language=C basicstyle= backgroundcolor= showstringspaces=false breaklines=true morecomment=[l][]%,morecomment=[s][]/**/ numbers=left

DIPLOMARBEIT

JavaChess, ChessPI AndChess

Höhere Technische Bundeslehr- und Versuchsanstalt Anichstrasse

---

Abteilung

Elektronik & Technische Informatik

Ausgefuehrt im Schuljahr 2017/18 von:

Alexander Beiser 5CHEL

Marcel Huber 5CHEL

Betreuer/Betreuerin:

Ing. MSc. Signitzer Markus

Innsbruck, am 30.10.2017

Abgabevermerk:

Datum:

Betreuer/in:

Inhaltsverzeichnis

Erklaerung der Eigenstaendigkeit der Arbeit

EIDESSTATTLICHE ERKLÄRUNG

Ich erklaere an Eides statt, dass ich die vorliegende Diplomarbeit selbstaendig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen woertlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Ort, Datum	Verfasser, Verfasserinnen Vor- und Zunamen

1 Zusammenfassung des Projektergebnisses

1.1 Kurzfassung /Abstract

Alexander Beiser und Marcel Huber entwickelten im Zuge ihrer Diplomarbeit 2017/18 ein Schachspiel, welches auf einem PC, Android-Smartphone und Raspberry PI spielbar ist. Das Schachspiel basierte auf einem bereits von ihnen geschriebenen rohen „Gerüst".
Dieses Spiel wurde mit der Programmiersprache Java entwickelt, weiteres war für den Raspberry PI ein Gehäuse zu Designen und mit einem 3D Drucker zu realisieren. Um das Spielvergnügen für den Raspberry PI auch Unterwegs zu ermöglichen wurde eine Akkusteuerung entworfen und realisiert.

1.1.1 Alexander Beiser

Alexander Beiser war für große Teile des Backends zuständig. Ein Hauptteil bestand aus der Entwicklung einer Chess Engine, also eines Zugmechanismusses welcher

speziell für die ebenso von Alexander Beiser erschoffene Künstliche Intelligenz performiert wurde. Er entwickelte auch die Akkusteuerung für den Raspberry PI und designte das Gehäuse.

1.1.2 Marcel Huber

Marcel Huber war für weite Teile des Frontend Bereichs zuständig, die anderen Schwerpunkte bestanden in der Entwicklung des Netzwerkalgorithmus und der Entwicklung des Schachspiels für Android Geräte.

1.2 Projektergebnis

Allgemeine Beschreibung, was vom Projektziel umgesetzt wurde, in einigen kurzen Sätzen. Optional Hinweise auf Erweiterungen. Gut machen sich in diesem Kapitel auch Bilder vom Gerät (HW) bzw. Screenshots (SW).

Liste aller im Pflichtenheft aufgeführten Anforderungen, die nur teilweise oder gar nicht umgesetzt wurden (mit Begründungen).

Inhaltsverzeichnis

2 Lizenz:

Das Schachprogramm wird unter der „Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License” entwickelt.
Dies räumt jeden Menschen folgende Rechte ein:

- **Teilen:** Das Programm darf frei kopiert und weiterverteilt werden.
- **Verändern:** Das Programm darf frei verändert werden. Somit dürfen natürlich Verbesserungen implementiert werden.

Allerdings muss dies unter den folgenden Bedingungen geschehen:

- **Zuschreibung:** Man muss die Namen der Entwickler entsprechend anführen und angeben, ob Veränderungen gemacht wurden.
- **Nicht kommerziell:** Das Programm darf nicht kommerziell benützt werden.
- **Gleiche Lizenz:** Sobald Veränderungen gemacht wurden, muss die original Lizenz weiter verwendet werden. Auch darf nicht von der obigen genannten Lizenz abgewichen werden.
- **Gesetzeskonform:** Das Programm darf nicht so verändert werden, dass die Nutzung illegal wird.

1 Einleitung

Alexander und Marcel sind beide begeisterte Schachspieler, womit die Entwicklung eines Schachspiels nahe liegt. Gegen Ende des 4.Jahres der HTL trafen sie die Entscheidung ein Schachspiel selber zu entwickeln und keine Diplomarbeit von einer Firma anzunehmen. Für die Entwicklung ihres Schachspiels, werden sie von „Elektrotechnik Beiser" unterstüzt. Diese Firma übernimmt etwaige anfallende kosten für die Hardwarekomponenten.
Anfang der 5.Klasse der HTL kamen die Sondierungsgespräche mit ihrem Betreuer Ing. MSc. Signitzer Markus, welcher Ihnen sagte was im Zuge dieser Diplomarbeit alles erledigt werden müsse.
Durch die Gespräche kam man zum Schluss, dass für die Diplomarbeit ein Schachspiel in Java entwickelt werden muss und dieses auf einen RaspberryPI, als auch auf Android Geräte potiert werden soll. Weiteres wird eine Akkusteuerung für den RaspberryPI entwickelt und ein Gehäuse designed und mittels Schuleigenen 3D-Drucker ausgedruckt.
Die GUI des Spiels soll mit JavaFX erstellt werden. Das Spiel soll gegen eine selbstentwickelte Künstliche Intelligenz spielbar sein, im Hot Seat Modus oder im Local Area Network.
Im Hot Seat Modus spielt man auf einem PC abwechselnt die Partien. Details werden in einem Pflichtenheft festgehalten, dieses Pflichtenhet befindet sich im Anhang.

In der Einleitung wird erklärt, wieso man sich für dieses Thema entschieden hat. (Zielsetzung und Aufgabenstellung des Gesamtprojekts, fachliches und wirtschaftliches Umfeld)

2 Vertiefende Aufgabenstellung

2.1 Alexander Beiser

Überarbeitung des Schachmattalgorithmus, Entwicklung der Zugmechanik und Entwicklung einer künstlichen Intelligenz.
Implementierung des Schachspiels auf den Raspberry-PI, gleichzeitige designen des Gehäuses für den Raspberry-PI und Entwicklung der Akkuansteuerungsschaltung.

2.2 Marcel Huber

Entwicklung der Netzwerkfähigkeit und Implementierung von Java FX. Verbesserung und Weiterentwicklung der audio- und visuellen Gestaltung. Entwicklung der

Android-App und Fertigung des Raspberry PI-Gehäuses

3 Schach, eine Erklärung

3.1 Was ist Schach?

Um den Aufbau des Programmes nachzuvollziehen zu können, sollten die Grundregeln des Schachspiels geläufig sein. Hier haben wir versucht, die wichtigsten kurz Zusammenzufassen.
Was ist Schach?
Schach ist ein strategisches Brettspiel, indem es darum geht die feindliche Seite zuschlagen. Die feindliche Seite hat asbald verloren, wann der König gefallen ist.
Der Name Schach kommt aus dem persichen „Schah" und bedeutet so viel wie König, woher der Name „königliches Spiel" stammt.
Ursprünglich wurde das Spiel vermutlich in Nordindien erfunden und kam im Zuge der islamischen Expansion, von 630 bis ca. 750, nach Europa.

