

Game Bot

Game Boy Robot

Bachelorthesis

Tetris ist eines der bekanntesten Spiele der Welt. Ziel dieser Arbeit ist es, einen Roboter zu entwickeln, der Tetris auf dem Game Boy spielen und punktemässig mit den weltbesten Spielern mithalten kann. Zur Verfügung stehen dem Roboter dafür eine Kamera zur Informationserfassung und mehrere Aktoren zum Drücken der Game-Boy-Tasten. Die Entwicklung der zugehörigen Software ist ein Kernstück der Arbeit und umfasst Bildverarbeitung, Bildanalyse und eine künstliche Intelligenz, die annähernd optimale Tetris-Spielzüge berechnen kann. Damit der Roboter schnell genug ist, werden alle zeitkritischen Abläufe der Hard- und Software bis ans Limit optimiert.

Studiengang: Informatik

Autoren: Patrik Marti, Mathias Winkler

Betreuer: Prof. Marcus Hudritsch

Experte: Dr. Harald Studer

Datum: 19.01.2017

Versionen

Version	Datum	Status	Bemerkungen
0.1	05.10.2016	Entwurf	Struktur des Dokuments aufgesetzt
1.0	07.01.2017	Korrekturen	Erste Version zum Korrekturlesen
1.1	13.01.2017	Korrekturen	Zweite Version zum Korrekturlesen
1.2	18.01.2017	Korrekturen	Letzte Korrekturen
1.3	19.01.2017	Definitiv	Definitive Version

Management Summary

Diese Arbeit knüpft an frühere Bemühungen an, einen Echtzeit-Tetris-Roboter zu entwickeln. Um einen fairen Vergleich zwischen Mensch und Maschine machen zu können, soll der Roboter möglichst die gleichen Bedingungen haben, wie der Mensch: Die Informationen muss der Roboter über einen optischen Sensor erfassen. Die Tastendrücke muss er mechanisch bzw. kinetisch ausführen. Gespielt wird auf dem Game Boy.

Ziel dieser Arbeit ist es, einen Roboter zu entwickeln, der Tetris auf dem Game Boy spielen kann und dem Weltmeister in einem Punktevergleich überlegen ist. Dafür muss einerseits die Hardware des Roboters konstruiert, andererseits die zugehörige Software implementiert werden. Die Software umfasst Bildverarbeitung, Bildanalyse und eine künstliche Intelligenz, die annähernd optimale Tetris-Spielzüge berechnen kann.

Damit der Roboter schnell genug ist, um in Echtzeit spielen zu können, ist die Optimierung sämtlicher zeitkritischen Abläufe der Hard- und Software nötig. Die beim Optimierungsprozess erreichten Grenzen sollen identifiziert und aufgezeigt werden.

Der Versuchsaufbau besteht aus dem Game Boy, dem Roboter (Konstruktion, Kamera, Hubmagnete als Aktoren) und einem Computer, auf dem die Software läuft.

Die Kamerabilder werden mit einer Reihe von einfachen Operatoren (Entzerrung, Binarisierung, morphologischem Opening) verarbeitet. Anhand des verarbeiteten Bildes (schwarze und weisse Blöcke) kann der Bot den neuen Spielstein bestimmen. Rauschen und ungleiche Lichtverhältnisse erschweren die Bildanalyse zusätzlich.

In Tetris kann der optimale Spielzug in vernünftiger Zeit nur annähernd berechnet werden. Dafür wird eine Breitensuche in Kombination mit einer Bewertungsfunktion eingesetzt. Das Ziel ist es, einen möglichst hohen Punktestand zu erreichen – das Spielende also lange hinauszuzögern.

Hubmagnete führen die Aktionen in Form von Tastendrücken auf dem Game Boy aus. Drehung und Verschiebung des Tetris-Spielsteins können die Aktoren parallel ausführen.

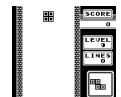
Die Autoren haben einen funktionierenden Roboter inkl. Software entwickelt. Der Roboter erreicht sogar im höchsten Level von Tetris eine beträchtliche Punktzahl. Während des Optimierungsprozesses haben sie zudem unterschiedliche technische Grenzen erreicht und ausgereizt.

Inhaltsverzeichnis

Management Summary	i
1. Einleitung	1
1.1. Motivation	1
1.2. Zielsetzung	1
1.3. Herausforderungen	2
1.4. Vorgehen	2
1.5. Aufbau	2
1.6. Abgrenzung	3
2. Grundlagen	5
2.1. Game Boy	5
2.1.1. Game Boy Advance SP	5
2.1.1.1. Hardware	5
2.1.1.2. Eingabeverzögerung	5
2.2. Tetris	6
2.2.1. Spielprinzip	6
2.2.2. Punkte	6
2.2.3. Varianten und Tetris Guideline	6
2.2.4. Komplexität und Determinismus	6
2.2.5. Multiplayer-Modus	7
2.2.6. Spielstrategien	7
2.2.6.1. Stacking	7
2.2.6.2. Überleben	8
2.2.7. Weltmeisterschaft und Rekorde	9
2.3. Robotik	9
2.3.1. Freiheitsgrade	9
2.3.2. Aktoren	9
3. Realisierung	11
3.1. Versuchsaufbau	11
3.2. Roboter	12
3.2.1. Grundgedanke	12
3.2.2. Prototypen	12
3.2.2.1. Prototyp 1	12
3.2.2.2. Prototyp 2	13
3.2.3. Aktoren	14
3.2.3.1. Tinkerforge	14
3.2.4. Elektronik	15
3.2.4.1. Schutz vor Überspannung	15
3.2.4.2. Schutz vor Überhitzung	16
3.2.4.3. Schutz vor Abgleiten	16
3.2.4.4. Wegbegrenzung	16
3.2.4.5. Reibungsreduzierung	16
3.2.5. Stückliste	17
3.2.6. Kamera	17
3.2.6.1. Kamerahalterung	17
3.2.6.2. Framerate	18

3.3. Plattform	18
3.3.1. Varianten	18
3.3.1.1. Raspberry Pi 3	18
3.3.1.2. Mini-ITX-Mainboard	18
3.3.1.3. Lenovo ThinkPad T450s	19
3.3.1.4. Lenovo ThinkPad T440p	19
3.4. Softwarearchitektur	19
3.4.1. Designziele	19
3.4.2. Angewandte Designpatterns	19
3.4.3. Testing	20
3.4.4. Module	20
3.4.5. Externe Abhangigkeiten	20
3.5. Engine	21
3.5.1. Hardware-Ansteuerung	21
3.5.2. Bildverarbeitung	22
3.5.2.1. Entzerrung	22
3.5.2.2. Binarisierung	23
3.5.2.3. Opening	23
3.5.2.4. Voraussetzungen	24
3.5.3. Emulator	24
3.6. Agent	25
3.6.1. Spiellogik	25
3.6.1.1. Levels und Geschwindigkeit	25
3.6.1.2. Zufallsgenerator	26
3.6.1.3. Effiziente Datenstrukturen	27
3.6.2. Bildanalyse	27
3.6.2.1. Blockbasierte Methode	28
3.6.2.2. Spielsteinbasierte Methode	28
3.6.2.3. Sampling	29
3.6.2.4. Verifizieren des Spielfeldes	29
3.6.3. Breitensuche	29
3.6.3.1. Bewertungsfunktion	30
3.6.3.2. Vorhersage	31
3.6.3.3. Gentischer Algorithmus	32
3.6.4. Befehlsausführung	32
3.6.4.1. Pessimistische Befehlsausführung	32
3.6.4.2. Optimistische Befehlsausführung	33
3.6.4.3. Parallele Befehlsausführung	33
3.6.5. Zustandsautomat	33
3.6.6. Zeitkomplexitt	33
4. Messungen und Ergebnisse	35
4.1. Performance der Bildverarbeitung	35
4.1.1. Messverfahren	35
4.1.2. Ergebnisse	35
4.2. Zuverlassigkeit der Bildanalyse	35
4.2.1. Messverfahren	35
4.2.2. Ergebnisse	36
4.3. Spielstrke der KI	37
4.3.1. Messverfahren	37
4.3.2. Ergebnisse	37
4.4. Zuverlassigkeit der Aktoren	38
4.4.1. Messverfahren	38
4.4.2. Ergebnisse	38
4.5. Performance der Aktoren	39
4.5.1. Messverfahren	39
4.5.2. Ergebnisse	39

