteht, lag die Teilnahme an einem Seminarkurs nahe. Indiespiele inspirierten, ein eigenes kleines Spiel zu entwickeln. Eine grundlegende Erfahrung in der Spieleprogrammierung bot uns das kleine Projekt „Schiffe versenken“.  
 Zudem ist es eine neue Herausforderung, ein Spiel im Team zu planen und zu entwickeln. Einen festgelegten Zeitplan einzuhalten, bedeutet für jeden einzelnen Teilnehmer, ebenso wie für die Gruppe als Ganzes, eine große Aufgabe. Diszipliniertes Arbeiten, eine hohe Motivation, sowie die Bereitschaft, sich in der Gruppe miteinander auseinanderzusetzen und abzusprechen, sind unabdingbar für den Erfolg unserer Seminararbeit. Durch alternative Lösungswege und Ideen von Gruppenmitgliedern lässt sich viel lernen und der Programmierstil verbessern.

## Ideen

Bereits zu Beginn war klar, dass ein Spiel entstehen sollte, dessen Genre jedoch war noch unklar. Es stellte sich als schwierig heraus, die Komplexität richtig einzuschätzen.   
Der Umfang überstieg oftmals unsere Fähigkeiten und unser Wissen, zum Beispiel bei einem Echtzeitstrategiespiel aufgrund des komplexen Pathfindings. Deshalb entschieden wir uns für ein Spiel, welches sich an „Factorio“\* des Indiegameentwicklers „Wube Software LTD“ orientiert.

## Ziele und Spielprinzip

Das Ziel des Projektes ist es, ein Spiel, das „Factorio“ ähnelt, zu entwickeln. Der Spieler ist in einer Welt, die aus Quadraten besteht. er sieht die Welt oben herab und steuert einen Charakter, der Ressourcen abbauen kann. Bei dem zu entwickelnden Spiel baut der Spieler Ressourcen ab, um dann aus einzelnen Elementen eine Fabrik aufzubauen, welche er mit der Zeit automatisieren und vergrößern kann. Diese Elemente sind zum Beispiel automatische Ressourcen Förderer und Förderbänder, um Ressourcen zu transportieren. Der Spieler besitzt ein Inventar, das Ressourcen speichern und ordnen kann. Zudem soll es noch Gegner geben die den Spieler verfolgen und ihn töten können, dadurch wird das Spiel spannender und der Spieler hat eine Herausforderung. Das Spiel soll am Ende fehlerfrei funktionieren, spielbar sein und Spaß machen.

Abbildung 1: Factorio Einblick

\*Factorio ist ein Sci-Fi-Simulationspiel. Der Spieler sammelt Ressourcen und baut sich damit eine Fabrik auf, die man immer mehr automatisieren kann

# Durchführungsteil

## Grundstruktur/Klassendiagramme

### Grundaufbau

Das Projekt ist in drei Teile gegliedert, dies ist praktisch da pro Person ein Themen Gebiet ergibt:

Der erste Teil beinhaltet alles was in der Welt platziert wird, also zum Beispiel den Spieler, und alle Teile der Fabrik. Natürlich beinhaltet dieser Teil auch die Welt selbst. Da das Themengebiet sehr groß ist Arbeiten zwei Personen daran

Der zweite Teil besteht aus dem Inventar des Spielers und der GUI. Hinzu kommen noch die Items und das fertigen von Gegenständen.

Speicherung der Items des Spielers. Das Speichern wird benötigt, um den Fortschritt des Spielers nach dem Schließen der Anwendung nicht zu verlieren.

Für dieses Projekt wird "libGDX" und "Android Studio" verwendet. libGDX liefert viele nützliche Klassen, die performanter und besser angepasst sind. Android Studio bietet ein gute Entwicklungsumgebung, da sie beim schreiben durch zum Beispiel eine bessere automatische Vervollständigung, als bei Netbeans, den Workflow verbessern.

### libGDX

LibGDX ist ein Java-Framework, welches für plattformunabhängige Spieleentwicklung entwickelt wurde. LibGDX vereinfacht die Spieleentwicklung, es verfügt über Klassen und Methoden, welche die Performance verbessern. Zudem vereinfacht es das Zeichnen und andere Aufgaben.

Die wichtigsten Klassen, die libGDX liefert sind SpriteBatch und Disposable.

Der SpriteBatch ist für das Zeichnen verantwortlich und hat die Methode Draw. Es erbt von dem Batch, dies zeichnet möglichst effektiv und optimiert. Ein Batch wird über OpenGL gemanagt. Ein Batch ist ein Performance kritisches Objekt, deshalb sollte man nur eines erstellen.

Ein weiteres wichtiges Interface von LibGDX ist Disposable. Dieses Interface erlaubt ein vereinfachtes Speichermanagement. So kann nach Beendigung eines Zeichenvorgangs eine Grafik aus dem Arbeitsspeicher entfernt werden, um somit Ressourcen zu sparen, und damit das Spiel flüssiger laufen zu lassen.

### Android Studio

Diese Entwicklungsumgebung vereinfacht mehrere Sachen. Zum einen kann man damit das Projekt über GitHub synchronisieren, was die Teamarbeit vereinfacht. Außerdem besitzt es nützliche Funktionen, wie zum Beispiel den automatischen Import. Es ist auch übersichtlich und hat eine automatische Vervollständigung, die das Programmieren vereinfacht.

### Gesamtüberblick

In dem folgenden Klassendiagramm sind nur die wichtigsten Klassen enthalten, auf Methoden sowie Attribute wurde aufgrund der Übersichtlichkeit verzichtet.

In dem Projekt gibt es verschiedene Ordner. Die wichtigsten sind: Actor, Player, Item, Saving, Types.

#### World

Die World regelt die Interaktion zwischen den Objekten. Die Klasse World wird in dem Projekt WorldM genannt, da World bereits als libGDX Klasse existiert. Die World speichert die Objekte, welche in der Welt platziert werden und hilft bei der Interaktion mit dem Spieler. Sie ist auch eine Schnittstelle zwischen libGDX und dem Spiel, diese initialisiert und zeichnet die Objekte. Die Welt ist auch für das Zeichnen, das Aktualisieren der Kamera und anderen Objekten in der Welt zuständig.   
Wichtige Begriffe sind die Klassen Tile (eine Kachel des Spielfelds) und Actor (ein Objekt, das in der Welt platziert wurde), auf diese Klasse wird gleich noch genauer eingegangen.

#### Actor

In dem Ordner Actor Sind alle Klassen, die von dem Spieler gebaut werden können, dazu gehören zum Beispiel: Oven, Conveyor, Miner

Die Klasse Actor vererbt an alle anderen Klassen in dem Ordner Actor(Außer FLayer). Er bildet die Wichtigste Schnittstelle zwischen der World und den anderen Actoren, über den Actor kann die World die anderen Actoren Zeichnen und die Methode update aufrufen. Diese Methode ermöglicht den Actoren zeitabhängige Aufgaben zu erledigen.

#### Player

In dem Ordner Player gibt es nur einen Klasse: PlayerController sie Ist die Schnittstelle zwischen dem Spieler und dem Programm, das heißt der Input des Spielers wird in dieser Klasse verarbeitet. In PlayerController findet auch die Berechnung für die Bewegung inklusive der Kollision statt.

#### Item

Items sind Gegenstände, die der Spieler besitzt. Is gibt verschiedene Items, die für verschieden Aktionen gebraucht werden, zum Beispiel: Herstellen eines neuen Produkts, oder eine Spitzhacke um Ressourcen schneller abbauen zu können.

#### Saving

Saving ist dazu da um Variablen und Klassen zu speichern. Genauer gesagt wird zum Beispiel die Position des Spielers gespeichert und was für Actoren sich wo befinden. Dies wird mit Hilfe von Input- und Outputstreams realisiert. Das speichern in einer datei findet in ResourceManager statt.

In SaveData befinden sich die Variablen, die gespeichert werden sollen

#### Types

Viele Typen die bereits durch libGDX und Java bereitgestellt werden. Jedoch gab es bei unserem Projekt besondere Typen die noch benötigt wurden. Der IVector2, die Abkürzung für integer Vektor zwei dimensional, diese Art des Vektors ist praktisch da dort nur ganzzahlige Positionen möglich sind, deshalb sind sie gut um eine Position in der Welt anzugeben. FMath enthält einige nützliche Berechnungen, die für dieses Projekt nötig sind, zum Beispiel um einen Vector2 in einen IVector2 umzurechenen.

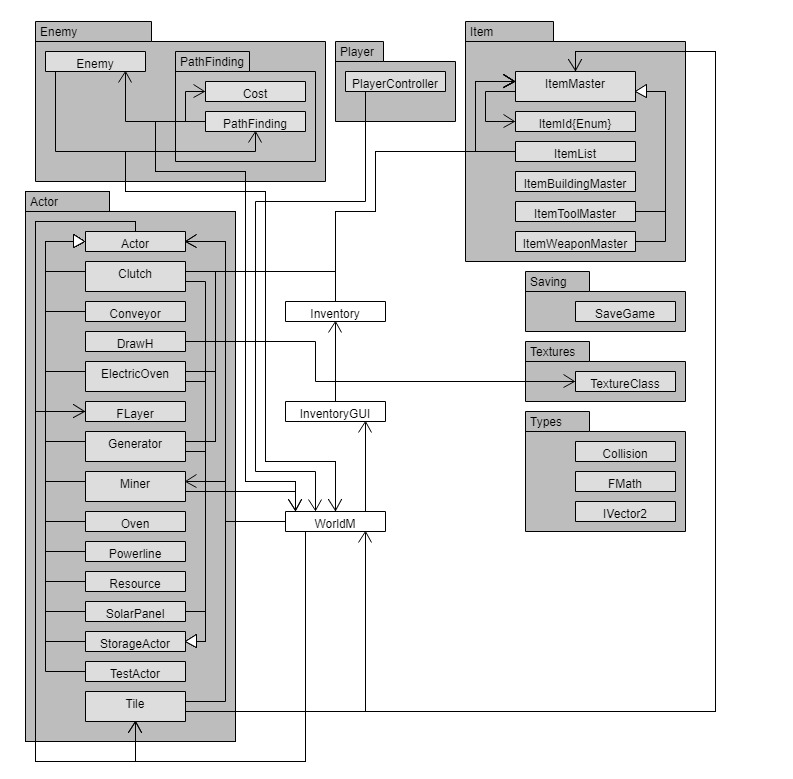


Abbildung 1: gesamtes Klassendiagramm

#### GameLoop

In der Methode render werden folgende Teile der Gameloop behandelt, diese befindet sich in der Klasse World und fungiert als Gameloop:

Die DeltaTime die für Bewegungen der Spielfiguren benutzt wird, wird hier aktualisiert. Anschließend werden die Enemies und Actoren, die ein update benötigen, bewegt und an diese wird auch die aktualisierte DeltaTime weitergegeben. Ab batch.begin() wird der batch aktualisiert und mit neuen Daten fürs Zeichnen versorgt. In zwei Schleifen wird das Tiles-Array durchgearbeitet, darin werden die Ressourcen, Actoren und Items jedes Tiles gezeichnet. Anschließend werden noch die Gegner gezeichnet.