3.2 Spielregeln:

Nach der 1.Erklärung was Schach ist, kommen wir zu den Spielregeln. Schach wird aufm einem 8*8 karierten Feld gespielt. Die Nummerrierung erfolgt horizontal durch das Alphabet, a bis h und vertikal durch Ziffern, 1 bis 8. Zu Beginn gibt es zwei Teams, meist Weiß und Schwarz, mit jeweils 16 Figuren. Folgende Figuren sind zu Beginn am Feld:

- 8 Bauern
- 2 Springern
- 2 Läufern
- 2 Türmen
- 1 Dame
- 1 König

Das Ende des Spiels erfolgt entweder durch Schachmatt, Aufgabe oder durch ein Remis/Patt. Schachmatt bedeutet, dass der König bedroht wird und es dem Spieler nicht mehr möglich ist den König aus dieser Position zu befreihen. Patt Möglichkeiten:

- entsteht wenn eine der Parteien keinen legalen Zug mehr ausführen kann
- Durch ein „technisches Remis", wenn außer den beiden Königen nur mehr ein Läufer oder Springer am Feld ist.

- Wenn 50 Züge lang keine Spielfigur geschlagen oder ein Bauer bewegt wurde und der am Zug befindliche Spieler das Remis verkündet.
- Wenn eine identische Stellung drei mal mit identischen Zugmöglichkeiten mindestens drei mal vorkommt, kann ein Spieler ein Remis beantragen.
- Wenn 75 Züge lang keine Figur geschlagen bzw. ein Bauer gezogen wurde oder fünmal diesselbe Stellung mit gleichen Zugmöglichkeiten entstanden ist. Hierbei wird das Remis automatisch vom Schiedsrichter/Schachspiel eingeleitet.

Nun folgen die Zugregeln:

3.2.1 Zugregel Bauer:

- Bauer darf einen Schritt nach vorne ziehen, wenn das Feld leer ist
- Befindet sich der Bauer in der Ausgangsposition und wurde noch nicht gezogen, kann er auch wahlweise zwei Schritte vorrücken.
- Der Bauer schlägt vorwärts diagonal ein Feld.
- Spezialzug: „En Passant". Dies kann er als einzige Spielfigur, wenn ein feindlicher Bauer zuvor einen Doppelschritt gemacht hat und somit den eigenen Bauern die Option nimmt, den gegnerischen Bauern anzugreifen. Falls er ausgeführt wird, ist der feindliche Bauer vernichtet und der eigene rückt diagonal ein Feld hinter den nicht mehr existierenden Bauern.
- Sobald ein Bauer die gegnerische „Grundreihe" erreicht, wird ein Bauerntausch durchgeführt. Hier muss der Bauer gegen Dame, Turm, Läufer oder einen Springer eingetauscht werden.

3.2.2 Zugregel Springer:

- Der Springer darf auf das Feld ziehen, dass zwei Felder horizontal bzw. diagonal und eines diagonal bzw. horizontal (gegengleich) versetzt ist. z.B.: Von b8 auf c6

3.2.3 Zugregel Läufer:

- Läufer dürfen diagonal, so weit wie sie wollen ziehen und schlagen. Jedoch darf er nicht über eine Figur ziehen.

3.2.4 Zugregel Turm:

- Ein Turm darf horizontal bzw. vertikal ziehen und schlagen wie weit er will, jedoch nicht über Figuren hinweg.

3.2.5 Zugregel Dame:

- Eine Dame darf horizontal, vertikal bzw. diagonal ziehen und schlagen so weit wie sie will, jedoch nicht über Figuren hinweg.

3.2.6 Zugregel König:

- Der König kann horizontal, vertikal bzw. diagonal ein Feld ziehen.
- Spezialzug: „Rochade”. Dabei wird der König entweder zwei Felder nach links, bzw. zwei Felder nach rechts bewegt. Der Turm bewegt sich dabei drei Felder nach rechts bzw. zwei Fleder nach links. König und Turm dürfen bis zu diesen Zug noch nicht bewegt worden sein, weiters darf keines der Felder über das sie ziehen, der König oder der Turm bedroht werden.

3.3 Schachmaschinen:

Seit dem es die Möglichkeit gibt einen Schachspielenden Mechanismus zu bauen, hat man dies auch getan. Zu Anfang war dies noch der „schachspielende Türke”, welcher 1769 von Wolfgang von Kempelen konstruiert wurde.
Der richtige Durchbruch geschah aber erst durch die Erfindung des Computers. Die Hardware wurde immer Leistungsfähiger, wodurch der Mensch als Gegner immer weiter in Bedrängung geriet. 1997 schlug der von IBM speziell entwickelte Schachcomputer Deep Blue, den damaligen Schachweltmeister Kasparow, wodurch die Künstliche Intelligenz in diesem Bereich offiziell den Menschen überholt hat.
Heutzutage wird gegen Schachcomputer vor allem zu Trainingszwecken gespielt. Solche Schachcomputer finden sich mittlerweile auf so ziemlich jeden Gerät, egal ob Smartphone, Tablet oder PC/Laptop. Meist sind diese Programme aber Proprietär und „closed source”. Wir entwickeln deshalb ein „open Source” Schachspiel, dass auf mehreren Devices spielbar ist.

4 Java Chess

4.1 Einführung:

Bevor mit der Dokumentation des Programmcodes begonnen werden kann, werden zuerst einige Möglichkeiten beschrieben, wie ein Schachprogramm prinzipiell programmiert werden kann.

Hierfür gibt es prinzipiell zwei Möglichkeiten:

1. Die Figuren kennen ihre Position
2. Das Brett kennt die Positionen der Figuren

Das die Figuren ihre Position kennen, klingt zuerst gar nicht so abwegig. Probleme treten aber auf, sobald Schachmatt überprüft werden soll. Hierfür muss überprüft werden ob irgendeine gegnerische Figur den König schlagen kann, wofür man aber das Objekt der Figur benötigt. Natürlich ist dies Programmiertechnisch kein Problem, dadurch entstehen aber längere Wartezeiten.

Falls das Brett die Position der Figuren kennt und diese Figuren lediglich über eine Zahlenmatrix dargestellt werden, ist das Spiel nicht nur sehr viel performanter, es ergeben sich auch große Vorteile beim entwickeln der Künstlichen Intelligenz.
Wir entschieden uns für diese Lösung.