4.6. Spielstärke des Roboters	40
4.6.1. Messverfahren	40
4.6.2. Ergebnisse	40
5. Diskussion	43
5.1. Mensch gegen Maschine	43
5.2. Limiten	44
5.2.1. Kamera	44
5.2.2. Bildverarbeitung und -analyse	44
5.2.3. Künstliche Intelligenz	44
5.2.4. Plattform	44
5.2.5. Akteure	45
6. Schlussfolgerungen	47
Selbständigkeitserklärung	49
Glossar	51
Literaturverzeichnis	53
Abbildungsverzeichnis	55
Tabellenverzeichnis	57
A. Projektmanagement	59
A.1. Projektmethode	59
A.1.1. Issue-Tracking	59
A.2. Projektverlauf	59
A.3. Projektorganisation	60
A.3.1. Beteiligte Personen	60
A.3.2. Projektmeetings	60
A.3.3. Arbeitsweise	60
A.4. Hilfsmittel	61
A.5. Artefakte	61
A.6. Finanzen	61
B. Korrespondenz	63
B.1. Jonas Neubauer	63
B.2. Tetris European Championship	63
B.3. Yiyuan Lee	63
C. Konstruktionspläne	65
D. Datenblätter	79
D.1. Hubmagnete	79
E. Code	81
E.1. Klassendiagramme	81
E.2. Befehle in der Applikation	82
E.3. Konfigurationsdatei	82



1. Einleitung

Tetris ist eines der bekanntesten Spiele der Welt. Jeder der es gespielt hat weiss, wie packend und frustrierend zugleich das Spiel sein kann. In der Psychologie gibt es sogar ein Syndrom, das nach dem Spiel benannt ist – der Tetris-Effekt¹. 2017 soll Tetris ausserdem verfilmt werden².

Die Autoren haben sich gefragt: Kann man Tetris auch *einem Roboter* beibringen?

Der Roboter soll aber nicht nur Tetris spielen, er soll auch die Konsole bedienen können, auf der das Spiel läuft. Die Autoren haben sich aufgrund seines hohen Bekanntheitsgrades für den Game Boy entschieden³.

Diese Arbeit ist eine Fortsetzung der Vorstudie, die im Rahmen des Moduls *Projekt 2* gemacht wurde. Es handelt sich um eine interdisziplinäre Arbeit in den Bereichen Mobile Computing/Robotics und Computer Perception/Artificial Intelligence.

Der Rahmen der Arbeit war einerseits zeitlich, andererseits finanziell vorgegeben.

1.1. Motivation

In der Vergangenheit gab es bereits mehrere Umsetzungen eines Tetris-Roboters. Colin Faheys Studie aus dem Jahr 2003 ist eine bekannte Arbeit in diesem Gebiet[24]. Sein Roboter bedient die Tasten allerdings nicht mechanisch, sondern nur elektronisch. Branislav Kisačanin hat 2010 einen Roboter gebaut, der die Tasten mittels LEGO®-Finger bedienen kann[30]. Die Tastendrücke sind jedoch sehr langsam.

Diese Arbeit knüpft an die erwähnten Bemühungen an, einen Echtzeit-Tetris-Roboter zu entwickeln. Um einen fairen Vergleich zwischen Mensch und Maschine machen zu können, soll der Roboter möglichst die gleichen Bedingungen haben, wie der Mensch: Die Informationen muss der Roboter über einen optischen Sensor erfassen. Die Tastendrücke muss er mechanisch bzw. kinetisch ausführen. Gespielt wird auf dem Game Boy.

Aufgaben an Maschinen zu übertragen liegt im Trend – selbstfahrende Autos sind nur *ein* Beispiel dafür. Nicht immer macht eine Automatisierung Sinn, manchmal kann sie auch beängstigend sein. Diese Arbeit soll demonstrieren, wie weit man in vorgegebener Zeit und mit einem beschränkten Budget eine zunächst einfache Aufgabe automatisieren kann. Unterschiedliche Disziplinen der Informatik sollen dabei zum Einsatz kommen.

Die Autoren wollten ausserdem etwas erschaffen, was nicht abstrakt und nur für Fachleute verständlich ist – jede Person soll sich unter der Arbeit etwas vorstellen können.

1.2. Zielsetzung

Ziel dieser Arbeit ist es, einen Roboter zu entwickeln, der Tetris auf dem Game Boy spielen kann und dem Weltmeister in einem Punktevergleich überlegen ist. Dafür muss einerseits die Hardware des Roboters konstruiert, andererseits die zugehörige Software implementiert werden.

Damit der Roboter schnell genug ist, um in Echtzeit spielen zu können, ist die Optimierung sämtlicher zeitkritischen Abläufe der Hard- und Software nötig. Die beim Optimierungsprozess erreichten Grenzen sollen identifiziert und aufgezeigt werden.

¹Der Effekt tritt auf, wenn Personen zu viel Zeit in eine einzige Aktivität stecken, so dass diese Aktivität anfängt, das ganze Denken, die Vorstellung, die Wahrnehmung und sogar die Träume zu formen[27].

²2017 sollen in China die Dreharbeiten für „Tetris – The Movie“ beginnen[26].

³Zu Beginn sollte der Game Boy Classic bzw. Game Boy Color verwendet werden. Später entschied man sich jedoch für den Game Boy Advance SP. Vor allem wegen dessen Hintergrundbeleuchtung (siehe Abschnitt 3.5.2) und der praktischen Geometrie. Tetris wurde neben dem Game Boy aber auch für diverse andere Plattformen herausgegeben (siehe Abschnitt 2.2).

1.3. Herausforderungen

Bei der Interaktion einer Software mit der physikalischen Welt gibt es grundsätzlich zwei mögliche Fehlerquellen: Sensoren und Aktoren. Der Roboter kann also beim Bestimmen des Tetris-Spielstandes versagen oder auch bei der Ausführung der Aktionen auf dem Game Boy.

Eine weitere Herausforderung liegt in der KI. Tetris ist *NP*-vollständig (siehe Abschnitt 2.2.4). Für das Problem, die Anzahl komplettierten Linien zu maximieren, gibt es also keine optimale und zugleich effiziente Lösung. Der Roboter muss in Echtzeit spielen und hat nur wenige Millisekunden Zeit, die Zielposition eines Spielsteins zu berechnen.

Auch die Zeit für die Ausführung der Tastendrücke ist begrenzt. Die Aktoren müssen also extrem schnell sein. Vor allem dann, wenn ein Spielstein mehrfach gedreht und über weite Distanzen verschoben werden muss.