### Übersicht Inventar

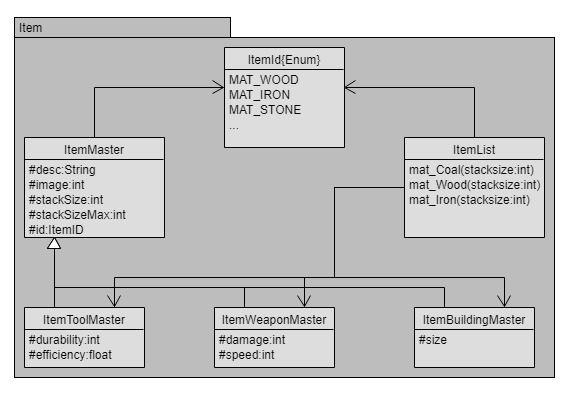


Abbildung 3: Übersicht Inventar

## World

### Grundprinzip

In WorldM wird auch die Größe der Welt festgelegt.   
In dem Attribut tiles sind die Felder in der Welt gespeichert. Die Klasse WorldM beinhaltet zusätzlich den Spieler, der durch die Klasse PlayerController repräsentiert wird.   
Es gibt außerdem das Attribut batch, das ein Spritebatch ist. Zudem gibt es noch ein Objekt der TextureClass, in der alle Texturen gespeichert werden.

Die Einheit, die benutzt wird, ist "tile".

Die Methode create erfüllt den Zweck eines Compilers, das heißt, dort werden Objekten Referenzen zugewiesen und Attribute erhalten Werte. render wird jeden Programmzyklus aufgerufen, um Objekte zu aktualisieren und zu zeichnen.

Der Enum Collision kann zwei Werte annehmen: *none* oder *collides*. *None* bedeutet, dass es keine Kollision gibt und *collides*, dass es kollidieren kann.

### Tile

Die Klasse Tile hat die Attribute actor, ressource, collision und image.   
In ressource wird die Ressource des Tiles gespeichert, wenn keine vorhanden ist, dann ist ressource ein Nullpointer.   
Image ist ein integer, er gibt die Nummer mit der das Bild assoziiert wird an.

Actor ist der Actor, der auf dem Tile platziert ist. Ein Actor kann zu einem Tile mit addActor(actor: Actor, pos: IVector2):Actor hinzugefügt werden. Dies gewährleistet, dass alle wichtigen Methoden, um den Actor hinzuzufügen aufgerufen werden.



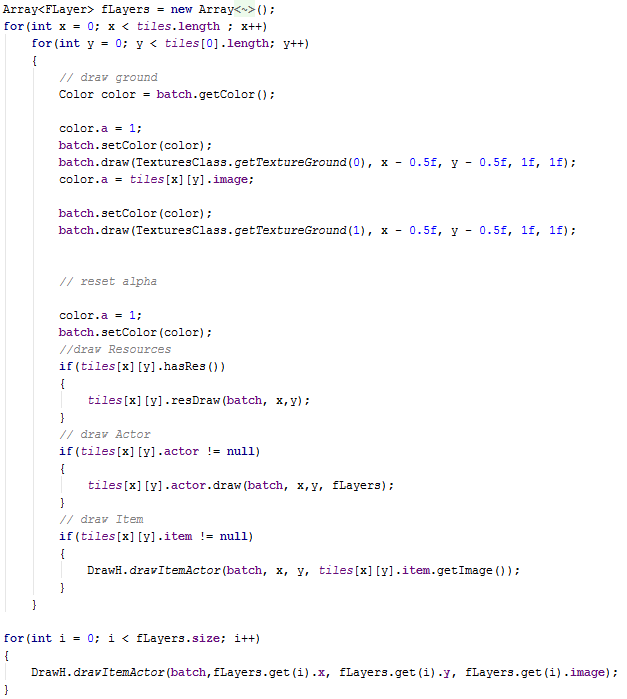
Abbildung 4: Hinzufügen eines Actors

### FLayers

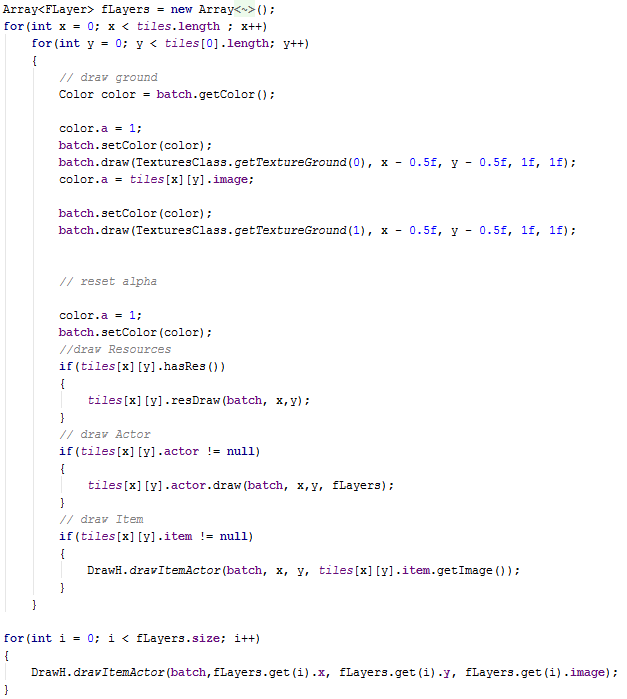
Die Klasse FLayers dient dazu, ein Item, das in der Welt liegt, zu zeichnen. Dies muss natürlich über den tiles und Actoren gezeichnet werden.   
Die Informationen über Position und das Bild werden dann in FLayers gespeichert. Erst wenn die tiles und Actoren gezeichnet wurden, werden die Items mithilfe von den Informationen in FLayer gezeichnet. Dieses Verfahren garantiert, dass ein tile nicht über dem Item gezeichnet wird. Anwendung findet dies in dem Conveyor.

Dies lässt sich mit einem Bespiel leicht erklären.

Zuerst wird ein Array vom Typ FLayers angelegt.

 Abbildung 5: Erstellen des fLayers Arrays

Danach wird fLayers allen Actoren übergeben.

 Abbildung 6: Übergeben des fLayers Array und zeichnen des Actors

In dem Actor kann ein item zu fLayers hinzugefügt werden.

   
Abbildung 7: Hinzufügen eines Items zu fLayers

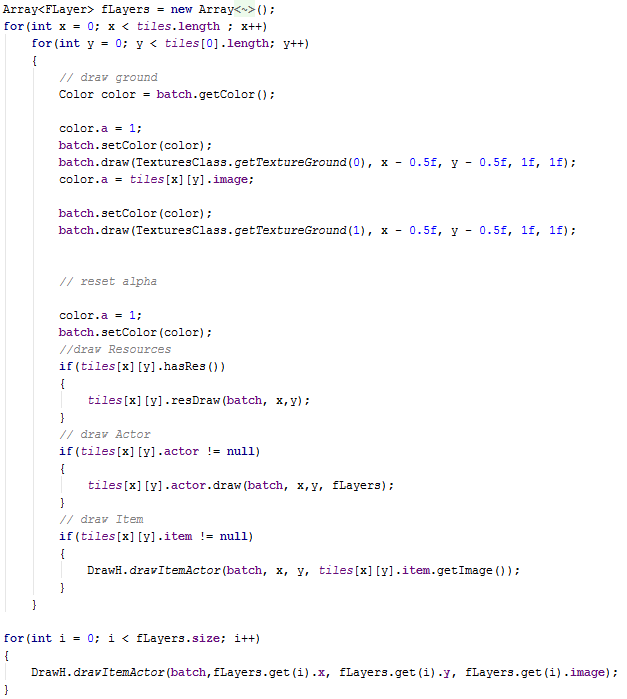
Als letztes werden alle Items gezeichnet.

Abbildung 8: Zeichnen der Items in fLayers

### Actor

Die Methode needUpdate gibt an, ob die update Methode jeden Programmzyklus aufgerufen werden soll. Die Methode führt Funktionalitäten aus, die öfter benötigt werden und zeitabhängig sind.

render wird jeden Programmzyklus aufgerufen und übernimmt das Zeichnen des Actors.

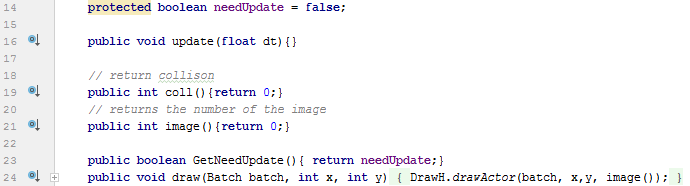


Abbildung 9: Actor Übersicht

Abbildung 10: Methode draw in Actor

Bei der Methode draw wird ein Spritebatch, die Position und ein Array übergeben.   
Die Position ist für Actoren wie das Förderband wichtig (s. 2.2.4.1). Zudem gibt es noch den Parameter fLayers, der es ermöglicht, Items die auf dem tile liegen oder für dessen Verwaltung es zuständig ist, in den Vordergrund zu zeichnen.   
Die Klasse DrawH erleichtert das Zeichnen. Die Methode draw kann überschrieben werden und ist sehr flexibel. Ist ein Programmierer mit dem System wenig vertraut, ist es jedoch fehleranfällig.