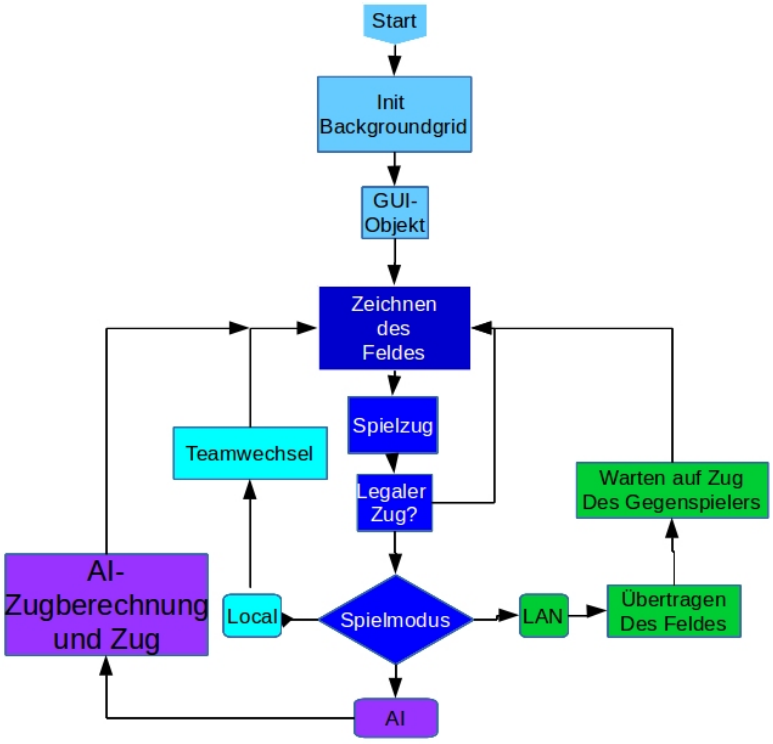
4.2 Java Chess - Übersicht

JavaChess ist in der Programmiersprache Java geschrieben. Java ist eine Objektorientierte, Klassenbasierte hochsprache der Informatik. Java hat den Vorteil, dass es nicht Hardware gebunden ist und somit ein Programm, geschrieben auf einer Distribution des Betriebssystems GNU/Linux auf (zumindest theoretisch) allen unterstüzten Systemen läuft.
Somit können wir unser Spiel auch auf einem Raspberry-PI lauffähig machen.
Das von uns verwendete GUI Environment ist JavaFX. Es wurde erstmals im Dezember 2008 den Programmierern zugänglich gemacht und soll das bis dahin Standard Java GUI Environment „Swing” ersetzen. Die Unterschiede bestehen im Aufbau, wie eine GUI realisiert werden kann bis hin zu den verbesserten grafischen Effekten, die durch JavaFX möglich sind.

Dadurch wurden die Entscheidungen gefällt Java mit JavaFX zu verwenden.
Java Chess nützt in Folge einige dieser Vorteile aus, vorallem Objektorientiertes design.

4.2.1 Blockschaltbild

Hier wird ein Einblick gegeben, wie Java Chess funktioniert. Dies geschieht Anhand von einem Blockschaltbild, welches Pakete bzw. Klassen beschreibt:



4.2.2 Initialisierung

Als Referenz bzw. Hilfe siehe 4.2.1. Zuerst startet das Programm in der Main Methode der Main Klasse. Als nächstes wird das Backgroundgrid Objekt initialisiert und das GUI-Objekt von der GUI-Klasse geladen.
Dieses Objekt ladet im Anschluss die Board Gui Klasse, welche ein Canvas ist. In

diesem ist eine gewisse Art des Zeichnens möglich. Dadurch wird auch das Schachbrett gezeichnet und in dieser Klasse findet der SSpielflussßtatt.
Im default Modus startet das Spiel im "Local"game mode, siehe 4.4. Hier spielt der Spieler zuerst einen Zug, woraufhin kontrolliert wird ob der Zug legal ist. Da das Spiel im Local-Mode startet, wechselt der Spieler und das Schachbrett wird mit der Aufstellung nach Zug 1 neu gezeichnet.

4.3 Repräsentation der Figuren:

Die Figuren werden über eine Zahlenmatrix repräsentiert. Dabei bekommt jede Figur eine individuelle Zahl zugeteilt.
Eine solche Zahl besteht aus 3-Ziffer, z.B.: 102. Diese ist der 2. weiße Bauer, die 1. Ziffer gibt dabei an, ob es Team Weiß (1) oder Schwarz (2) ist. Die 2. Ziffer gibt den Figurentyp an, also Bauer, Turm, etc. Die 3.Ziffer gibt an die wievielte Figur es ist.
Diese Matrix ist in einem Objekt von der „Background-Matrix" gespeichert.
Zu Beginn einer jeden Partie wird einmal die Startaufstellung im „Constructor" der „Background-Matrix" initialisiert:

110	120	130	140	150	131	121	111
101	102	103	104	105	106	107	108
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
201	202	203	204	205	206	207	208
210	220	230	240	250	231	221	211

Durch diese Matrix werden in Folge alle Zugberechnungen, KI-Berechnungen, etc. durchgeführt.

4.4 Zugmechanik und Local-Mode

Sobald das Spiel geladen und initialisiert ist, wird automatisch der „Local" Spielmodus ausgewählt. Dies ist der Hotseat modus, indem man auf einem Device nacheinander spielt. Dieses eigentliche Spiel geschieht in einem Objekt der „BoardGui" Klasse. Die BoardGui Klasse ist ein Canvas Objekt, also ein Objekt auf dem man zum Beispiel zeichnen kann. Diese Funktion wird ausgenutzt, um das Spielfeld zu zeichnen. Wie dies genau geschieht wird in erläutert.
Nun ist der weiße Spieler an der Reihe. Welches Team an der Reihe ist, wird durch

den Boolean „team" bestimmt. Dieser Spielstand wird in einem Objekt der Backgroundgrid Klasse gespeichert. True bedeutet das der weiße Spieler am Zug ist, false das der Schwarze am Zug ist.
Der Spieler kann nun die Figur anwählen, die er bewegen möchte, oder die linke Maustaste gedrückt halten und so die Figur über das Brett „schweben" lassen. In dieser Position werden alle möglichen Bewegungen des Spielsteins angezeigt. Hier ist zu erwähnen, dass potentielle Angriffe anders dargestellt werden, wie eine Bewegung.
Nun muss der Spieler nur noch das Feld, auf das er ziehen möchte klicken bzw. die Figur darüber absetzen. Der Move algorithmus berechnet nun ob dieser Zug auch möglich ist, falls dieser Zug erlaubt ist wird die Hintergrundmatrix entsprechend umgeschrieben, entsprechend den neuen Positionen der Figuren.
Im Local mode wird jetzt das Team gewechselt und die GUI neu gezeichnet, damit die Änderungen in der Matrix sichtbar werden.