1.4. Vorgehen

Um den Weltmeister schlagen zu können, muss der Roboter extrem schnell sein. An verschiedenen Stellen werden Engpässe auftreten, welche es zu überwinden gibt. Damit der Roboter systematisch optimiert werden kann, muss die Leistung der einzelnen Abläufe gemessen werden können. Dafür haben die Autoren verschiedenen Messgrößen definiert, die sie im Verlaufe der Arbeit immer wieder messen, um das Resultat ständig verbessern zu können. Im Folgenden sind diese Messgrößen aufgeführt:

Performance der Bildverarbeitung Wie lange dauert es, ein Kamerabild zu verarbeiten?

Messgröße: ms

Zuverlässigkeit der Bildanalyse Mit welcher Zuverlässigkeit werden die Tetris-Spielsteine erkannt?

Messgröße: %

Spielstärke der KI Wie gut ist die Tetris-KI?

Messgröße: abgebaute Linien, erzielte Punktzahl

Zuverlässigkeit der Aktoren Mit welcher Zuverlässigkeit werden Tastendrücke von den Aktoren ausgeführt?

Messgröße: %

Performance der Aktoren Wie schnell können Tastendrücke auf dem Game Boy ausgeführt werden?

Messgröße: APM = $\frac{\text{Aktionen}}{\text{Minute}}$

Spielstärke des Roboters Wie gut ist der Roboter insgesamt?

Messgröße: abgebaute Linien, erzielte Punktzahl

1.5. Aufbau

Kapitel 2 fasst die Grundlagen dieser Arbeit zusammen. Es behandelt die Hardware des Game Boys (Abschnitt 2.1), grundlegendes Wissen zum Spiel Tetris, Varianten und Spielstrategien (Abschnitt 2.2) sowie Grundlagen der Robotik (Abschnitt 2.3).

Kapitel 3 beschreibt die Methode und das Vorgehen der Autoren bei der Realisierung. Es beginnt mit dem Versuchsaufbau (Abschnitt 3.1) und ist dann unterteilt anhand der unterschiedlichen Komponenten des Roboters: Roboter (Abschnitt 3.2), Plattform (Abschnitt 3.3), Software (Abschnitt 3.4), Engine (Abschnitt 3.5) und Agent (Abschnitt 3.6).

Kapitel 4 beschreibt die Messverfahren und führt die Ergebnisse auf.

Kapitel 5 diskutiert die Ergebnisse.

Kapitel 6 enthält die Schlussfolgerungen der Autoren.

Weitere Dokumente und Informationen sind im Anhang untergebracht.



1.6. Abgrenzung

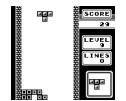
Das Ziel dieser Arbeit ist nicht ausschliesslich die Perfektionierung der KI. Vielmehr sollen alle Abläufe optimiert werden, die einen möglichen Engpass in Hard- oder Software darstellen.

In Tetris gibt es gewisse Manöver und Tricks, die vor allem fortgeschrittene Spieler anwenden⁴. Um die KI simpel zu halten, wurde auf diese Möglichkeiten bewusst verzichtet.

Der Roboter soll nicht komplett autonom sein. Er muss nicht selbstständig das Spiel starten oder in den Menüs navigieren können. Auch für die Kalibrierung der Kamera ist die Einwirkung eines Menschen vorgesehen.

Beim Roboter handelt es sich um einen Prototypen, bei dem die Funktionalität und nicht die Kompaktheit oder das Design im Vordergrund stehen. Diese Arbeit beinhaltet detaillierte Baupläne des Roboters, auf Angaben wie Toleranzen, Oberflächenbehandlung, Fertigung usw. wird jedoch nicht eingegangen. Es ist kein Ziel, den Roboter industriell fertigen zu lassen.

⁴Ein bekannter Trick ist z.B. der T-Spin: Dabei wird durch Drehen eines T-Spielsteins eine sonst unerreichbare Lücke im Spielfeld geschlossen.



2. Grundlagen

2.1. Game Boy

Der Game Boy wurde am 21. April 1989 in Japan veröffentlicht[13]. In Europa wurde das Gerät am 28. September 1990 eingeführt. Der Game Boy wurde weltweit über 118 Millionen Mal verkauft und war somit lange die meistverkaufte portable Konsole der Geschichte. Vom Game Boy gibt es sechs Generationen und zahlreiche Spezialversionen. Sämtliche Nachfolger des Game Boys sind jeweils rückwärtskompatibel mit den Spielmodulen der Vorgängerversionen.

Diese Arbeit wurde vorwiegend mit einem Game Boy Advance SP durchgeführt. Deshalb liegt der Fokus speziell auf diesem Modell.

2.1.1. Game Boy Advance SP

Der Game Boy Advance SP (siehe Abb. 2.1) war das letzte Modell der Game-Boy-Linie und erschien 2005.



Abbildung 2.1.: Game Boy Advance SP.

2.1.1.1. Hardware

Bildschirm 4,08 cm × 6,12 cm; 240 × 160 Pixel¹; TFT-Bildschirm mit max. 32'768 Farben

Prozessor 16,77 MHz, 32 Bit RISC-CPU (ARM7TDMI, ARM-Architektur); 4 oder 8 MHz, 8 Bit CISC-CPU (Z80/8080-Derivat)²

Arbeitsspeicher 32 KB I-RAM (1 cycle/32 bit) + 96 KB VRAM (1-2 cycles) + 256 KB eRAM (6 cycles/32 bit)

Framerate 59,73 Hz

2.1.1.2. Eingabeverzögerung

Über die maximale Geschwindigkeit, mit welcher der Game Boy Tastendrücke verarbeiten kann, ist wenig bekannt. Es gab aber Versuche, die Eingabeverzögerung zu messen[3]. Das Resultat war: 54 ms Verzögerung nach einem Tastendruck. Bei diesem Versuch ist jedoch unklar, ob die Verarbeitung des Inputs oder die Aktualisierung des Displays zu dieser Verzögerung führt.

¹Kann dennoch alte Spiele mit einer Auflösung von 160 × 144 Pixel wiedergeben.

²Bei alten Spielen wird nur der Z80-Prozessor verwendet.

2.2. Tetris

Das Computerspiel Tetris wurde von Alexei Paschitnow entwickelt und erschien erstmals im Jahr 1985[19]. Seither wurde das Spiel auf mehr als 65 Computerplattformen portiert[4]. Nintendo gab das Spiel 1989 für den Game Boy heraus[20]. Es ist das meistverkaufte Game-Boy-Spiel aller Zeiten[15].

2.2.1. Spielprinzip

Sequentiell werden zufällige Spielsteine – Tetriminos genannt – generiert, welche der Spieler auf einem rasterisierten Spielfeld platzieren muss. Erlaubt sind Drehungen und Verschiebungen nach links, rechts und unten. Führt der Spieler keine Aktion aus, fällt der Spielstein allmählich automatisch nach unten. Sobald der Spielstein gelandet ist, wird er fixiert und ein neuer Tetrimino wird generiert. Der Spieler kann den Turm aus Blöcken wieder abbauen, indem er Linien vervollständigt. Dafür erhält er Punkte. Kann ein Tetrimino nicht mehr generiert werden, weil dessen Startposition durch den Turm blockiert ist, verliert der Spieler.

2.2.2. Punkte

Für eine einzelne Linie erhält der Spieler 40 Punkte, für zwei Linien 100 Punkte, für drei Linien 300 Punkte und für vier Linien – ein Tetris – 1200 Punkte. Die Punkte werden noch mit dem Faktor ($\text{Level} + 1$) multipliziert[8]. Mehrere Linien gleichzeitig abzubauen wird also vom Spiel belohnt.