#### Conveyor

Das Förderband (Conveyor) befördert Items und ist 1x1 tiles groß. Es bewirkt, dass die Position des Items geändert wird. Die Richtung des Förderbands wird durch den Konstruktor bestimmt.   
In der Methode draw wird durch die Klasse DrawH das Item gezeichnet. Das Item verschiebt sich abhängig von progress. Das Item kann je nach richtung des Förderbandes nach oben, unten, rechts und links verschoben werden. Das Item wird nicht mit dem Batch gezeichnet sondern mit fLayers, da es über den actors und tiles gezeichnet werden soll.

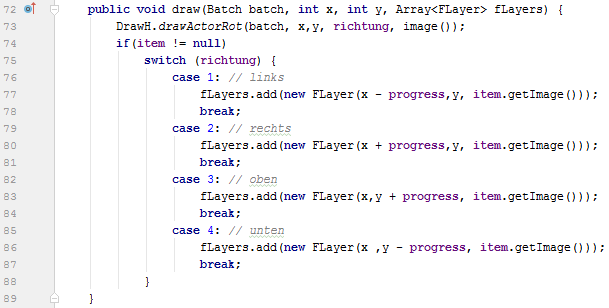


Abbildung 11: überschriebene Methode draw in Conveyor

In der Methode update wird die Deltatime dt zu progress addiert, damit progress sich erhöht.  
In der Funktion coll wird angegeben, dass das Förderband keine Kollision hat.  
In der Methode image wird die Nummer des Bildes, welches das Förderband braucht zurückgegeben.

Bei transfer wird das Item, nachdem es durch das Förderband gegangen ist, an das nächste Tile in der Laufrichtung weitergegeben.

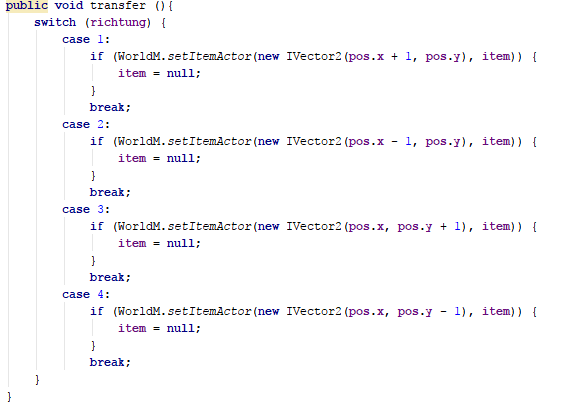


Abbildung 12: Die Methode transfer in Conveyor

#### Oven

Der Ofen (Oven) ist 1x1 tiles groß und schmilzt mithilfe von Kohle und Eisenerz reines Eisen. Dieses kann beim Crafting verwendet werden. Bei der Methode addItem wird ein Item hinzugefügt, darin wird geprüft, ob das Item die ID von Kohle hat. Wenn es der Fall ist, so wird dem Stack des Attributes coal die Anzahl der Items von item hinzugefügt. Wenn das nicht zutrifft wird geprüft, ob die ID des Attributes item mit der ID des Parameters item übereinstimmt. Falls diese übereinstimmen, wird dem Stack von dem Attribut item der Stack von dem Parameter item hinzugefügt.

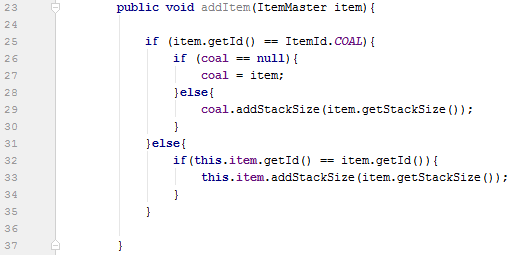


Abbildung 13: Methode addItem

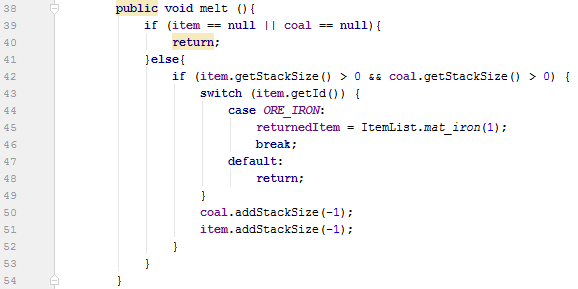
In der Methode melt wird zuerst geprüft, ob coal und item kein Objekt haben. Haben sie ein Objekt, wird in Abhängigkeit der ID von item, returnedItem das neue Objekt zugewiesen und der Stack von coal und item um 1 verkleinert. In der Methode „update“ wird die Deltatime zu progress addiert und wenn progress gleich oder größer als 100 ist, wird progress um 100 verringert und melt ausgeführt.   
  


Abbildung 14: Methode melt

In der Funktion coll wird angegeben, dass der Ofen mit dem Spieler kollidiert. In der Methode image wird die Nummer des Bildes, welches den Ofen darstellt, zurückgegeben.

#### Clutch

Ein Clutch ist ein [Greifer](https://dict.leo.org/englisch-deutsch/Greifer). Er soll ein Item von zum Beispiel einem Ofen auf ein nahegelegenes Förderband transportieren und ablegen.

Das integer richtung gibt an, in welche der vier Richtungen der Greifer das item verschiebt. Der IVector2 pos gibt die Position des Greifers an und die Anfangsposition des Items.

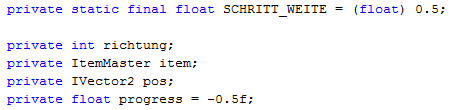


Abbildung 15: Die Attribute des Clutch

Die Werte für richtung, item und pos werden durch Parameter im Konstruktor übergeben. Die Methode needUpdate aus der Klasse Actor, von der Clutch erbt, gibt an, ob die Methode update ausgeführt werden soll.

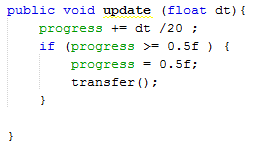
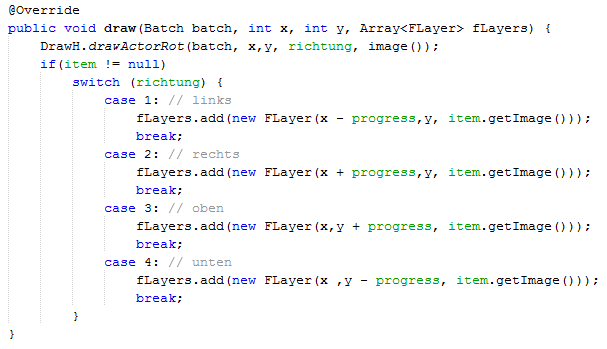
In der Methode update wird das Attribut bis zu einem bestimmten Punkt hochgezählt, um dann beim Erreichen von 0.5 Progress ein Stück zurückzusetzen und transfer auszuführen.

Abbildung 16: Die Methode update

Durch die Methode draw wird das Zeichnen realisiert. Die Methode drawActorRot zeichnet den Greifer selbst. Durch „fLayers.add(new FLayer (…));“ wird das Item gezeichnet und mit progress addiert damit eine Bewegung stattfinden kann.   
Abbildung 17: Die Methode draw

In transfer wird, je nach Richtung, die Position des Items um eins geändert, da der Greifer einen Radius von eins besitzen soll.

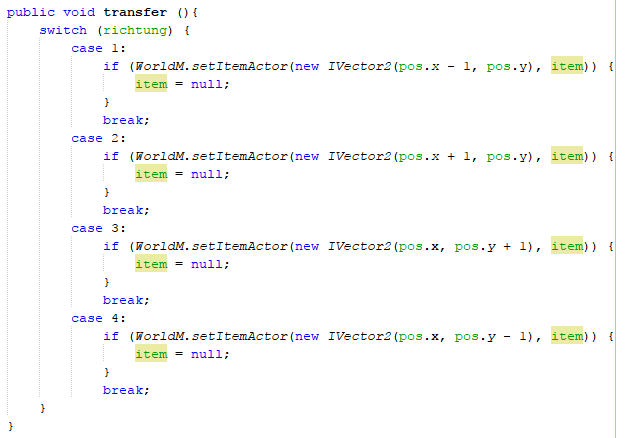


Abbildung 18 Die Methode transfer

#### ElectricOven

Der ElectricOven ist, wie der Name schon sagt, ein elektrisch betriebener Ofen. Er ist prinzipiell dem normalen Ofen sehr ähnlich, nur dass er eben seine Wärmeenergie anstatt aus Kohle aus Strom gewinnt.

Der Ofen besitzt das zu schmelzende item,  
eine eingebaute battery und das fertige Item.

Abbildung 19: Die Attribute item, battery, returnedItem

In der Methode update wird melt aufgerufen.

In melt wird das Schmelzen geregelt. Es kann nur geschmolzen werden, wenn item ein Objekt besitzt und die Anzahl von item und battery größer als null ist. Falls das der Fall ist, wird returnedItem um eins erhöht. Am Ende wird battery und die Stacksize von item um eins verringert.

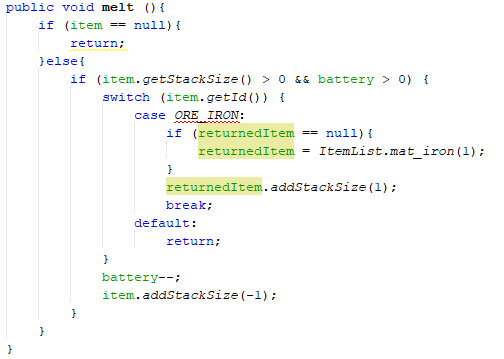
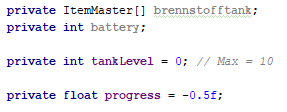


Abbildung 20: Die Methode melt

#### Generator

Der Generator soll Energie bzw. Strom produzieren, welcher als integer gespeichert wird.

Der Generator besteht aus dem Attribut Brennstofftank ,welches ein Array vom Typ ItemMaster ist, es fungiert als Speicher für den zu verbrennenden Stoff, der zur Energiegewinnung beiträgt.