4.4.1 Die Move Klasse - Funktion

Es gibt aber mehrere Möglichkeiten, wie eine Abfrage des erlaubten Zuges entwickelt werden kann. Unser 1.Ansatz bestand darin, dass jede Figur ihren erlaubten Zug selber überprüft. Hierbei muss klar sein, dass wir für jede Figur ein eigenes Objekt, des jeweiligen Klassentypes (z.B.: Bauer) angelegt haben. Für die Zugüberprüfung wird an die Figur die Position übergeben, wohin sie ziehen soll und das momentane Spielfeld. Ein Boolean als Rückgabwert hat dann indiziert, ob dieser Zug legal war.
Bevor eine solche Move Abfrage, aber überhaupt durchgeführt werden kann, muss diese auch aufgerufen werden und erkannt werden welcher Spielstein ausgewählt wurde. Dies geschah über eine weitere Move Klasse Der 1. Ansatz war somit nicht wirklich eine Klasse sondern auf viele Klassenverteilt. Funktioniert hat dies ohne bekannte Bugs, dadurch ist der Code aber stellenweise sehr unübersichtlich geworden. Das Spiel wurde teilweise unperformant und eine AI mit diesem Ansatz zu schreiben ist schlicht unvorstellbar.

Der 2. Ansatz bestand darin, alle Zugabfragen in einer Klasse zu implementieren. Dazu wurde dem Objekt dieser Klasse, die Koordinaten des zu bewegenden Spielsteines gegeben, wo dieser ist und wohin gezogen werden soll und das Spielfeld. Geprüft wird wieder, ob der Spielzug erlaubt ist. Auch geschieht die Abfrage, welcher Spielstein ausgewählt wurde wieder über eine extrige Klasse.
Vorteile ergeben sich aus der Übersicht, das Problem mit der Performance hat sich auch weitestegehend gelöst. Das Problem mit der KI ist aber geblieben und nun ist ein weiteres Problem dazugekommen: Wenn man auf eine Figur klickt erscheinen

alle Felder, auf die man ziehen kann. Dies ist mit dieser Implementierung der Move Klasse wiederum eine unmöglichkeit zu programmieren.

Der 3.Ansatz beschäftigt sich mit der Vorschau der möglichen Züge. Man gibt dem Objekt der Move Klasse einfach alles, was bereits im 2. Ansatz übergeben wurde. Nun wird aber ein int[][] aus möglichen Zügen zurückgegeben. Dies funktionierte ohne Probleme.
Das einzige was als Problem deklariert werden kann, ist das dadurch das „DRY" (Dont repeat yourself) Prinzip verletzt wurde. In der Move Klasse ggab es nun einmal die Abfrage, ob der Zug erlaubt ist und einmal die Abfrage welche Züge erlaubt sind. Für die KI ist es von Vorteil, wenn sie alle möglichen Züge eines Spielsteines bekommt. Es sollte aber auch klar sein welcher Spielstein zuvor auf dem Feld stand, was durch diese Methode nur indirekt möglich ist.

Der 4.Ansatz nimmt sich allen diesen Problemen an, indem den 3. Ansatz ausbaut und eine neue „MovePos" Klasse einführt. Die Move Klasse kann nun als eine Art Box verstanden werden: Man sagt der Move Methode welche Figur man ausgewählt hat und man erhält alle möglichen Züge als MovePos-ArrayList zurück. Der eigentliche Zug muss aber extern durchgeführt werden.
Das Objekt der Klasse MovePos, beinhaltet die alte Position des Spielsteins, die neue Position, die ID des Spielsteins, die ID des Feldes auf das gezogen wurde und für die Rochade bzw. den En-Passant noch zwei weitere Informationen zu den Feldern, wo und was darauf war.
Dadurch wird für die KI Berechnung, für die Zugüberprüfung und für das Anzeigen aller möglichen Züge die gleiche Basisstruktur der Zugberechnung verwendet. Der Unterschied besteht darin, dass die KI direkt die Zugberechnung aufruft währenddessen die Zugüberprüfung und das Anzeigen der möglichen Züge auf die Methode GetMove zurückgreift (überschreiben das Spielfeld).