Ein Softdrop gibt soviele Punkte, wie die zurückgelegte Höhe des Spielsteins. Aktives und schnelles Fallenlassen eines Tetriminos wird also ebenfalls belohnt.

2.2.3. Varianten und Tetris Guideline

Es existieren viele verschiedene Varianten von Tetris[16]. Sie unterscheiden sich je nach Plattform oder sogar Version[9][8].

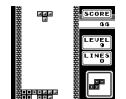
The Tetris Company hat 2001 Richtlinien herausgegeben, wie zukünftige Tetris-Spiele implementiert werden müssen, damit sie dem Standard entsprechen. Die Richtlinien definieren, wie sich das Spiel verhalten soll, welche Aktionen dem Spieler zur Verfügung stehen und auch wie das Spielfeld auszusehen hat. Viele Arbeiten, welche sich mit den Thema Tetris befassen, beziehen sich auf diese Richtlinien. Game Boy Tetris jedoch entspricht *nicht* der Tetris Guideline[7].

2.2.4. Komplexität und Determinismus

Das Problem, die Anzahl komplettierten Linien in einem Tetris-Spiel zu maximieren, ist *NP*-vollständig[23]. Dies bedeutet, dass die optimale Lösung für dieses Problem nicht in Polynomialzeit gefunden werden kann. Hinzu kommt der Zufallseinfluss bei der Generierung neuer Spielsteine. Da nicht alle nächsten Spielsteine bekannt sind, kann höchstens eine probabilistische Lösung gefunden werden.

Abgesehen von der Erzeugung neuer Spielsteine, ist Game Boy Tetris deterministisch – zumindest im A-Type-Modus³. Dies ist ein Vorteil für einen Bot, da er weniger Informationen über den Spielstand ermitteln muss, je deterministischer das Spiel ist.

³Die Game-Boy-Variante von Tetris unterscheidet zwei Spielmodi: A-Type und B-Type. Im Modus B-Type wird das Spielfeld zu Beginn zufällig mit Blöcken gefüllt, während es im Modus A-Type anfangs leer ist.



2.2.5. Multiplayer-Modus

Im Multiplayer-Modus spielen zwei Spieler – per Game Link Cable miteinander verbunden – gegeneinander⁴. Ziel ist es, länger im Spiel zu überleben als der Gegner. Die zusätzliche Schwierigkeit in diesem Modus besteht darin, dass man vom Gegner abgebaute Linien am unteren Ende des eigenen Spielfeldes angefügt bekommt (siehe rote Blöcke in Abb. 2.2).

Baut man nur eine Linie ab, wird dem Gegner keine Linie angefügt. Bei zwei Linien wird ihm eine angefügt, bei drei Linien werden ihm zwei angefügt und bei vier Linien werden dem Gegner auch vier Linien angefügt. Auch hier wird das Abbauen mehrerer Linien gleichzeitig vom Spiel belohnt. Die zufällig positionierte Lücke von der Breite eines Blocks ermöglicht es, die vom Gegner übertragenen Linien wieder abzubauen.

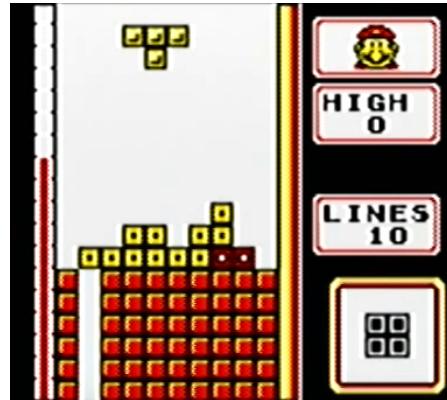


Abbildung 2.2.: Tetris Multiplayer-Modus mit Linien des Gegners am unteren Ende des Spielfeldes.

2.2.6. Spielstrategien

Abhängig von der Variante des Tetris-Spiels variiert auch die optimale Spielstrategie⁵. Die hier behandelten Strategien gelten primär für Tetris-Spiele, die nach der Tetris Guideline entwickelt wurden, können aber auch für den A-Type-Modus von Game Boy Tetris angewendet werden. In Game Boy Tetris kann kein Tetrimino zurückgestellt werden und es ist jeweils nur ein Spielstein im Vorschaufenster sichtbar.

Mit der Game-Boy-Variante kann man grundsätzlich zwei Ziele verfolgen:

- Punktzahl maximieren
- Anzahl abgebauter Linien maximieren

Allgemein sind folgende Verhaltensweisen strategisch sinnvoll:

- Höhe des Turms aus Blöcken tiefer als die Maximalhöhe halten
- Der Turm soll möglichst keine Löcher aufweisen
- Der Turm sollte eine relativ flache Oberfläche haben, jedoch nicht komplett flach[25]

2.2.6.1. Stacking

Beim Stacking geht es darum, in vorgegebener Zeit möglichst viele Punkte zu erzielen. Dazu wird ein Turm aufgebaut, welcher mindestens vier Linien Höhe hat und eine freie Spalte für ein I-Tetrimino aufweist (siehe Abb. 2.3). Sobald ein I-Tetrimino erscheint, werden vier Linien gleichzeitig abgebaut. Dies garantiert einen maximalen Punktertrag (siehe Abschnitt 2.2.2). Die Lücke sollte jeweils ganz links oder ganz rechts freigehalten werden, da sich so mehr Optionen beim nächsten Spielstein ergeben[6].

⁴Zumindest existiert dieser Modus in Game Boy Tetris.

⁵In einigen Varianten des Spiels kann ein Tetrimino zurückgestellt und im späteren Spielverlauf wieder abgerufen werden. Auch die Anzahl der Tetriminos im Vorschaufenster kann einen Einfluss auf die Spielstrategie haben.

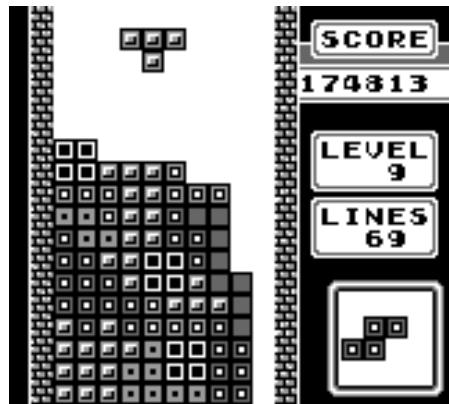


Abbildung 2.3.: Beispiel eines Spielfeldes bei der Spielstrategie *Stacking*.

Die meisten Tetris-Profis spielen nach dieser Strategie, wie bei den Finalspielen der Classic Tetris World Championship (CTWC) schön zu sehen ist (siehe Abb. 2.4). Der Stapel ist bei den meisten Spielern links angesiedelt. Grund dafür ist, dass es angenehmer ist, die Links- statt die Rechts-Taste zu betätigen, da diese mit dem Finger einfacher erreichbar ist.