Abbildung 21:Die Attribute des Generators

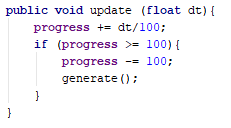
Außerdem beinhaltet der Generator einen Energiespeicher, der ein integer mit dem Namen battery ist. Er speichert die Anzahl der Ladungen, die die interne Batterie beinhaltet. tankLevel gibt an, wie voll der Speicher für den Brennstoff noch ist. Das Float progress reguliert das Generieren des Stromes in der Methode update.

Abbildung 22: Die Methode update

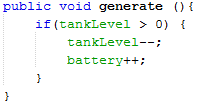
In dieser Methode wird die Zeit zwischen dem Darstellen der Frames (dt) genutzt, um in bestimmten Abständen generate aufzurufen. Diese Methode besteht aus einer If-Abfrage, die prüft, ob der Tank noch nicht leer ist und daraufhin das tankLevel um eins reduziert und zu battery eins dazu addiert wird.

Abbildung 23: Die Methode generate

#### Powerline

Die Powerline ist eine Stromleitung, sie soll Strom von einem Erzeuger, wie zum Beispiel einem Solarpanel oder Generator, zu Verbrauchern leiten.

Die Powerline soll nur als Platzhalter dienen, sie soll bloß eine grafische Funktion haben und anzeigen, welche Gegenstände miteinander verbunden sind.

Weil sie eben nur ein Platzhalter ist, hat sie als Attribut auch nur eine Position namens pos.

Abbildung 24: Das Attribut pos

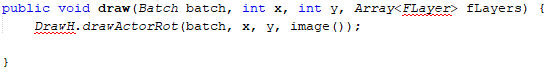
An dieser Stelle wird dann der „Strommast“ positioniert. Dies passiert grafisch in der Methode draw, mithilfe der Klasse DrawH wird die Grapfik, die in der Methode image festgelegt ist, gezeichnet.

Abbildung 25: Die Methode draw

#### Solarpanel

Das Solarpanel ist ein Stromerzeuger, der vom Prinzip her dem Generator sehr ähnlich ist, der Unterschied besteht darin, dass das Solarpanel langsamer Strom produziert, dafür aber keine Ressourcen braucht. Also ist kein Brennstoff beim Solarpanel nötig. Die Klasse beinhaltet außerdem auch einen eingebauten Energiespeicher, dieser wurde battery genannt. Er speichert die Anzahl der „Ladungen“, die das SolarPanel produziert hat. Das Attribut progress wurde bereits in anderen Klassen wie zum Beispiel in Oven oder ähnlichen Actors erklärt.

Abbildung 29: Die Attribute battery und progress

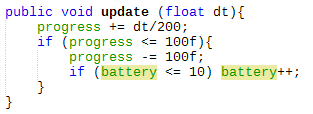
Die Methode update wurde überschrieben. Es wird geprüft, ob progress 100 erreicht. Wenn das stimmt, wird geschaut, ob die battery schon voll ist, falls dies nicht der Fall ist, wird battery um eins erhöht.

Abbildung 30: Die Methode update

#### Miner

Der Miner baut automatisch Ressourcen ab und speichert diese. Die Geschwindigkeit des Miners ist von seinem speed abhängig.

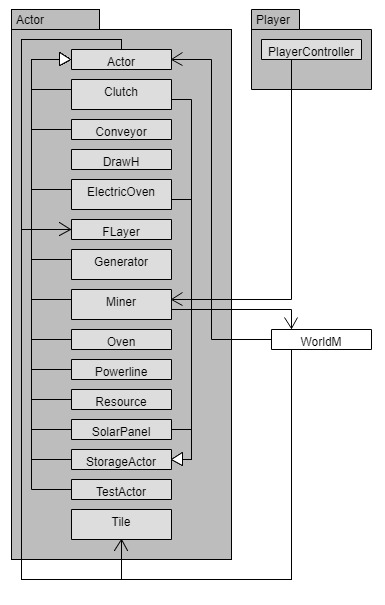


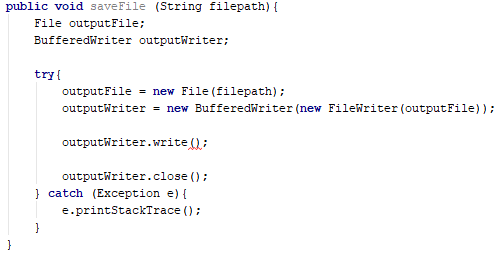
Abbildung 2: Übersicht World

### SaveGame

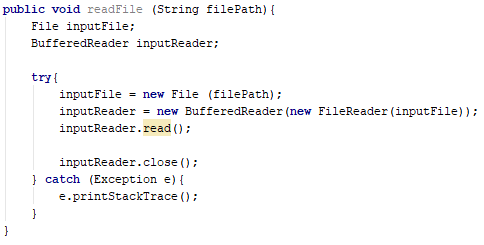
Die Klasse SaveGame soll die wichtigsten Spieldaten speichern. Diese sind erstens die Position des Spielers und die tiles aus dem Array in WorldM. Das Speichern von integeren ist nötig, weil durch diese Art des Speicherns keine komplexen Datentypen gespeichert werden können.

Abbildung 26: Das Attribut playerPos

In der Speicher-Methode saveFile wird zunächst eine Referenz von File und BufferedWriter erstellt, daraufhin wird versucht, ein Objekt von diesen zu erstellen. Dabei wird nach einer Datei gesucht, die wie der Parameter filepath heißt. Mit der Methode write aus BufferedWriter kann ein String oder ein integer gespeichert werden. Daraufhin wird der Output geschlossen.

 Abbildung 27: Die Methode saveFile

Die Lese-Methode readFile ist der Speicher-Methode sehr ähnlich, nur dass eben alles auf Input getrimmt ist, anstatt BufferedWriter gibt es nun den BufferedReader. Es wird wieder versucht, ein Objekt von File und in diesem Fall BufferedReader anzulegen. Woraufhin die Daten ausgelesen werden können.

 Abbildung 28: Die Methode readFile

### PlayerController

Der PlayerController verarbeitet den Input, die Kollision und das Equipment des Spielers.   
Er besitzt die Methode update, die wie bei Actor bei jedem Programmzyklus aufgerufen wird.

#### Input

Der Spielerinput wird im PlayerController verarbeitet. So ermöglicht zum Beispiel die Methode scrolled das Vergrößern und Verkleinern der Ansicht mit dem Drehen des Mausrades.

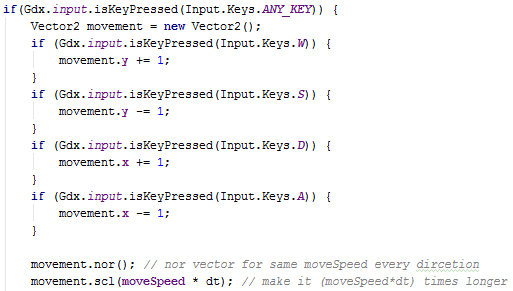


Abbildung 31: Verarbeitung des Bewegungsinputs

Dieser Codeabschnitt aus der Methode update ermöglicht es, sich zu bewegen. Zuerst wird geschaut, in welche Richtung sich der Spieler bewegen will und das in einem Vektor festgehalten. Dann wird der Vektor normiert, dadurch ist der Spieler immer gleich schnell. Ohne normieren wäre der Spieler wenn er W und A drückt schneller als wenn er nur W drückt. Zum Schluss wird der Vektor noch um moveSpeed mal deltatime skaliert. moveSpeed muss mal deltatime genommen werden, damit die zurückgelegte Strecke zeit- und nicht zyklusabhängig ist. Dies verhindert, dass verschiedene Spieler aufgrund ihrer Computer unterschiedlich schnell laufen. Danach wird noch die Kollision überprüft (s. 2.2.5.2).

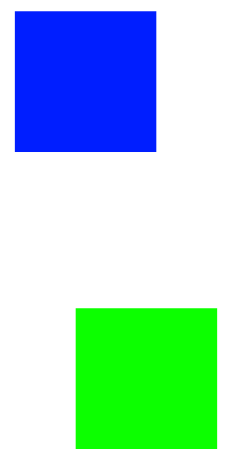
#### Kollision

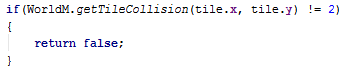
Die Kollision wird hauptsächlich von der Methode checkCollision gehandhabt.



Abbildung 32: Signatur checkCollision

Die Methode hat den Rückgabetyp Boolean, der angibt, ob der Spieler kollidiert ist. Der erste Parameter bestimmt, ob in X- oder Y-Richtung geprüft werden soll. Movement gibt die Richtung sowie die Geschwindigkeit der Bewegung an. Tile bestimmt die Position, zu der die Kollision geprüft werden soll.





Spieler

d

Tile

s1

s2

t3

t4

Abbildung 33: Prüfung tile Kollision

Zuerst muss überprüft werden, ob das Attribut collision des tile den Wert *collides* hat. Ist dies nicht der Fall, kann der Spieler nicht mit ihm kollidieren. Der Rest der Methode basiert auf einem einfachen Algorithmus. Um herauszufinden, ob der Spieler mit etwas kollidiert, muss zuerst der Abstand (d) zwischen dem Spieler und dem tile errechnet werden. Danach wird der wirkliche Abstand berechnet. Ist er größer als die Strecke, die der Spieler in diesem Programmzyklus zurücklegen würde, so gibt es keine Kollision. Im anderen Fall muss noch geprüft werden, ob auch bei der anderen Achse eine Kollision stattfindet. Dieses erfolgt dadurch, dass geschaut wird, ob s1 kleiner ist als t4 und s2 kleiner als t3. Ist dieses der Fall, gibt es eine Kollision.

Abbildung 34: Schaubild Kollision

#### Equipment

Es gibt verschiedene Arten von Equipment, zum Beispiel Waffen und Werkzeuge.   
Der Spieler bekommt durch sie bestimmte Vorteile, wie zum Beispiel eine schnellere Abbaugeschwindigkeit. Es ist möglich, das Equipment zu wechseln und so andere Vorteile zu erhalten.

## PathFinding

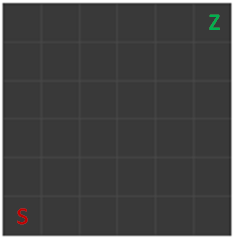
Das Pathfinding (Wegfindung) wird angewandt, um den kürzesten Weg zwischen zwei Positionen zu suchen. Ein vorbereiteter Pathfinding Algorithmus ist A\*, da er einfach zu implementieren und performant ist.