4.4.2 Die Move Klasse - Code

Die Move Klasse beinhaltet momentan ca. 1400 Zeilen Code.
Die folgende Dokumentation erfolgt als Pseudo Code:

```
import...
...
public class Move{
   private ArrayList _HitList;
   private ArrayList _MoveList;
   private ArrayList _LastMoveList;
```

```
private boolean _bSelect;
private int _iSelect;
...
public Move(){
   _bSelect = false;
   ...
   das Standardmaessig keine Figur ausgewaeht wurde
   ...
}
...
//Methode fuer Spielerzug bzw. zum anzeigen aller moeglichen Positionen
public int[][] GetMove(Position und ID von Spielfigur, Objekt von
    Hintergrundmatrix){
   ...
   //Die Differenz zwischen zuvor ausgewhlter Figur und jetzt ausgewhlten
       Zug Feld
   iDif = iPos - _iSelect
   //Wenn die Figur bewegt werden darf
   if(_bSelect && iDif >= 50){
      //ArrayList von der Klasse MovePos
      ArrayList MoveList = getMoveMeeple(Spielfeld,Team, Position Figur)
      for(MovePos MP in MoveList){
         if(MP-Position == Gewaehlte Position){
            ...
            Ueberschreiben des alten Feldes mit den neuen Positionen
            Auch fuer alle Spezialzuege
            ...
         }

      }
   } else {
      ...
      alle moeglichen Zuege
      ...
      if(Position ist Figur){
         ArrayList MoveList = getMoveMeeple(Spielfeld,Team,Position Figur)
         for(MovePos MP in MoveList){
            if(Zug auf Leeres Feld){
               _MoveList.add(GezogenesFeld)
            }else{
               _HitList.add(GezogenesFeld)
```

```
            }
         }
      }
   }
   return GeaendertesSpielfeld
}

//Herzstueck der Move Klasse - gibt alle moeglichen Zuege zurueck
public ArrayList getMoveMeeple(Spielfeld, Position von Spielfigur){
   new ArrayList MovePos...MP
   if(Bauer){
      //Zuege
      if(einfacher Zug moeglich){
         MovePos Zug...
         ...
         MP.add(Zug)
         ...
      }
      if(zweifacher Zug moeglich){
         MovePos Zug...
         ....
         MP.add(Zug)
         ...
      }
      //Schlaege
      if(weisses Team){
         if(Schlag diagonal nach links moeglich){
            MovePos Zug...
            ....
            MP.add(Zug)
            ...
         }
         if(Schlag diagonal nach rechts moeglich){
            MovePos Zug...
            ....
            MP.add(Zug)
            ...
         }
      } else {
         if(Schlag diagonal nach links moeglich){
            MovePos Zug...
```

```
            ....
            MP.add(Zug)
            ...
         }
         if(Schlag diagonal nach rechts moeglich){
            MovePos Zug...
            ....
            MP.add(Zug)
            ...
         }
      }
      //EnPassant
      if(min. 2. Zug){
         ...
         letzerZug = getLastMove
         ...
         if(wenn feindlicher Bauer danebensteht && ID letzer Zug == id
             Bauer daneben && Im letzen Zug 2 Felder bewegt worden sind){
            MovePos Zug...
            ....
            MP.add(Zug)
            ...
         }
      }
   }else if(Turm){
      for(i=1 bis 7){
         if(Feld in X bzw. Y Richtung Ziehbar bzw. Figur schlagbar &&
             keine Figur dazwischen) {
            MovePos Zug...
            ....
            MP.add(Zug)
            ...
         }
      }

   }else if(Springer){
      if(Feld auf eine von acht Arten ziehbar / schlagbar){
         MovePos Zug...
         ....
         MP.add(Zug)
         ...
```

```
    }
  }else if(Lauefer){
    for(i=1 bis 7){
      if(Feld in eine von vier Richtungen schlagbar/ziehbar){
        MovePos Zug...
        ....
        MP.add(Zug)
        ...
      }
    }
  }else if(Dame){
    for(i=1 bis 7){
      if(Feld in eine von vier Richtungen schlagbar/ziehbar){
        MovePos Zug...
        ....
        MP.add(Zug)
        ...
      }
      if(Feld in X bzw. Y Richtung Ziehbar bzw. Figur schlagbar &&
          keine Figur dazwischen) {
        MovePos Zug...
        ....
        MP.add(Zug)
        ...
      }
    }

  }else if(Koenig){
    if(Standard Zuege moeglich){
      MovePos Zug...
      ....
      MP.add(Zug)
      ...
    }
    if(Feld 4 Felder links vom Koenig leer){
      if(Check Rochade Bedingungen-alle Felder dazwischen leer-kein
          Feld ist bedroht){
        MovePos Zug...
        ....
        MP.add(Zug)
        ...
```

```
            }
          }
          if(Feld 3 Felder rechts vom Koenig leer){
            if(Check Rochade Bedingungen-alle Felder dazwischen leer-kein
                Feld ist bedroht){
              MovePos Zug...
              ....
              MP.add(Zug)
              ...
            }
          }

      }

      return MP;
    }

    ...
    Methode Bauerntausch
    ...
    getSchach //ueberprueft ob Koenig im Schach ist (=vl. illegaler Zug)
    getSchach2 //ueberprueft ob Koenig im Schach ist (Warnung an Spieler)
    ...
    getter und setter Methoden fuer Private Variablen
}
```

4.5 LAN-Mode

4.6 AI-Mode

Wie funktioniert die AI (Artificial Intelligence) des Schachspiels?
Für die Beantwortung dieser Frage werden folgende Kapitel behandelt:

- Prinzipielle Moeglichkeiten einer AI
- Verwendete Schach-AI-Funktion
- Der MinMax-Algorithmus:
- Code

4.6.1 Prinzipielle Moeglichkeiten einer AI

Der Terminus ÄI"wird sehr oft verwendet, jedoch gibt es Unterschiede zwischen den AIs die größer nicht sein könnten. So gibt es zum Beispiel Künstliche Intelligenzen die durch Machine-Learning-Algorithmen basieren und andere, einfachere, denen ein Min-Max Prinzip zu Grunde liegt.
Das Min-Max Prinzip lässt sich aber nur bei Spielen mit perfekter Information anwenden, wie Schach eines ist. Ein Spiel mit perfekter Information bedeutet, dass jeder Spieler alles weiß, so haben im Schach immer beide Spieler das gesamte Spielfeld im Blick.

Durch diese Art des Spiels können die theoretisch besten Züge ermittelt werden, um den besten möglichen Zug zu ziehen. Dazu muss der Algorithmus alle möglichen Spielzüge analysieren, bewerten und vergleichen um zu einem Zug zu machen. Dies ist ein sehr hoher Rechenaufwand, dafür einfacher zu implementieren. Diese Art der AI haben wir verwendet, siehe 4.6.2.

Die andere Möglichkeit ist ein Machine-Learning-Algorithmus. Dieser versucht ein biologisches Gehirn nachzubauen, indem es ein Künstliches-Neurales-Netzwerk bildet. Diese Neuronen werden dahingehend trainiert, dass die AI aus einem gegebenen Satz Daten und/bzw. mit Hilfe einer Lernfunktion Rückschlüsse auf mögliche zukünftige Ereignisse schließt. Mathematisch gesehen basiert diese Art der KI auf der Wahrscheinlichkeitsrechnung.
Wichtig ist noch anzumerken, dass diese Art der KI selber lernen kann wie sie besser wird. Als Beispiel nehmen wir Schach, die KI spielt über längere Zeit gegen sich selber und muss um sich zu verbessern nicht unbedingt gegen andere Spieler, sowohl menschlich als auch maschinell antreten.
Maschinelles Lernen ist ein sehr umfangreiches Thema, womit hier nur noch gesagt sei, dass maschinelles Lernen in letzter Zeit einige Durchbrüche erlebt hat. Um noch ein konkretes Beispiel zu nennen: Im Dezember 2017 gewann der von Google entwickelte Algorithmus AlphaZero gegen die Chess-Engine „Stockfish 8”. AlphaZero basiert auf maschinellen Lernen und Stockfish 8 auf der Vorausberechnung aller möglichen Züge.

Wieso wurde die AI nicht im maschinellen Lernen verfahren entwickelt?
Dies hätte den Rahmen der Diplomarbeit gesprengt und wäre für sich allein genommen eine eigene Diplomarbeit.