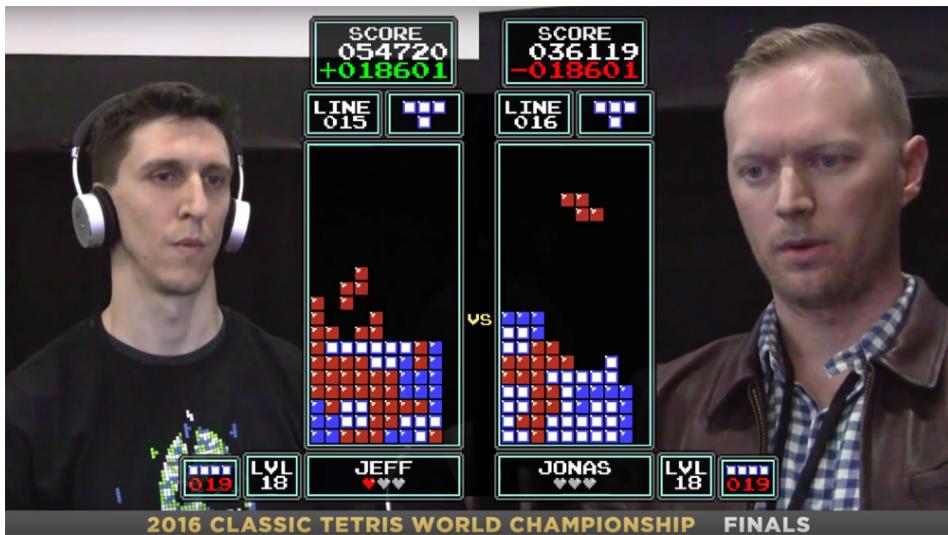


Abbildung 2.4.: Jeff Moore und Jonas Neubauer beim Finale der Classic Tetris World Championship 2016.

Beim Stacking riskiert man jedoch immer einen Block out, da man bei ungünstigen Spielsteinsequenzen weniger Ausweichmöglichkeiten hat und das Spielfeld so schneller überfüllt. Es lohnt sich, möglichst schnell viele Punkte zu sammeln, da mit dem Level auch die Spielgeschwindigkeit ansteigt, was das Stacking erschwert.

Im Multiplayer-Modus (siehe Abschnitt 2.2.5) empfiehlt sich ebenfalls die Stacking-Strategie. Der Spieler kann die Niederlage seines Gegners beschleunigen, je mehr Linien er diesem überträgt.

2.2.6.2. Überleben

Das Ziel dieser Strategie ist es, möglichst lange zu spielen. Der Spieler geht dabei keine Risiken ein, Linien baut er wenn möglich sofort ab. Oft werden bei dieser Methode nur eine oder zwei Linien auf einmal abgebaut.

Mit dieser Strategie ist tendenziell eine längere Spielzeit möglich, was sich dann auch positiv auf die Punktzahl auswirkt. Dies rechtfertigt die Strategie zumindest dann, wenn kein Zeitdruck vorhanden ist.



2.2.7. Weltmeisterschaft und Rekorde

Im Jahr 1990 gab es den ersten Nintendo-Weltmeisterschaftswettbewerb[2].

Seit anhin ist es das Ziel eines jeden Tetris-Spielers, die sogenannte Max-Out-Score zu erreichen – die höchstmögliche Punktzahl von 999'999. Harry Hong war 2009 der erste dokumentierte Spieler, dem dies gelang.

Seit 2010 existiert die Classic Tetris World Championship. Die Turniere dieser Meisterschaft werden auf der Konsole Nintendo Entertainment System (NES) von 1983 mit dem klassischen Nintendo-Controller und dem originalen Game-Modul *Tetris* von 1984 gespielt. Amtierender und sechsfacher Weltmeister ist Jonas Neubauer. Er hat bereits selber die Maximalpunktzahl von 999'999 erreicht[29].

In der Geschichte der Classic Tetris World Championship wurde der Punkterekord von 999'999 bereits über 20 mal erreicht. Jedoch gelingt das selbst den besten Spielern nur äußerst selten.

2.3. Robotik

Dieser Abschnitt beschreibt die Grundlagen der Robotik, welche für diese Arbeit relevant sind.

2.3.1. Freiheitsgrade

Soll ein Roboter ein Gerät bedienen, ist die Frage zu klären, wie viel Freiheitsgrade F benötigt werden. In der Mechanik drückt der Begriff *Freiheitsgrad* die Möglichkeit aus, im Raum voneinander unabhängige Bewegungen auszuführen.

Die Anzahl benötigter Freiheitsgrade bestimmt, welchen Typ von Roboter eingesetzt werden muss. Will man einen starren Körper im dreidimensionalen Raum bewegen, so braucht man drei Freiheitsgrade, also drei Achsen. Will man einen dreidimensionalen Körper um jede Achse rotieren lassen und frei im dreidimensionalen Raum bewegen, sind sechs Freiheitsgrade, also sechs Achsen notwendig. Die Anzahl benötigter Freiheitsgrade bestimmt den Aufbau und die Anzahl der Achsen des Roboters.

2.3.2. Aktoren

Aktoren (Antriebselemente) setzen elektrische Signale in mechanische Bewegung oder andere physikalische Größen um und greifen damit aktiv in den Prozess ein[11].

Die meisten Aktoren werden elektrisch oder magnetisch angetrieben. Andere, beispielsweise Bimetall-Aktoren, sind thermisch betrieben. Für diese Arbeit sind nur elektrische Aktoren relevant, andere Typen sind hier bewusst nicht beschrieben.

Es gibt unterschiedliche Aktoren, die elektrische Signale in Bewegung umwandeln. Die besonders im Modellbau und in der Robotik stark verbreiteten Varianten sind im Folgenden kurz beschrieben:

Servos Ein Servo bezeichnet in der Elektrotechnik einen Verbund aus Ansteuerungs- und Antriebseinheit. Dies kann beispielsweise ein Elektromotor samt seiner Steuerelektronik sein. Im allgemeinen Sprachgebrauch werden Servos häufig mit Servomotoren gleichgesetzt[18].

Schrittmotoren Ein Schrittmotor ist ein Synchronmotor, bei dem der Rotor (drehbares Motorteil mit Welle) durch ein gesteuertes, schrittweise rotierendes, elektromagnetisches Feld der Statorspulen (Stator = nicht drehbarer Motorteil) um einen minimalen Winkel (Schritt) oder sein Vielfaches gedreht werden kann[17].

Elektromagnete Ein Elektromagnet besteht aus einer Spule, in der sich bei Stromdurchfluss ein magnetisches Feld bildet. In der Spule befindet sich meist ein offener Eisenkern, der das Magnetfeld führt und verstärkt[12].



3. Realisierung

3.1. Versuchsaufbau

Dieser Abschnitt beschreibt den Versuchsaufbau, der für die Entwicklung und das Testing des Roboters zum Einsatz gekommen ist.

Der Versuchsaufbau besteht grundsätzlich aus einer Hardware- und einer Software-Komponente. Weiter gehört dazu ein Game Boy und das zugehörige Tetris-Spiel. Abb. 3.1 zeigt den Versuchsaufbau des finalen Roboters.

Die Hardware-Komponente setzt sich zusammen aus der Kamera, den Aktoren, der Halterungskonstruktion und einem Labornetzteil. Die Halterungskonstruktion hält alle Einzelteile zusammen und ermöglicht es, den Game Boy zu fixieren. Dieser Teil wird im Abschnitt 3.2 erläutert. Außerdem gehört die Plattform dazu, worauf die Software läuft. Mehr dazu im Abschnitt 3.3.

Die Software-Komponente besteht aus einem spielunabhängigen Teil – der Engine (Abschnitt 3.5) – und einem spielspezifischen Teil – dem Agenten (Abschnitt 3.6).