Für den A\* Algorithmus werden 2 Arrays genützt. Das erste Array speichert alle fertig errechneten Felder, es ist der fertige Array, das zweite speichert alle nicht fertig errechneten Felder, er ist der nicht fertige Array.

Das Spielfeld wird in einzelne Felder unterteilt. Ein Feld hat eine Position, einen Zeiger auf ein Feld, die Kosten und Bewegungskosten, die nachher genauer erklärt werden.

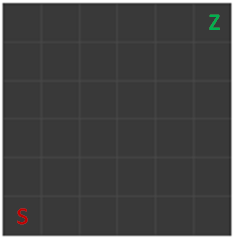
Felder in dem fertigen Array werden der Einfachheit halber als fertige Felder und Felder im nicht fertigen Array als nicht fertige Felder bezeichnet.

Alle fertigen sowie nicht fertigen Felder enthalten einen Zeiger auf das Vorgängerfeld. Damit ist es möglich den weg vom Start- zum End Feld herauszufinden. Es gibt eine Start(s)- und eine Zielposition(z).

Der Algorithmus beginnt bei dem Ziel und rechnet bis er beim Feld Start ist. Deshalb wird zuerst das Ziel zu den nicht fertigen Array hinzugefügt.

Dann wird eine Schleife ausgeführt, die immer wieder die Kosten der Felder um das nicht fertige Feld mit den geringsten Kosten, dies ist FeldX, berechnet. Felder, die dabei nicht berechnet werden, sind fertige Felder und Hindernisse(H). Zuletzt wird das berechnete Feld noch in dem fertigen Array gespeichert.

Abbildung 35: Das Spiel Feld

Die Schleife geht so lange, bis man am Start angekommen ist oder keine Felder mehr in dem nicht fertigen Array vorhanden sind.

H

4,4

9

5,8

9

3

9

2

7

6,2

8

4,8

7

3,4

7

3,8

7

6,6

7

6,2

8

5,8

9

4,4

9

4,8

7

3,8

7

3,4

7

3

9

2

7

5,2

7

2,4

7

2,4

7

2,8

7

1,4

7

1

7

1

7

6,6

7

5,2

7

4,2

7

Das Errechnen der Kosten ist etwas schwieriger. Zuerst werden die Bewegungskosten, also die Kosten, die zum Erreichen des Feldes nötig sind, errechnet. Die Bewegungskosten errechnen sich aus den Bewegungskosten des FeldX plus bei einer schrägen Bewegung 1,4 und bei einer geraden Bewegung 1. Zu den Bewegungskosten rechnet man noch den Abstand zum Start und erhält die Kosten, welche in dem Array für die nicht fertigen Felder gespeichert wird. Wenn das Feld noch keine Kosten gespeichert hat, werden die Kosten übernommen und der Zeiger auf das Vorgänger-Feld wird das FeldX. Hat das Feld bereits geringere Kosten wird nichts getan.

Abbildung 36: Der A\* Algorithmusberechnet einen Weg

## Inventar

Die Aufgabe des Inventars ist es, die vom Spieler gesammelten Ressourcen zu speichern und ihm dabei zu helfen, diese zu verwalten. Dabei soll es aber auch eine Schwierigkeit geben: Der Platz im Inventar ist begrenzt. Somit muss der Spieler sich intensiv mit der Verwaltung seines Inventarplatzes beschäftigen, um effizient mit seinen Ressourcen umgehen zu können. Ist das Inventar voll, so kann der Spieler keine weiteren Items aufnehmen. Um dem entgegenzuwirken kann der Spieler Kisten bauen, welche ein eigenes Inventar besitzen in denen der Spieler ungenutzte Items aufbewahren kann. In einer grafischen Oberfläche (GUI) sind die Items einsehbar. Die GUI enthält ebenso eine für den Spieler immer einsehbare sogenannte „Hotbar“ der ersten 10 Items im Inventar.

Die Slots werden in einem Array (aus libgdx) gespeichert. Dies erlaubt sowohl einen einfachen Zugriff auf die einzelnen Slots und hat auch weniger Fehlermöglichkeiten.   
Soll ein Item hinzugefügt werden, so muss die Methode addItem(ItemMaster item, int quantity) aufgerufen werden.

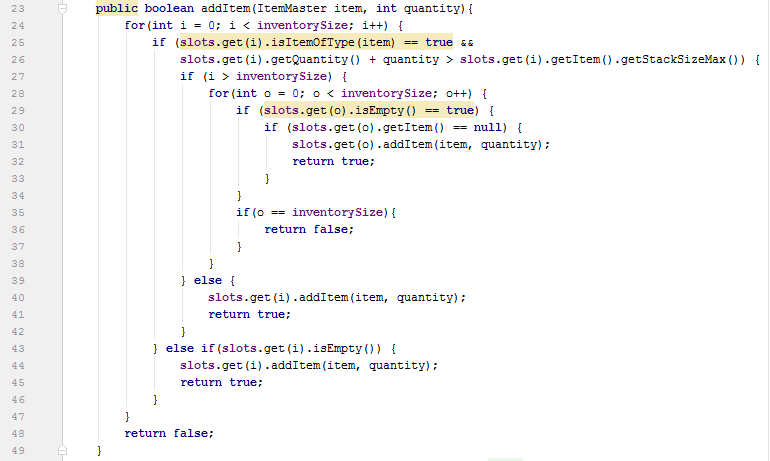
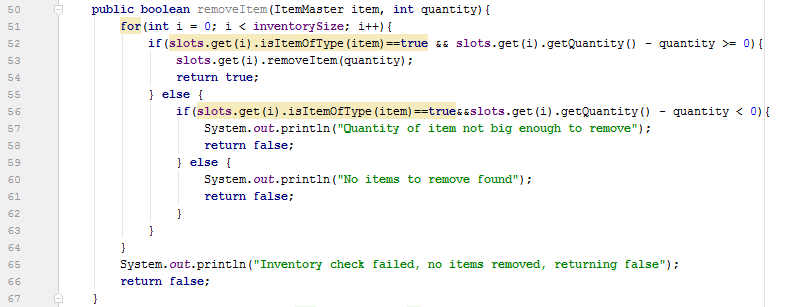


Abbildung 37: die Methode addItem(...)

Hierbei muss kein Parameter für den Slot gegeben werden, da diese Methode das Item beziehungsweise die Items automatisch in den passenden Slot sortiert. So wird das Item, zum Beispiel Kohle, zu anderer Kohle hinzugefügt, sollte diese bereits im Inventar vorhanden sein, ansonsten wird ein neuer Slot benutzt. Ist das Inventar voll, so kann das Item nicht dem Inventar hinzugefügt werden. Erfolgt das Addieren, so wird ein *true* ausgegeben.   
Die Methode removeItem(ItemMaster item, int quantity) prüft, ob zum gegebenen item genügend im Inventar vorhanden ist, um die gegebene Anzahl quantity zu entfernen.

 Abbildung 38: die Methode removeItem(...)

Ebenso wie beim Addieren wird beim Erfolg des Entfernens ein *true* ausgegeben.

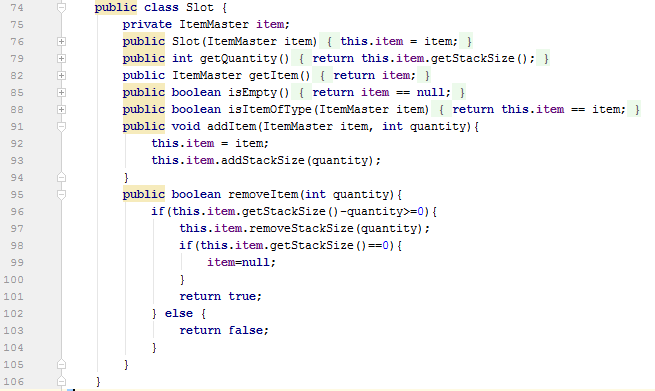
Die innere Klasse Slot in Inventory besteht nur aus dem Attribut ItemMaster item.   
Über diese Klasse funktioniert das gesamte Inventar. Im Gegensatz zum Slot der alten Klasse Inventory wird die Quantität des Items nun im ItemMaster selbst gespeichert, welches unter anderem ein Grund der Überarbeitung war. Die Methoden addItem(int quantity), removeItem(int quantity), isEmpty(), getQuantity(), isItemOfType(ItemMaster item) und getItem() stellen die Abhängigkeit des Inventars dar. Die Methoden addItem(ItemMaster item, int quantity) und removeItem(int quantity) addieren und entfernen direkt aus dem Slot das item. Dazu setzt addItem() das item zum gegebenen item und removeItem() setzt das Objekt item auf *null*, sollte die stackSize dieses items durch das Entfernen auf 0 fallen, so wird der Inhalt des Slots gelöscht, um Platz für ein neues item zu machen.

Abbildung 39: die innere Klasse Slot mit ihren Methoden

### Items

Ein Item ist ein Gegenstand, welcher vom Spieler entweder durch Abbau von Ressourcen, Crafting oder das Besiegen von Gegnern erhalten werden kann. Items werden in Unterkategorien aufgeteilt und haben je nach Unterkategorie verschiedene Aufgaben: Waffen sollen dem Spieler ermöglichen, die Gegner einfacher zu besiegen. Werkzeuge ihm helfen, Ressourcen schneller abbauen zu können. Letztendlich gibt es noch die Ressourcen, womit der Spieler Werkzeuge, Gebäude und Waffen „craften“ (herstellen) kann.  
Ein Item besteht aus einer Beschreibung, einer Textur und einer Kennnummer (ID). Spezifische Item-Kategorien, wie zum Beispiel die Waffen, besitzen weitere Attribute wie Schaden oder Haltbarkeit. Weiterhin sind diese Kategorien in eigene sogenannte „Master“-Klassen eingeteilt: ItemWeaponMaster, ItemBuildingMaster, ItemToolMaster und ItemAmmunitionMaster. Diese Klassen werden in den Spezifizierungen weiter erklärt. Alle existierenden Items sind in einer Klasse angegeben und können demnach leicht modifiziert werden.