4.6.2 Verwendete Schach-AI-Funktion

Wie vorher schon beschrieben verwendet die Java-Chess-AI eine Abwandlung des Min-Max Algorithmus. Dieser wird „alphaBeta" Algorithmus genannt.
Prinzipiell sucht der Algorithmus nach dem besten Spielzug. Um dies tun zu können, benötigt es einen Algorithmus welcher alle möglichen Spielzüge bis zu einer gewissen Tiefe durchsucht und den besten Spielzug in Folge herausschreibt.
Hierfür muss geklärt werden, welcher Spieler gerade die Oberhand hat. Im entwickelten Algorithmus besteht die fundamentale „Board-Evaluation" aus der Materiellen Balance. Also welcher Spieler hat mehr und bessere Figuren. Anschließend werden noch „Bauernformationen", also wenn die Bauern sich gegenseitig decken und sogenannte „Piece Square Tables" in die Kalkulation mit eingerechnet. Piece Square Tables geben an, wo die Figuren statistisch gesehen am besten stehen. Zum Beispiel stehen Türme lieber in der Mitte, als am Rand. Dadurch haben sie mehr Bewegungsfreiheit, was im Schachspiel eine der wichtigsten Strategien zum Sieg ist. Hierbei muss angemerkt werden, dass die Piece Square Tables von SchachmeisternInnen erstellt wurden, welche der Öffentlichkeit zugänglich gemacht worden sind.

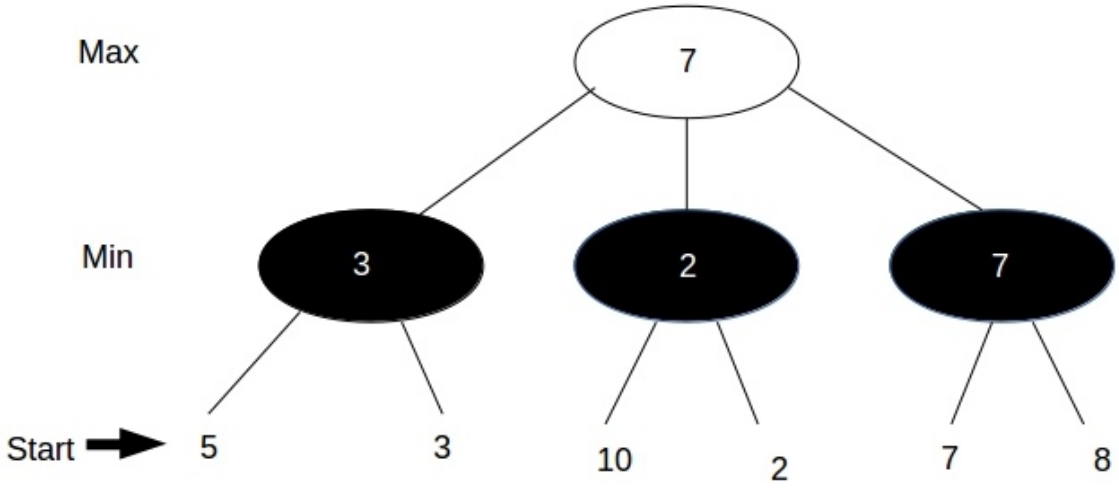
Das Durchsuchen der möglichen Spielzüge läuft folgendermaßen ab: Es wird zuerst festgelegt bis zu welcher Tiefe (z.B.: 5) gesucht werden soll. Anschließend wird der 1. mögliche Zug getätigt. Anschließend ruft sich die Methode selbst rekursiv auf, wobei die Tiefe erhöht und das Team gewechselt wird und tätigt wiederum den 1. möglichen Zug. Dies geschieht solange bis die gewünschte Tiefe (5) erreicht ist, bei welcher die „Board-Evaluation" durchgeführt wird. Dieser Wert wird zwischengespeichert und der zuletzt getätigte Zug wird rückgängig gemacht.
Nun befindet sich der Algorithmus wieder in der Tiefe 4, in welcher der 2.mögliche Zug getätigt wird. Falls dieser Zug besser ist als der vorherige, überschreibt dieser den zwischengespeicherten 1. Zug. Falls nicht, wird er von nun an nicht länger berücksichtigt.
Dies geschieht nun solange, bis alle relevanten Züge durchsucht wurden, welche nicht relevant sind wird in 4.6.4 behandelt.
Sobald der beste Zug ermittelt wurde, folgt die übliche Schachmatt-Abfrage und das Überschreiben des Schachfeldes mit den neuen Positionen. Anschließend ist wie in 4.2.1 gezeigt, wieder der Spieler an der Reihe.

Eines der größten Probleme des Algorithmus ist die Performance. Dazu ein simples Gedankenexperiment: Beim 1. Spielzug sind 20 mögliche Züge des weißen Spielers möglich und ebensoviele beim darauffolgenden Zug des schwarzen Spielers. Daraus resultiert aus den beiden Spielzügen $20 \cdot 20 = 400$ verschiedene Stellungen.

Gemittelt gibt es im Schach 28 mögliche Züge pro Spielzug. Nach 5 Zügen ergeben sich daraus schon 3.200.000 mögliche Stellungen, nach 6 wären es 64.000.000. Um zu einer Entscheidung zu kommen muss der Computer alle diese Züge analysieren und bewerten. Dies benötigt Rechenleistung, weshalb ein solcher Schachalgorithmus sehr abhängig von der verwendeten Hardware ist.
Im weiteren Sinne ist dies der Grund warum solche künstlichen Intelligenzen gegen Künstliche-Neuronale-Netzwerke verlieren (siehe 4.6.1). Die Min-Max Algorithmen können nicht ausreichend optimiert werden bzw. moderne Hardware besitzt einfach nicht genügend Rechenleistung um diesen Wettlauf standzuhalten.

4.6.3 Der MinMax-Algorithmus:

In den Kapiteln 4.6.1 und 4.6.2 wurde erwähnt, dass die JavaChess AI nach dem Min-Max Prinzip funktioniert, nur was ist dieses Prinzip?



MinMax ist nichts anderes, als das Erhalten des bestmöglichen Ergebnisses für einen Spieler, wenn die Züge des Gegenspielers miteingerechnet werden. In diesem Beispiel haben wir zwei Teams, Weiß und Schwarz. Diese sind durch die weißen Blasen erkenntlich. Weiters gilt für ein kompetitives Spiel wie Schach, dass das weiße Team immer ihr bestes Ergebnis herausholen möchte und das Schwarze auch ihres. Die Zahlen repräsentieren die Günstigkeit der Stellung für das weiße Team, wobei 10 die best mögliche Stellung ist und 1 die schlechteste. Für das schwarze Team ist dies umgekehrt, für sie ist 10 das schlechteste Ergebnis und 1 das Beste.
Somit wird das schwarze Team immer das kleinere Ergebnis nehmen, quasi das MINIMUM herausholen (=Min) und die Weißen immer das höchstmögliche, also das MAXIMUM (=Max). Daher auch der Name, MinMax.
Zurück zum Beispiel: Schwarz kann einmal wählen zwischen den Zahlen 5 & 3, zwischen 10 & 2 und zwischen 7 & 8. Da die Schwarzen immer die niedrigere Zahl

nehmen, kann die niedrigere Zahl in den schwarzen Bubble geschrieben werden (3, 2 & 7).
Somit muss sich der weiße Spieler nur noch zwischen 3 Zahlen entscheiden, bei denen er die höchste nimmt, also 7. Dies ist das best mögliche Ergebnis für das weiße Team.