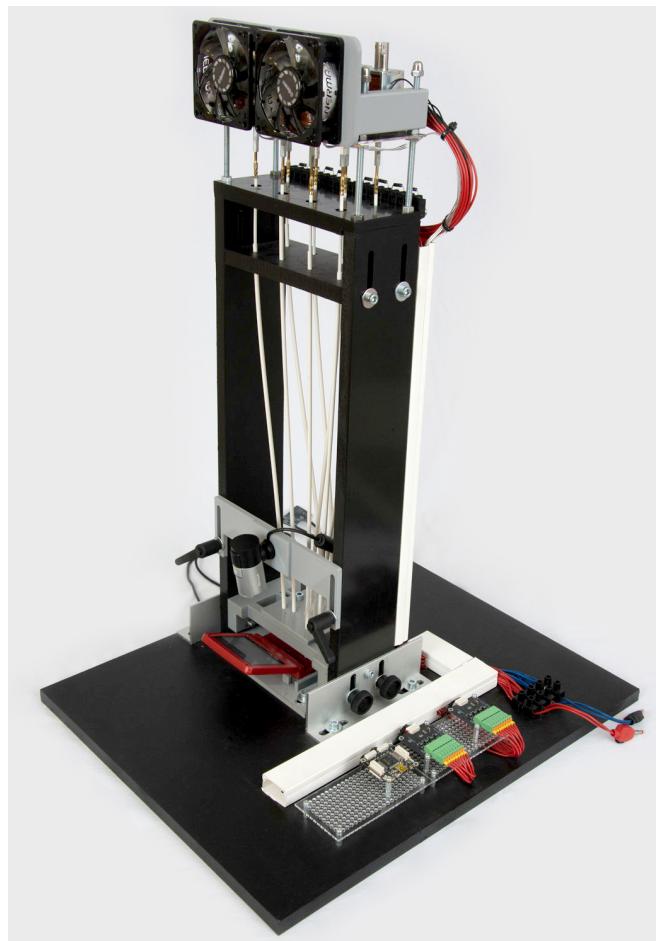


Abbildung 3.1.: Versuchsaufbau des finalen Roboters ohne Computer.

3.2. Roboter

Der erste Gedanke war, ein Roboter zu konstruieren, der die menschlichen Daumen imitiert und so auf dem Game Boy spielt. Eine Zwei-Finger-Bedienung entspricht der menschlichen Spielweise und ist somit fair. Ein solcher Roboter müsste über drei Freiheitsgrade verfügen – also drei Achsen pro Finger. Die Konstruktion eines solchen Roboters ist jedoch kompliziert und kostenintensiv. Außerdem verwenden Profi-Tetris-Spieler unter Umständen mehr als zwei Finger zum Spielen[10]. Aus diesem Grund wurde das Konzept der „Roboterdaumen“ nicht weiter verfolgt.

Beim zweiten Konzept verfügt jede Taste (A, B, Start, Select) und jede Richtung ($\uparrow, \downarrow, \leftarrow, \rightarrow$) des Steuerkreuzes über einen eigenen Aktor. Zusätzlich ist diese Variante solider und einfacher umzusetzen.

3.2.1. Grundgedanke

Um eine Taste des Game Boys zu betätigen, benötigt man genau einen Freiheitsgrad. Die Bewegungen Heben und Senken sind ausreichend, um eine Taste zu drücken bzw. loszulassen. Somit entstand die Idee, statt zwei Finger mit je drei Achsen, acht Finger mit je einer Achse einzusetzen. Dieses Konzept ist mechanisch einfach umzusetzen und hat den Vorteil, dass der Roboter alle acht Tasten gleichzeitig betätigen kann.

3.2.2. Prototypen

Dieser Abschnitt beschreibt die verschiedenen Prototypen, welche im Verlaufe dieser Arbeit entwickelt wurden.

3.2.2.1. Prototyp 1

Aktoren bewegen ein Stoss-Druckkabel durch ein Führungsrohr (Bowdenzug) und betätigen so die jeweiligen Tasten des Game Boys (siehe Abb. 3.2).

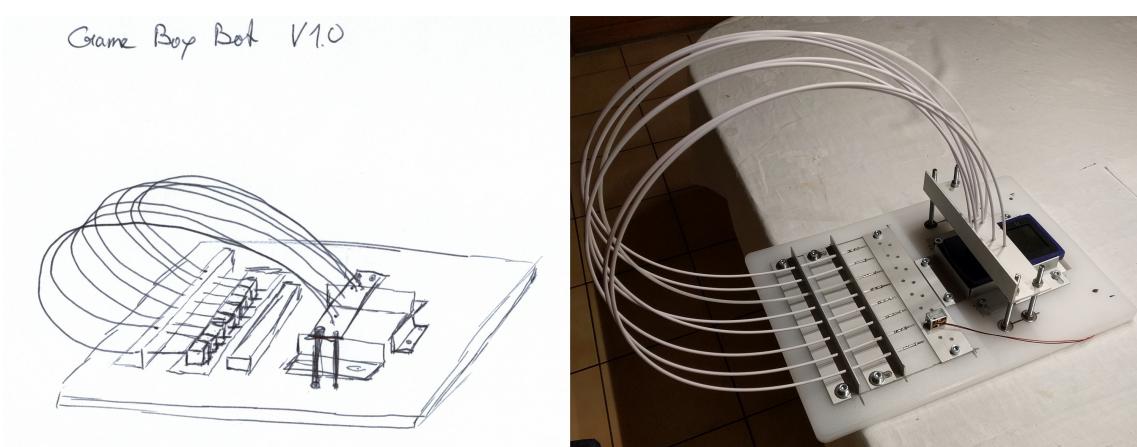


Abbildung 3.2.: Skizze und Foto des ersten Prototypen des Roboters.

Durch die starke Biegung des Bowdenzugs entstand ein grosser Reibungswiderstand. Dieser Widerstand war für die Hubmagneten zu gross. Die Hubmagneten können eine maximale Kraft von 2,8 Newton aufbringen. Die eingesetzten Hubmagnete haben wegen ihrer geringen Grösse auch eine sehr schlechte Kühlleistung. Bereits nach wenigen Sekunden Betrieb werden diese zu heiss.

Optimierungsmassnahmen

Die Bowdenzüge könnten ersetzt werden, indem man die Hubmagnete direkt über den jeweiligen Tasten des Game Boys montiert. Das Problem ist hierbei die Grösse der aktuell eingesetzten Kuhnke Hubmagneten. Mit $50 \times 30 \times 26$ mm passen diese nicht zwischen die Game-Boy-Tasten, die zwischen 17 – 12.65 mm auseinander liegen. Der Test mit kleineren Hubmagneten, Intertec ITS-LS1110B-D-24VDC mit einer Abmessung von $22 \times 11 \times 10$ mm, welcher



im Rahmen des Prototypen 1 durchgeführt wurde, ergab, dass diese durch ihre geringe Oberfläche schon nach wenigen Hüben extrem heiß werden. Ein weiterer Lösungsansatz, die Bowdenzüge zu ersetzen, ist der Einsatz eines Gestänges. Hierbei wird mit Hilfe zweier Gelenke und einer Verbindungsstange der Hubmagnet direkt mit einem Finger verbunden (siehe Abb. 3.3).

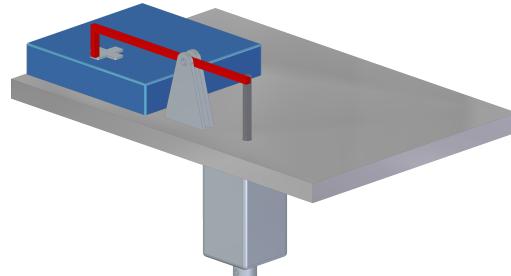


Abbildung 3.3.: Skizze des Konzepts mit Gelenken.