#### Waffen

Eine Waffe ist ein Item, welche zur Verwendung Munition benötigt und Zusatzattribute wie Schaden und Schussgeschwindigkeit besitzt. Hält der Spieler eine Waffe in der Hand, so kann er, wenn Munition im Inventar ist und die Waffe in der Hotbar aktiv ist, schießen. Es gibt die Zusatzattribute damage und speed, welche repräsentativ für den Schaden und die Schussgeschwindigkeit sind.

#### Werkzeuge

Ein Werkzeug (tool) ist ein Item, welches es möglich macht, Ressourcen abzubauen, wenn es in der Hotbar aktiv ist. Die Werkzeugs-Masterklasse erweitert die Item-Masterklasse um die Attribute „durability“ und „efficiency“, welche für die Haltbarkeit und Geschwindigkeit des Werkzeugs stehen.

#### Verbrauchsstoffe

Ein Verbrauchsstoff ist ein Item, welches mit einer bestimmten Effizienz zum Betrieb von Maschinen verwendet und verbraucht wird.

#### Liste

In der Itemliste sind alle Items angegeben. Ein neues ItemMaster-Objekt wird mit Nutzung von Gettern erstellt und kann somit genutzt werden.

### Crafting

Crafting (Herstellung), ermöglicht es dem Spieler, neue Items herzustellen. Möchte der Spieler ein Item craften, wird überprüft, ob es im Inventar genügend Ressourcen für dieses Item gibt. Ist dies nicht der Fall, so kann das gewünschte Item nicht gecrafted werden und der Spieler muss weiterhin Ressourcen erlangen, bis er diesen Gegenstand herstellen kann. Damit der Spieler bessere Items craften kann, muss dieser bestimmte Craftingstationen bauen, wie zum Beispiel einen Ofen, um Eisen einzuschmelzen, welches infolgedessen zur Weiterverarbeitung in andere Gegenstände verwendet werden kann.   
Einige Craftingstationen benötigen Ressourcen, damit diese funktionieren, wie zum Beispiel Kohle. Das Crafting erfolgt in einer GUI, welche mit dem Inventar verknüpft ist und die momentan von der Craftingstation abhängigen herstellbaren Items anzeigt.

## Speichern

### Grundlegende Funktion

Ermöglicht das Speichern von Spielständen beim Schließen des Programms und ermöglicht, den vorherigen Spielstand weiterzuführen.

## Zukünftige Pläne

Das Spiel soll leichter spielbar werden. Die Grundsysteme müssen besser miteinander interagieren. Außerdem sollen Gegner in das Spiel implementiert werden, die dem Spieler eine Herausforderung bieten. Durch die Gegner wird der Spieler angetrieben, ein Verteidigungssystem aufzubauen um seine Fabrik zu schützen.

# Schlussteil

Das Spiel bietet folgende Möglichkeiten: Der Spieler kann sich bewegen und mit Actoren kollidieren. Das Pathfinding wurde programmiert und muss noch eingebunden werden. Implementiert wurden Items, ein Inventar sowie die Actoren Förderband und Ofen.   
Es ist möglich, Ressourcen abzubauen und diese im Inventar zu speichern. Alle grundlegenden Systeme sind fertig erarbeitet.   
Das Spiel sollte visuell noch ansprechender werden, die Systeme sollten besser miteinander agieren und gut implementiert werden.

Für uns selbst brachte der Seminarkurs viele Erfahrung und Wissen. Zudem wurde die Teamarbeit mit der Zeit besser. Es gab auch einige Herausforderungen, wie zum Beispiel, den Zeitplan einzuhalten, sich neues Wissen anzueignen und die Ordner zu erstellen.   
Ein wichtiges Ziel war es, den Zeitplan besser einzuhalten. Außerdem mussten wir besonders auf einen guten Informationsaustausch achten und die Aufgaben gerecht untereinander verteilen.   
Wichtig ist, dass die grundlegenden Strukturen für alle Gruppenmitglieder nützlich und verständlich sind, damit jeder effektiv seinen Beitrag leisten kann.

# Literaturverzeichnis

https://www.youtube.com/watch?v=rzBVTPaUUDg&list=PLZm85UZQLd2TPXpUJfDEdWTSgszionbJy 09.11.17 18:41

http://plugins.netbeans.org/plugin/44510/gradle-support 09.11.17 18:41

https://en.wikipedia.org/wiki/Factorio 09.11.17 19:24

http://libgdx.badlogicgames.com/documentation 16.11.17 20:50

https://de.wikipedia.org/wiki/LibGDX 12.01.18 17:15

https://www.youtube.com/watch?v=cTRLqSu50Q4&t=911s 14.01.18 13 Uhr

https://www.youtube.com/watch?v=-xW0pBZqpjU 14.01.18 13 Uhr

https://stackoverflow.com/questions/2885173/how-do-i-create-a-file-and-write-to-it-in-java 14.01.18 13 Uhr

https://www.youtube.com/watch?v=qChQrNWU9Xw 12.11.17 19:30

https://gamedev.stackexchange.com/questions/25705/change-alpha-to-a-frame-in-libgdx 12.11.17 19:30

https://stackoverflow.com/questions/17023134/libgx-rotate-a-texture-region 12.11.17 19:30

https://www.youtube.com/watch?v=-L-WgKMFuhE 16.12.17 18:00

https://de.wikipedia.org/wiki/A\*-Algorithmus 03.01.18 15:53

Bilde:

1. https://i.ytimg.com/vi/9yDZM0diiYc/maxresdefault.jpg

# Zeitplan

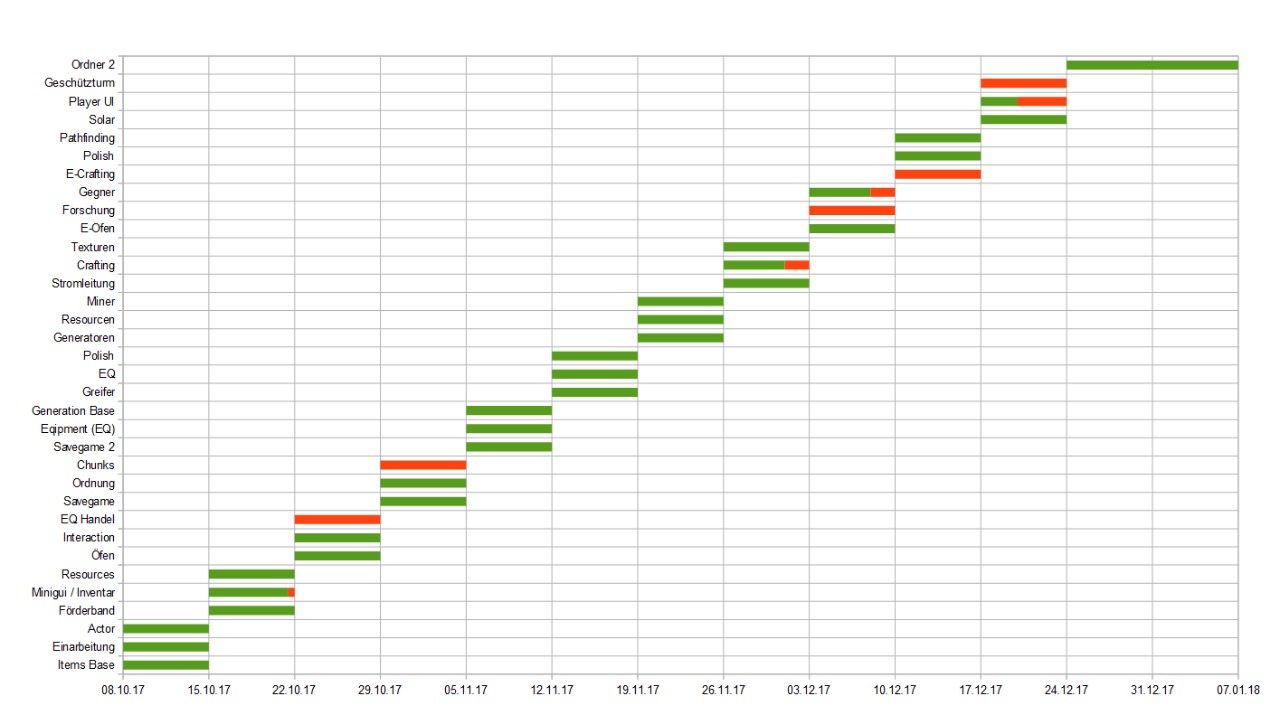


Abbildung 40: Zeitplan

# Gruppensitzungsprotokolle

## 1. Treffen

|  |  |
| --- | --- |
| **Betreuender Lehrer:** | - |
| **Zeit:**  **Ort:**  **Anwesende:**  **Gesprächsleiter:**  **Protokollant:**  **Thema/ Ziele:** | 1.9.2017  Nürtingen  Tobias Schnitzler, Robert Ryan  Tobias Schnitzler  Tobias Schnitzler  Welches Spiel?  Aufgabenverteilung |
| **Verlauf:** | **Was wurde besprochen?**  Die Verteilung der Aufgaben  Spielwahl |
| **Ergebnisse:** | **Was wurde geklärt? Begründungen?**  Grobe Aufgabenverteilung  Spiel: Factorio  **Welche Fragen blieben ungeklärt?**  - |
| **Weiterarbeit:** | **Weitere Aufgaben? Absichten und Ziele?**  Einarbeitung  **Wer übernimmt welche Aufgaben?**  Robert: GUI  Tobias: Welt |
| **Termine:** | **Bis wann sind die Aufgaben und Ziele?**  -  **Nächste Arbeitssitzung(en):**  15.9.2017 |

## 2. Treffen

|  |  |
| --- | --- |
| **Betreuender Lehrer:** | Herr Scheuermann |
| **Zeit:**  **Ort:**  **Anwesende:**  **Gesprächsleiter:**  **Protokollant:**  **Thema/ Ziele:** | 15.9.17  Kirchheim  Tobias Schnitzler, Robert Ryan  Herr Scheuermann  Robert Ryan  Neues Mitglied finden  Zeitplan |
| **Verlauf:** | **Was wurde besprochen?**  Wer wird das neue Teammitglied?  Wer macht den Zeitplan? |
| **Ergebnisse:** | **Was wurde geklärt? Begründungen?**  Zeitplan  **Welche Fragen blieben ungeklärt?**  Neues Teammitglied |
| **Weiterarbeit:** | **Weitere Aufgaben? Absichten und Ziele?**  Suchen nach neuem Teammitglied  **Wer übernimmt welche Aufgaben?**  Tobias: Zeitplan, Actorgrundsystem  Robert: Itemgrundsystem |
| **Termine:** | **Bis wann sind die Aufgaben und Ziele?**  Robert (Itemgrundsystem): 22.9.17  Tobias (Zeitplan, Actorgrundsystem): 18.9.17  **Nächste Arbeitssitzung(en):**  - |