4.6.4 Code

Die theoretischen Grundlagen zum Verständnis des Algorithmus sollten nun geklärt sein.

5 Fertigungsdokumentation

Die Fertigungsdokumentation ist als optionaler Dokumentationsteil zu sehen und wird mit der Betreuerin bzw. Betreuer besprochen, ob dieser Teil der Dokumentation notwendig ist. Gedacht ist die Fertigungsdokumentation speziell für aufwändige Schaltungen, bei denen es notwendig erscheint, einen Verkabelungsplan, spezielle Anleitungen für das Einlöten der Bauteile usw. auszuarbeiten.

6 Benutzerdokumentation

Hinweis: Die Benutzerdokumentation beschreibt das System aus der Sicht des Benutzers. Ein beliebiger Benutzer sollte in die Lage versetzt werden, das System zu verwenden (Bedienungsanleitung, technische Dokumentation).

6.1 Installationsanleitung

Schritt-für-Schritt-Anleitung, wie das System vom Benutzer erstmalig in Betrieb genommen werden kann. Weiters eine Anleitung, wie die Software des Systems mit Hilfe der Entwicklungswerkzeuge neu erstellt werden kann.

6.2 Anwendungsbeispiele

Beschreibung typischer Aufgaben, die der Benutzer mit dem System durchführen kann (Schritt-für-Schritt-Anleitungen).

6.3 Referenzhandbuch

Beschreibung der einzelnen Bedienungselemente (Frontplatten, Dialoge...).

6.4 Fehlermeldungen und Hinweise auf Fehlerursachen

Alle Fehlermeldungen, die das System dem Benutzer ausgeben kann, mit Beschreibung der Ursache und Vorschlägen zur Lösung des Problems.

I Abbildungsverzeichnis

→ LaTeX **erstellt mit *'\listoffigures'* Abbildungsverzeichnisse vollautomatisch**

II Tabellenverzeichnis

→ LaTeX **erstellt mit *'\listoftables'* Abbildungsverzeichnisse vollautomatisch**

III LITERATURVERZEICHNIS

Beispiele:
(Übernommen aus dem Leitfaden des BMBF Reife- und Diplomprüfungen März 2014)

1. Werke eines Autors Nachname, Vorname: Titel. Untertitel. - Verlagsort: Verlag, Jahr. Nachname, Vorname: Titel. Untertitel. Auflage - Verlagsort: Verlag, Jahr.

Beispiele:
Sandgruber, Roman: Bittersüße Genüsse. Kulturgeschichte der Genußmittel. – Wien: Böhlau, 1986. Messmer, Hans-Peter: PC-Hardwarebuch. Aufbau, Funktionsweise, Programmierung. Ein Handbuch nicht nur für Profis. 2. Aufl. - Bonn: Addison-Wesley, 1993.

2. Werke mehrerer Autoren Nachname, Vorname; Nachname, Vorname; Nachname, Vorname: Titel. Untertitel. Auflage - Verlagsort: Verlag, Jahr.

Beispiel:
Bauer, Leonhard; Matis, Herbert: Geburt der Neuzeit. Vom Feudalsystem zur Marktgesellschaft. - Mün- chen: Deutscher Taschenbuch Verlag, 1988.

3. Sammelwerke, Anthologien, CD-ROM mit Herausgeber Nachname, Vorname (Herausgeber): Titel. Untertitel. Auflage - Verlagsort: Verlag, Jahr. Nachname, Vorname: Titel. Untertitel. In: Nachname, Vorname (Herausgeber): Titel. Untertitel. Auflage - Verlagsort: Verlag, Jahr.

Beispiele:
Popp, Georg (Hg.): Die Großen der Welt. Von Echnaton bis Gutenberg. 3. Aufl. - Würzburg: Arena, 1979. Killik, John R.: Die industrielle Revolution in den Vereinigten Staaten. In: Adams, Willi Paul (Hg.): Die Vereinigten Staaten von Amerika. Fischer Weltgeschichte Bd. 30. - Frankfurt am Main: Fischer Taschenbuch Verlag, 1977. Killy, Walther (Hg.): Literatur Lexikon. Autoren u. Werke deutscher Sprache. – München: Bertelsmann, 1999. (Digitale Bibliothek, 2)

4. Mehrbändige Werke Nachname, Vorname: Titel. Bd. 3 - Verlagsort: Verlag, Jahr.

Beispiel:
Zenk, Andreas: Leitfaden für Novell NetWare. Grundlagen und Installation. Bd. 1 - Bonn: Addison Wesley, 1990.

5. Beiträge in Fachzeitschriften, Zeitungen Nachname, Vorname des Autors des bearbeiteten Artikels: Titel des Artikels. In: Titel der Zeitschrift, Heftnummer, Jahrgang, Seite (eventuell: Verlagsort, Verlag).

Beispiel:
Beck, Josef: Vorbild Gehirn. Neuronale Netze in der Anwendung. In: Chip, Nr. 7, 1993, Seite 26. - Würzburg: Vogel Verlag.

6. CD-ROM-Lexika

Beispiel:
Encarta 2000 - Microsoft 1999.

7. Internet Nachname, Vorname des Autors: Titel. Online in Internet: URL: www-Adresse, Datum. (Autor und Titel wenn vorhanden, Online in Internet: URL: www-Adresse, Datum auf jeden Fall)

Beispiel:
Ben Salah, Soia: Religiöser Fundamentalismus in Algerien. Online im Internet: URL: »http://www.hausarbeiten.de/cgi-bin/superRD.pl«, 22.11.2000. Der Weg zur Doppelmonarchie. Online in Internet: URL: http://www.parlinkom.gv.at/pd/doep/d-k1-2.htm, 22.11.2000.

8. Firmenbroschüren, CD-ROM Werden Inhalte von Firmenunterlagen verwendet, dann ist ebenfalls die Quelle anzugeben.

Beispiel:
Digitale Turbinenregler. Broschüre der Firma VOITH-HYDRO GmbH, 2012.

9. Abbildungen, Pläne Werden Abbildungen aus einer fremden Quelle [z.B. Download, Scannen) in die Diplomarbeit eingefügt, so ist unmittelbar darunter die Quelle anzugeben.

Beispiel:
Abb. 1: Digitaler Turbinenregler [ANDRITZ HYDRO]

10. Persönliche Mitteilungen

Beispiel:
Persönliche Mitteilung durch: König, Manfred: Kössler GmbH Turbinenbau am 8. März 2013.

ANHANG

6 PFLICHTENHEFT

Zur Umsetzung des Projektzieles werden messbare Kriterien formuliert.