Ein Gestänge hat im Vergleich zu den Bowdenzügen kaum Reibung. Der grosse Nachteil hierbei ist die grössere Masse, welche eine grössere Trägheit zur Folge hat. Nach dem ersten Newtonschen Gesetz $F = m \cdot a$ resultiert mit diesem Lösungsansatz eine geringere Beschleunigung a , da F konstant und $m_g < m_b$ ist. Also ist $a = \frac{F}{m_g}$.

Die Option, die am nächsten liegt, ist der Einsatz von Bowdenzügen, welche geringere Reibung, also speziell gute Gleiteigenschaften zwischen Innen- und Führungsrohr aufweisen. Die Firma LEOMOTION bietet solche Bowdenzüge an. Das Innenrohr weisst hierbei eine Sternform auf (siehe Abb. 3.4).

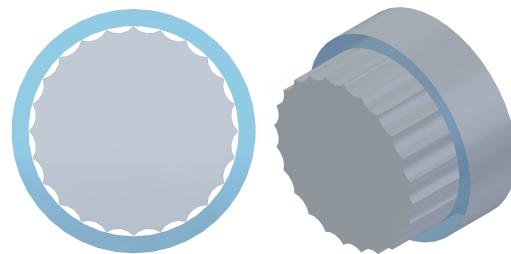


Abbildung 3.4.: Querschnitt des verwendeten Bowdenzuges. Querschnitt (links) und 3D-Modell (rechts).

Dadurch wird die Kontaktfläche zwischen Innen- und Führungsrohr minimiert und die Reibung stark reduziert.

Erkenntnis

Das Resultat der Analyse zeigt, dass das Grundkonzept eine gute Basis darstellt und darauf aufgebaut werden kann:

- Mit dem alten Prototypen 1 wurden schon viele Erfahrungen gesammelt.
- Das Konzept erlaubt eine einfache Bauweise ohne komplexe Mechanik.
- Die dafür notwendigen Komponenten wie Hubmagnete und Bowdenzüge sind einfach zu beschaffen.

3.2.2.2. Prototyp 2

Das Konzept mit acht Hubmagneten – also einen Aktor pro Taste – wird beibehalten. Um die zusätzliche Reibung, welche durch die gekrümmten Bowdenzüge (siehe Abb. 3.2) entsteht zu reduzieren, werden die Bowdenzüge möglichst gerade gehalten. Dazu wird allerdings eine neue Konstruktion benötigt.

Die neue Konstruktion, welche aus vielen Einzelteilen besteht, wird mittels einer professionellen Konstruktions-Software (Solid Edge ST9) gezeichnet (siehe Abb. 3.5). Mit Ausnahme von eingekauften Baugruppen wie Hubmagneten, Bowdenzügen, Schrauben und diversem Kleinmaterial wird jede Komponente als isometrisches Teil konstruiert. Aus den so konstruierten Teilen wird eine Baugruppe erstellt.

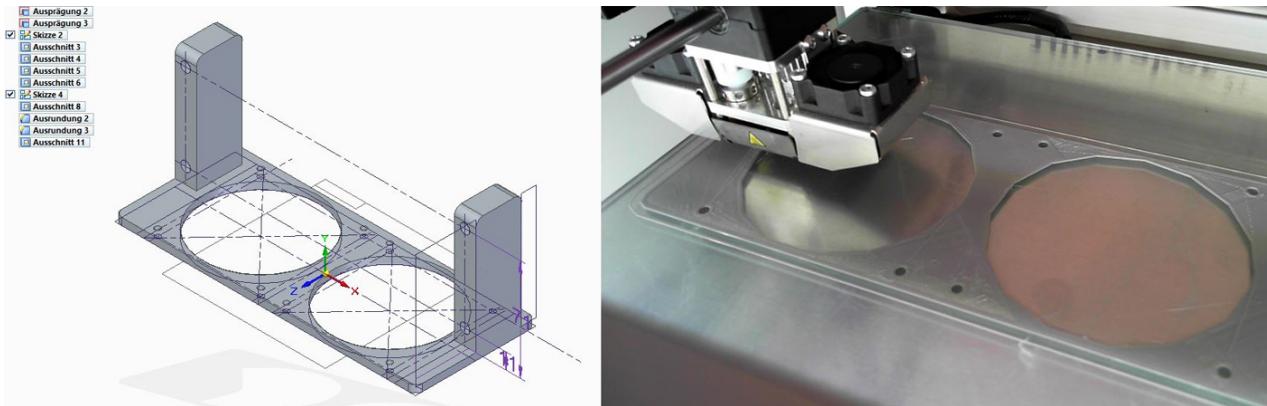


Abbildung 3.5.: Screenshot aus Solid Edge (links) und das Bauteil, erstellt vom 3D-Drucker (rechts).

Die schwarz eingefärbte Rahmenkonstruktion wird mit Hilfe einer Säulenbohrmaschine mit Kreuztisch aus einer PVC-Platte gefräst. Die gelb eingefärbten Bauteile sind aus einem Aluminiumwinkelprofil gefräst. Die grau eingefärbten Teile werden mit einem 3D-Drucker (Ultimaker 2+) ausgedruckt (siehe Abb. 3.6).

Die neue Konstruktion erlaubt es, den Game Boy in der X- und Y-Achse unterhalb der Finger auszurichten. Auch ist es möglich, jeden der acht Finger in der Höhe zu verstellen. Dies erlaubt es, jeden Finger in der optimalen Höhe über der Game-Boy-Taste zu platzieren.

Die Taste zum Ein- und Ausschalten sowie die Ladebuchse und Schnittstelle des Game Boys sind jederzeit zugänglich.

3.2.3. Aktoren

Aufgrund der enormen Geschwindigkeit von Hubmagneten, werden diese als Aktoren eingesetzt¹. Bei Hubmagneten muss die Kraft nicht via Getriebe oder Umlenkung umgewandelt werden, sondern kann direkt an die Tasten des Game Boys weitergegeben werden. Schrittmotoren und Servomotoren sind langsamer, da sie elektrische Energie nur indirekt in mechanische Bewegung umwandeln.

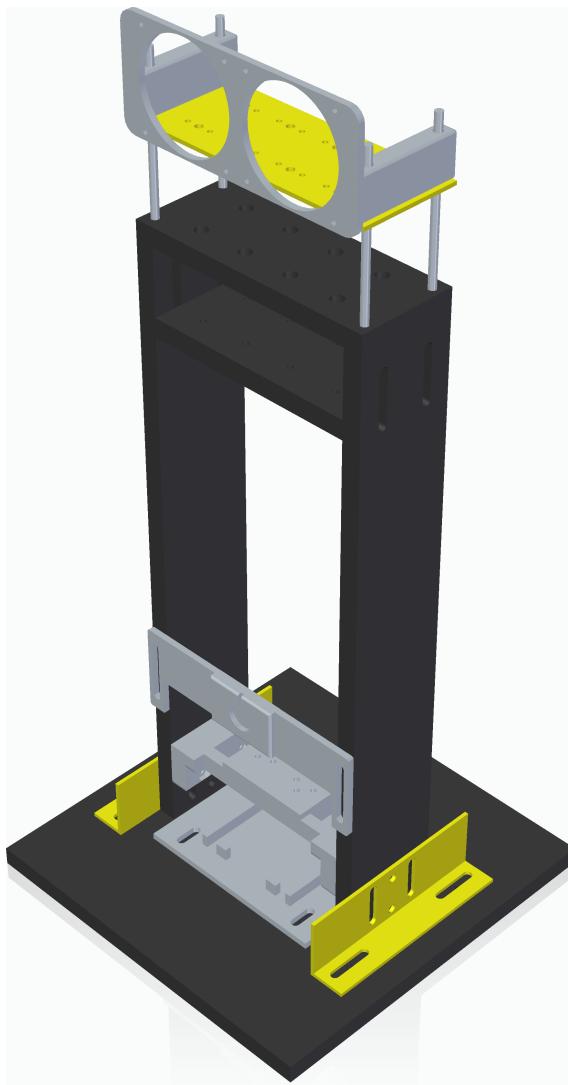
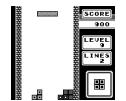
3.2.3.1. Tinkerforge