## 3. Treffen

|  |  |
| --- | --- |
| **Betreuender Lehrer:** | - |
| **Zeit:**  **Ort:**  **Anwesende:**  **Gesprächsleiter:**  **Protokollant:**  **Thema/ Ziele:** | 22.9.17  Kirchheim  Robert Ryan, Christopher Schleppe, Tobias Schnitzler  Robert Ryan  Christopher Schleppe  Neues Teammitglied einbinden  Aufgabenbereich klären |
| **Verlauf:** | **Was wurde besprochen?**  Einarbeitung des neuen Teammitglieds |
| **Ergebnisse:** | **Was wurde geklärt? Begründungen?**  Christopher S. -> neues Teammitglied  **Welche Fragen blieben ungeklärt?**  - |
| **Weiterarbeit:** | **Weitere Aufgaben? Absichten und Ziele?**  -  **Wer übernimmt welche Aufgaben?**  Christopher Schleppe: Einarbeiten (siehe Zeitplan) |
| **Termine:** | **Bis wann sind die Aufgaben und Ziele?**  Christopher Schleppe (Einarbeitung): 3.10.17  **Nächste Arbeitssitzung(en):**  - |

## 4. Treffen

|  |  |
| --- | --- |
| **Betreuender Lehrer:** | Herr Scheuermann |
| **Zeit:**  **Ort:**  **Anwesende:**  **Gesprächsleiter:**  **Protokollant:**  **Thema/ Ziele:** | 13.10.17  Kirchheim  Robert Ryan, Christopher Schleppe, Tobias Schnitzler  Herr Scheuermann  Robert Ryan  Verschönerung des Zeitplans  Aufgabenverteilung  Motivation |
| **Verlauf:** | **Was wurde besprochen?**  Gestaltung des Zeitplans  Steigerung der Motivation |
| **Ergebnisse:** | **Was wurde geklärt? Begründungen?**  Zeitplan -> Gantt-Diagramm  **Welche Fragen blieben ungeklärt?**  - |
| **Weiterarbeit:** | **Weitere Aufgaben? Absichten und Ziele?**  -  **Wer übernimmt welche Aufgaben?**  Robert: Zeitplan verschönern |
| **Termine:** | **Bis wann sind die Aufgaben und Ziele?**  Robert (Zeitplan verschönern): 20.10.17  **Nächste Arbeitssitzung(en):**  - |

## 5. Treffen

|  |  |
| --- | --- |
| **Betreuender Lehrer:** | Herr Scheuermann |
| **Zeit:**  **Ort:**  **Anwesende:**  **Gesprächsleiter:**  **Protokollant:**  **Thema/ Ziele:** | 20.10.17  Kirchheim  Robert Ryan, Christopher Schleppe, Tobias Schnitzler  Herr Scheuermann  Christopher Schleppe  Klärung der Anforderungen für Ordner und Doku |
| **Verlauf:** | **Was wurde besprochen?**  Unterschied zwischen Ordner und Doku  Definition Präsentation und Kollegium |
| **Ergebnisse:** | **Was wurde geklärt? Begründungen?**  Ordner ist die Vorarbeit für die Doku  Präsentation ist die Vorstellung des Projektes  Kollegium ist ein Interview zum Projekt  Tutorielles als Quelle angeben  **Welche Fragen blieben ungeklärt?** |
| **Weiterarbeit:** | **Weitere Aufgaben? Absichten und Ziele?**  Struktur des Ordners nachvollziehen  Chris: Förderband bis 20.10.17 17 Uhr  Robert: Test UI/GUI verbessern  Tobias: Resourcen, Actor-system verbessern  **Wer übernimmt welche Aufgaben?**  Robert Ordner 1 Überfliegen  Tobias Ordner 2 Überfliegen  Chris Doku Überfliegen |
| **Termine:** | **Bis wann sind die Aufgaben und Ziele?**  21.10.17 16 Uhr  **Nächste Arbeitssitzung(en):**  22.10.17 19 Uhr |

## 6. Treffen

|  |  |
| --- | --- |
| **Betreuender Lehrer:** | - |
| **Zeit:**  **Ort:**  **Anwesende:**  **Gesprächsleiter:**  **Protokollant:**  **Thema/ Ziele:** | 29.10.17  Nürtingen  Robert Ryan, Christopher Schleppe, Tobias Schnitzler  Tobias Schnitzler  Christopher Schleppe  Itemlist  Inventarmanagment |
| **Verlauf:** | **Was wurde besprochen?**  Bugs in Itemlist  Gestaltung Inventarmanagment (Allgemeine Funktionsweise) |
| **Ergebnisse:** | **Was wurde geklärt? Begründungen?**  Itemlist muss mit static Methoden, anstatt mit static Attributen funktionieren.  **Welche Fragen blieben ungeklärt?**  - |
| **Weiterarbeit:** | **Weitere Aufgaben? Absichten und Ziele?**  -  **Wer übernimmt welche Aufgaben?**  Robert: Itemlist |
| **Termine:** | **Bis wann sind die Aufgaben und Ziele?**  **Nächste Arbeitssitzung(en):**  1.11.17 |

## 7.Treffen

|  |  |
| --- | --- |
| **Betreuender Lehrer:** | - |
| **Zeit:**  **Ort:**  **Anwesende:**  **Gesprächsleiter:**  **Protokollant:**  **Thema/ Ziele:** | 1.11.17  Nürtingen  Robert Ryan, Christopher Schleppe, Tobias Schnitzler  Christopher Schleppe  Christopher Schleppe  Saved Games |
| **Verlauf:** | **Was wurde besprochen?**  Speichern der Spielstände (Wer?) |
| **Ergebnisse:** | **Was wurde geklärt? Begründungen?**  Wer programmiert? -> Christopher Schleppe  **Welche Fragen blieben ungeklärt?**  - |
| **Weiterarbeit:** | **Weitere Aufgaben? Absichten und Ziele?**  -  **Wer übernimmt welche Aufgaben?**  Speichern der Spielstände -> Christopher Schleppe |
| **Termine:** | **Bis wann sind die Aufgaben und Ziele?**  5.11.17  **Nächste Arbeitssitzung(en):**  - |

## 8. Treffen

|  |  |
| --- | --- |
| **Betreuender Lehrer:** | - |
| **Zeit:**  **Ort:**  **Anwesende:**  **Gesprächsleiter:**  **Protokollant:**  **Thema/ Ziele:** | 2.11.17  Nürtingen  Robert Ryan, Christopher Schleppe, Tobias Schnitzler  Christopher Schleppe  Christopher Schleppe  Zusammenfügen  Itemactor  ItemMaster |
| **Verlauf:** | **Was wurde besprochen?**  Das Zusammenfügen von Itemactor und ItemMaster |
| **Ergebnisse:** | **Was wurde geklärt? Begründungen?**  Zusammenfügung wird umgesetzt  **Welche Fragen blieben ungeklärt?**  - |
| **Weiterarbeit:** | **Weitere Aufgaben? Absichten und Ziele?**  -  **Wer übernimmt welche Aufgaben?**  Zusammenarbeit Robert und Tobias |
| **Termine:** | **Bis wann sind die Aufgaben und Ziele?**  3.11.2017  **Nächste Arbeitssitzung(en):**  5.11.2017 |

## 9. Treffen

|  |  |
| --- | --- |
| **Betreuender Lehrer:** |  |
| **Zeit:**  **Ort:**  **Anwesende:**  **Gesprächsleiter:**  **Protokollant:**  **Thema/ Ziele:** | 5.11.17  Nürtingen  Christopher Schleppe, Tobias Schnitzler, abwesend: Robert R.  Christopher S.  Christopher S.  Fertigstellung des Ordners |
| **Verlauf:** | **Was wurde besprochen?**  Fertigstellung des Ordners |
| **Ergebnisse:** | **Was wurde geklärt? Begründungen?**  Aufgabenverteilung (jeder erledigt seinen Teil)  **Welche Fragen blieben ungeklärt?** |
| **Weiterarbeit:** | **Weitere Aufgaben? Absichten und Ziele?**  Bessere Teamarbeit  **Wer übernimmt welche Aufgaben?**  Zeitplanverbesserung: Robert |
| **Termine:** | **Bis wann sind die Aufgaben und Ziele?**  5.11.17 Abends  **Nächste Arbeitssitzung(en):**  8.11.17 mit Herr Scheuermann |

## 10. Treffen

|  |  |
| --- | --- |
| **Betreuender Lehrer:** | Herr Scheuermann |
| **Zeit:**  **Ort:**  **Anwesende:**  **Gesprächsleiter:**  **Protokollant:**  **Thema/ Ziele:** | 8.11.17  Kirchheim  Robert Ryan, Christopher Schleppe, Tobias Schnitzler  Herr Scheuermann  Christopher S.  Ordnerbesprechung |
| **Verlauf:** | **Was wurde besprochen?**  Ordner  Enum darstellen |
| **Ergebnisse:** | **Was wurde geklärt? Begründungen?**  Pfeile zu Klassen nicht zu Packages  **Welche Fragen blieben ungeklärt?** |
| **Weiterarbeit:** | **Weitere Aufgaben? Absichten und Ziele?**  In NetBeans umwandeln  **Wer übernimmt welche Aufgaben?**  Tobias: NetBeans umwandeln |
| **Termine:** | **Bis wann sind die Aufgaben und Ziele?**  10.11.17 umwandeln  **Nächste Arbeitssitzung(en):**  15.11.17 mit Herr Scheuermann |