6 .1 Funktionale Anforderungen

Die Anforderungen müssen detailliert beschrieben werden, sie müssen mess- und überprüfbar sein (nicht: "Antwort so schnell wie möglich", sondern: "Antwort in 0.5s").

Sie definieren das Systemverhalten (z. B. Erfassen und Interpretieren von Sensorwerten; Suchen von Datenbankeinträgen. . .). Die Interaktionen (Mensch-Maschine, Maschine-Maschine) sollen auch graphisch dargestellt werden.

6 .2 Schnittstellen

Technische Eigenschaften des Systems nach *außen* (Hardware- und/oder Softwareschnittstellen). Hier wird nur der Verweis auf verwendete Schnittstellen gegeben, aber nicht die Definition oder Erklärung (außer bei selbst definierten Schnittstellen).

6 .3 Abnahmekriterien

Objektive Kriterien, mit denen die Vollständigkeit und Korrektheit der fertigen Lösung geprüft werden kann (z.B.: Bei einem Signalgenerator wird das Ausgangssignal in einem definierten Frequenzbereich mit einem Oszilloskop überprüft).

Die Prototypen der einzelnen Iterationen (im Management ist Iteration eine Vorgehensweise, um mit den Ungewissheiten und Überraschungen in komplexen Situationen umzugehen) sind zu spezifizieren. Die Funktionalität der Prototypen muss durch quantifizierbare Ergebnisse überprüfbar sein. Die Spezifikation des Prototypen der nächsten Iteration ist bei Präsentation einer Iteration vorzustellen. Eine Iteration wird mit der Präsentation des Prototypen abgeschlossen (Meilenstein).

6 .4 Dokumentationsanforderungen

Anforderungen, die über das Dokument "Gesamtdokumentation_Vorlage.doc" hinausgehen (z.B.: Online-Hilfe, Dokumentation in Englisch. . .).

6 .5 Qualitätsstandards

Die im Projekt verwendeten Qualitätsstandards und einzuhaltende Normen werden festgelegt, z.B. hausinterne oder industrielle Printfertigung, verwendete Normen. . .

6 .6 Abwicklungsprozess

Für die Projektabwicklung an der Abteilung für Elektronik und Technische Informatik der HTL-Anichstrasse gelten folgende Phasen:

Startphase Phase #1	(Sommersemester 4. Klasse):
	Diskussion mit Betreuern – Erstellung Pflichtenheft
Phase #2:	Kontrolle der Meilensteine (fruehestens Oktober)
Phase #3:	Kontrolle der Meilensteine
Phase #4:	Kontrolle der Meilensteine
Phase #5:	Kontrolle der Meilensteine
Phase #6:	Kontrolle der Meilensteine
Phase #7:	04.April 2017 - Abgabe
	Präsentation und Demonstration der Ergebnisse

Die Termine für die Meilensteine werden mit der Betreuerin/dem Betreuer fixiert.

Vertiefende Aufgabenstellung laut Antrag.

Die Erstellung des Pflichtenheftes kann vor September erfolgen, es können auch Vorarbeiten in den Ferien erfolgen, jedoch dürfen keine Meilensteine definiert werden, die vor und unmittelbar nach der Startphase liegen. Die 150 – 180 Stunden sind für das Abschlussjahr definiert.

7 Zusammenfassung

Kurzbeschreibung in Deutsch, eine A4-Seite. Die Zusammenfassung soll eine Einführung in das Thema der vertiefenden Aufgabenstellung geben, den praktischen Teil kurz beschreiben und die wichtigsten Ergebnisse des einzelnen Teammitgliedes anführen. Die Zielgruppe der Zusammenfassung sind auch Nicht-Techniker!

7 .1 Schlussfolgerung / Projekterfahrung

7 .2 Projektterminplanung

Screenshots der MS Project-Datei. Die Ausgabe muss lesbar sein (eventuell auf mehrere Bilder verteilen). Insbesondere ist darauf zu achten, dass die Zeitachse und die Vorgangsachse auf jedem Bild sichtbar sind! Es muss nicht MS-Project verwendet werden!

Projektbalkenplan (Gantt-Diagramm) mit Meilensteinplan:

Abbildung 2: Projektzeitplan

7 .3 Projektpersonalplanung und Kostenplanung

Eine Abschätzung von *Personal, Material, Fremdleistungen* und der damit zusammenhängenden *Kosten* ist in der Vor- bzw. Startphase zu erstellen.

Grundsätzlich ist die Verfügbarkeit der Ressourcen zu klären. ACHTUNG: Es ist zu beachten, dass nicht immer alle notwendigen Materialien im Lager und Ressourcen (zBsp. Werkstaetten) vorhanden bzw. frei sind. Die Beschaffung der Materialien ist im Zeitplan mit zu berücksichtigen.

7 .3.1 Projektkostenplan

Kalkulation des Gesamtprojektes: Personalaufwand (Kosten laut WIR3-Unterricht), Kosten für Hard- und Software, externe Kosten (z.B.: Sensoren, Bausteine, Kabelkanäle. . .).

7 .3.2 Arbeitsnachweis Diplomarbeit

Dieser erfolgt durch ständige Aufzeichnungen der Schüler im Projekttagebuch.

Für jeden Projektmitarbeiter wird eine Tabelle gemäß Muster ausgefüllt. In dieser Aufzeichnung werden auch die Unterrichtsprojektanteile, die in die Arbeit eingeflossen sind, aufgezeigt.

Tabelle 1: Arbeitsaufstellung

Name				
Datum	Uhrzeit	Stunden nn:nn	Beschreibung	Betreuer
01.11.2004	08:00–11:30		Was wurde gemacht (eine Zeile!)	
		SUMME		

7 .3.3 Leistungscontrolling

Liefert Informationen über den Fortschritt der Projektleistungserstellung. Tabellarische Übersicht über alle Vorgänge: Welche Vorgänge wurden erfüllt, welche nicht und warum.

Abzugeben sind:

2 gebundene Dokumentationen mit Deckblatt (Format: A4)

2 CDs mit allen Unterlagen (Word, Bilder, Code. . .)

2 PowerPoint Folien im HTL Design (1. Folie: Vorstellung des Teams und die einzelnen Schwerpunkte der Kandidatinnen und Kandidaten. 2. Folie: Überblick über das Projekt mit Fotos)

Weiters ist vorzubereiten:

Ein PowerPoint Vortrag für die Präsentation und Diskussion der Diplomarbeit im HTL Design. Die Präsentation behandelt nur die Schwerpunkte der einzelnen Kandidatin und des Kandidaten. Die Teamleiterin/der Teamleiter gibt eine Gesamtübersicht des Projektes. Die Präsentation dauert maximal 8 Minuten/KandidatIn.