## 11. Treffen

|  |  |
| --- | --- |
| **Betreuender Lehrer:** | Herr Scheuermann |
| **Zeit:**  **Ort:**  **Anwesende:**  **Gesprächsleiter:**  **Protokollant:**  **Thema/ Ziele:** | 15.11.17  Kirchheim  Robert Ryan, Christopher Schleppe, Tobias Schnitzler  Herr Scheuermann  Christopher S.  Ordner 1, Projekt in Netbeans |
| **Verlauf:** | **Was wurde besprochen?**  Projekt-Betrachtung |
| **Ergebnisse:** | **Was wurde geklärt? Begründungen?**  **Welche Fragen blieben ungeklärt?** |
| **Weiterarbeit:** | **Weitere Aufgaben? Absichten und Ziele?**  Freie Texturen  In pdf umwandeln  **Wer übernimmt welche Aufgaben?** |
| **Termine:** | **Bis wann sind die Aufgaben und Ziele?**  18./19.11.17  **Nächste Arbeitssitzung(en):**  17.11.17 mit Herr Scheuermann |

## 12. Treffen

|  |  |
| --- | --- |
| **Betreuender Lehrer:** |  |
| **Zeit:**  **Ort:**  **Anwesende:**  **Gesprächsleiter:**  **Protokollant:**  **Thema/ Ziele:** | 29.12.17  Nürtingen  Robert Ryan, Christopher Schleppe, Tobias Schnitzler  Tobias Schnitzler  Christopher S.  Verbesserung des Codes |
| **Verlauf:** | **Was wurde besprochen?**  Gemeinsame Verbesserung des Codes |
| **Ergebnisse:** | **Was wurde geklärt? Begründungen?**  Namensgebung Save Games  **Welche Fragen blieben ungeklärt?** |
| **Weiterarbeit:** | **Weitere Aufgaben? Absichten und Ziele?**  Save Games verbessern  **Wer übernimmt welche Aufgaben?**  Christopher: Save Games |
| **Termine:** | **Bis wann sind die Aufgaben und Ziele?**  **Nächste Arbeitssitzung(en):**  5.1.18 |

## 13. Treffen

|  |  |
| --- | --- |
| **Betreuender Lehrer:** |  |
| **Zeit:**  **Ort:**  **Anwesende:**  **Gesprächsleiter:**  **Protokollant:**  **Thema/ Ziele:** | 5.1.18  Nürtingen  Robert Ryan, Christopher Schleppe, Tobias Schnitzler  Christopher S.  Christopher S.  Ordner Nr. 2 |
| **Verlauf:** | **Was wurde besprochen?**  Mehr gute Beispiele und Bilder |
| **Ergebnisse:** | **Was wurde geklärt? Begründungen?**  Neue Texte wurden geschrieben, korrigiert und verbessert.  Code verbessert.  **Welche Fragen blieben ungeklärt?** |
| **Weiterarbeit:** | **Weitere Aufgaben? Absichten und Ziele?**  Aktualisieren von Klassendiagramm, Zeitplan  Fertigstellung der Texte  **Wer übernimmt welche Aufgaben?**  Jeder schreibt seine Texte  Robert: Klassendiagramm und Zeitplan |
| **Termine:** | **Bis wann sind die Aufgaben und Ziele?**  12.1.18  **Nächste Arbeitssitzung(en):**  10.1.18 |

## 14. Treffen

|  |  |
| --- | --- |
| **Betreuender Lehrer:** | Herr Scheuermann |
| **Zeit:**  **Ort:**  **Anwesende:**  **Gesprächsleiter:**  **Protokollant:**  **Thema/ Ziele:** | 10.1.18  Kirchheim  Robert Ryan, Christopher Schleppe, Tobias Schnitzler  Herr Scheuermann  Christopher S.  Zeitplan und Demo |
| **Verlauf:** | **Was wurde besprochen?**  Zeitplan überarbeiten  Demo bis zum nächsten Mal (17.1.18) |
| **Ergebnisse:** | **Was wurde geklärt? Begründungen?**  Aufgabenverteilung  **Welche Fragen blieben ungeklärt?** |
| **Weiterarbeit:** | **Weitere Aufgaben? Absichten und Ziele?**  Zeitplan überarbeiten  Demo Zusammenspiel Inventar und Charakter verbessern.  Ordner 2  **Wer übernimmt welche Aufgaben?**  Alle: Demo Ordner  Robert, Tobias: Zusammenspiel |
| **Termine:** | **Bis wann sind die Aufgaben und Ziele?**  17.1.18: Demo  14.1.18: Zusammenspiel  **Nächste Arbeitssitzung(en):**  12.1.18 |

## 15. Treffen

|  |  |
| --- | --- |
| **Betreuender Lehrer:** |  |
| **Zeit:**  **Ort:**  **Anwesende:**  **Gesprächsleiter:**  **Protokollant:**  **Thema/ Ziele:** | 12.1.18  Nürtingen  Robert Ryan, Christopher Schleppe, Tobias Schnitzler  Tobias Schnitzler  Christopher S.  Ordner Verbesserung |
| **Verlauf:** | **Was wurde besprochen?**  Zeitplan überarbeiten  Demo bis zum nächsten Mal (17.1.18) |
| **Ergebnisse:** | **Was wurde geklärt? Begründungen?**  Anmerkungen ESA Ordnernote  **Welche Fragen blieben ungeklärt?** |
| **Weiterarbeit:** | **Weitere Aufgaben? Absichten und Ziele?**  Demo spielbar machen  **Wer übernimmt welche Aufgaben?**  Alle: Demo Ordner |
| **Termine:** | **Bis wann sind die Aufgaben und Ziele?**  14.1.18  **Nächste Arbeitssitzung(en):**  14.1.18 |

## 16. Treffen

|  |  |
| --- | --- |
| **Betreuender Lehrer:** |  |
| **Zeit:**  **Ort:**  **Anwesende:**  **Gesprächsleiter:**  **Protokollant:**  **Thema/ Ziele:** | 14.1.18  Robert Ryan, Christopher Schleppe, Tobias Schnitzler  Tobias Schnitzler  Tobias S.  Ordner erweitern und Demo machen |
| **Verlauf:** | **Was wurde besprochen?** |
| **Ergebnisse:** | **Was wurde geklärt? Begründungen?**  **Welche Fragen blieben ungeklärt?** |
| **Weiterarbeit:** | **Weitere Aufgaben? Absichten und Ziele?**  Demo spielbar machen  **Wer übernimmt welche Aufgaben?**  Alle: Demo Ordner |
| **Termine:** | **Bis wann sind die Aufgaben und Ziele?**  14.1.18  **Nächste Arbeitssitzung(en):**  14.1.18 |

# Zeitnachweis

## Robert Ryan

|  |  |  |
| --- | --- | --- |
| Datum | Zeitaufwand in Stunden | Thema |
| 1. -2.9.17 | 10 | Einarbeitung |
| 20. - 22.9.17 | 5 | Item Base |
| 30.9. - 5.10.17 | 5 | Item Überarbeitung und Inventar anfangen |
| 9. - 15.10.17 | 3 | Inventar Fortführung |
| 17. - 24.10.17 | 3 | Inventar Verbesserung 1 |
| 24. - 30.10.17 | 3 | Inventar Verbesserung 1 |
| 03.11.17 | 9 | Erstellung des Ordners |
| 14.11.17 | 0,5 | Erstellung Klassendiagramme |
| **Summe** | 38,5 |  |

## Christopher Schleppe

|  |  |  |
| --- | --- | --- |
| Datum | Zeitaufwand in Stunden | Thema |
| 1. - 3. 10.17 | 10 | Einarbeitung |
| 10. 10.17 | 4 | Einarbeitung in das Projekt |
| 14.10.17 | 1 | Conveyor Strukturdiagramm |
| 15.10.17 | 2 | Grobe Codeerstellung |
| 21.10.17 | 3 | Verbesserung des Codes |
| 28.10.17 | 3 | Oven programmieren |
| 03.11.17 | 9 | Erstellung des Ordners 1 |
| 05.11.17 | 8 | Erstellung des Ordners 1 |
| 8.11.17 | 2 | Recherche SaveGames |
| 12.11.17 | 2 | Programmierung SaveGames |
| 19.11.17 | 2 | Greifer programmieren |
| 25.11.17 | 1 | Generator programmieren |
| 4.12.17 | 2 | Stromleitung programmieren |
| 10.12.17. | 2 | ElectricOven programmieren |
| 23.12.17 | 2 | Solarpanel programmieren |
| 5.1.18 | 4 | Erstellung Ordner 2 |
| 12.1.18 | 6 | Erstellung Ordner 2 |
| 14.1.18 | 4 | Erstellung Ordner 2 |
| **Summe** | 67 |  |

## Tobias Schnitzler

|  |  |  |
| --- | --- | --- |
| Datum | Zeitaufwand in Stunden | Thema |
| 1. - 2.9.17 | 10 | Einarbeitung |
| 17.9.17 | 4 | World Grundstruktur |
| 20.9.17 | 4 | Kollision Spieler Bewegung |
| 2.10.17 | 3 | World verbessert |
| 9.10.17 | 4 | Actor System |
| 20.10.17 | 3 | Actor System verbessert |
| 22.10.17 | 3 | Ressource system |
| 24.10.17 | 0,5 | Item Actor erstellt |
| 30.10.17 | 5 | Ressource abbauen |
| 1.11.17 | 0,5 | Ordnung des Projektes Verbessern |
| 03.11.17 | 9 | Erstellung des Ordner 1 |
| 05.11.17 | 8 | Erstellung des Ordner 1 |
| 12.11.17 | 3 | Korrekturlesen des Ordners 1 |
| 14.11.17 | 5 | Generation Base programmieren |
| 17.11.17 | 4 | Polish |
| 24.11.17 | 5 | Minen und zeichnen des Förderbands |
| 3.12.17 | 5 | Texturen erstellen |
| 10.12.17 | 2 | Gegner programmieren |
| 16.12.17 | 2 | Pathfinding Recherche |
| 20.12.17 | 6 | Pathfinding Programmierung |
| 5.1.18 | 4 | Erstellung Ordner 2 |
| 12.1.18 | 6 | Erstellung Ordner 2 |
| 14.1.18 | 4 | Erstellung Ordner 2 |
| 16.1.18 | 4 | Verbesserung des Ordners 2 |
| **Summe** | 104 |  |