Der Controller für die Aktoren wurde mit Tinkerforge-Bricks und -Bricklets realisiert. Tinkerforge ist ein Hardware-Baukastensystem, das es dem Entwickler erlaubt, seine Ideen einfach umzusetzen. Der Master Brick 2.1 kommuniziert via Modbus mit Aktoren und Sensoren, den sogenannten Bricklets. Folgende Liste führt die Vorteile von Tinkerforge gegenüber anderen Plattformen wie z.B. Arduino oder Raspberry Pi auf:

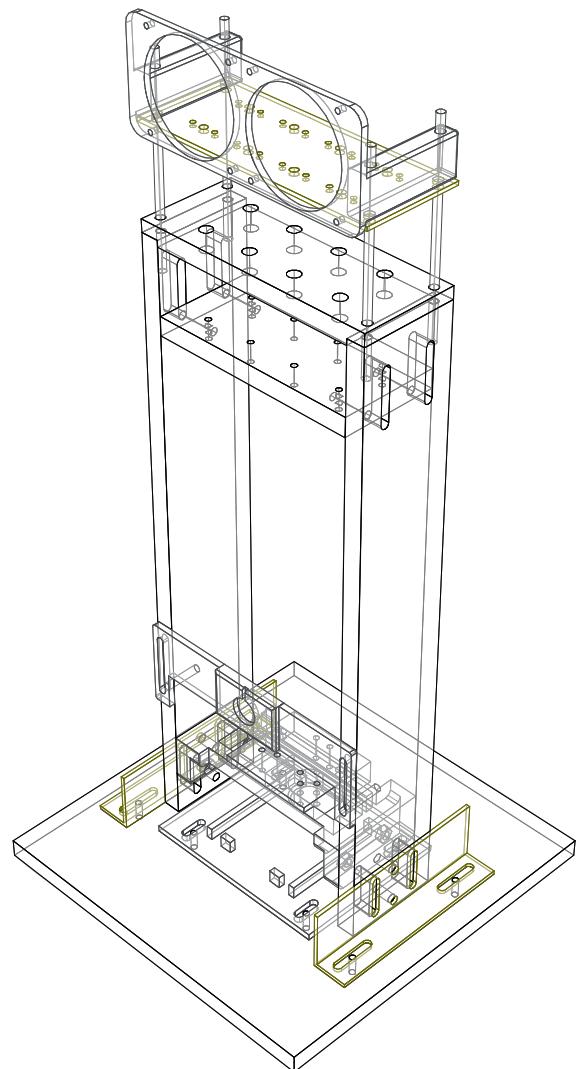
- Dank Abstraktionsschicht unabhängig von Programmiersprache oder Plattform
- Industrial Bricklets verfügen über genügend hohe Schaltleistung
- Tinkerforge steht durch die Berner Fachhochschule zur Verfügung

Aufgrund dieser Vorteile haben sich die Autoren für den Einsatz von Tinkerforge entschieden. Nicht zuletzt wegen der Einfachheit und Flexibilität der Plattform.

¹Prominentes Beispiel für Hubmagnete sind die Bedienhebel eines Flipperautomaten. Auch da sind sehr schnelle Bewegungen erforderlich.



(a) Ausgefülltes Modell.



(b) Drahtgittermodell.

Abbildung 3.6.: Roboter als 3D-Modell.

3.2.4. Elektronik

Die Leistungsaufnahme beträgt 8 Watt pro Hubmagnet. Für das Spiel Tetris müssen maximal zwei Tasten gleichzeitig gedrückt werden (Rotieren und Verschieben). Somit ist davon auszugehen, dass eine maximale Leistungsaufnahme von 16 Watt stattfindet. Abb. 3.7 zeigt den elektronischen Schaltplan des Roboters. Der Master Brick 2.1 von Tinkerforge (M) ist über ein Modbus-Kabel mit zwei Quad Industrial Relais Bricklets (I und II) gesteuert. Der Master Brick wird über die USB-Schnittstelle mit 5V DC versorgt. Die acht Hubmagnete (H1, H2, ..., H8) sind durch die Kontakte der Industrial Relais beschalten. Die Speisung der Hubmagnete erfolgt über ein externes 24V-DC-Netzteil. Zwei 12V-DC-Lüfter (M1 und M2) sind in Serie geschalten und werden durch das externe Netzteil mit Strom versorgt.

3.2.4.1. Schutz vor Überspannung

Für den Dauerbetrieb der Hubmagnete werden diese zusätzlich mit einer Freilaufdiode nachgerüstet. Grund hierfür ist die Selbstinduktion der Hubmagnete. Nach dem Abschalten der Speisespannung sorgt die Selbstinduktion der Spule dafür, dass der Strom zunächst in der ursprünglichen Richtung weiter fliesst. Ohne Freilaufdiode führt das zu einer Spannungsspitze, die sich zur Betriebsspannung addiert und die Quad Relais schädigen oder zerstören

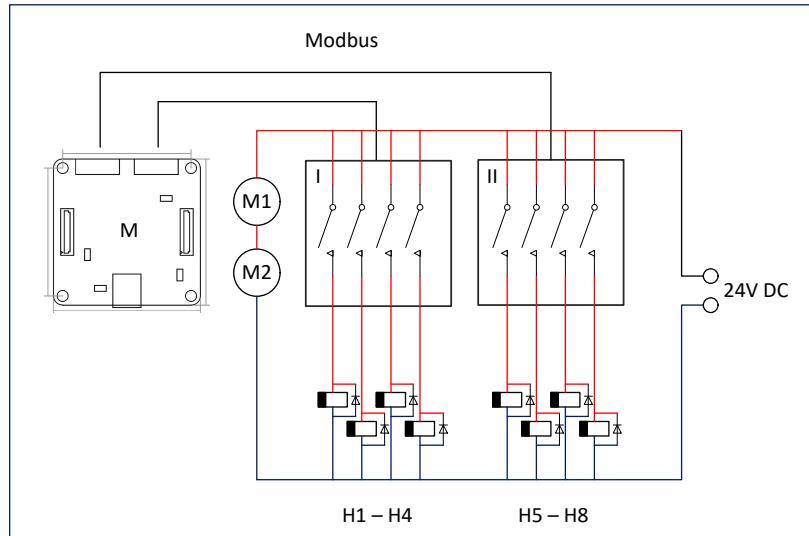


Abbildung 3.7.: Elektronischer Schaltplan des Roboters.

kann. Mit Freilaufdiode wird die Spannungsspitze jedoch auf die Durchlassspannung der Diode (bei Silizium etwa 0.6 V) begrenzt. Das schützt die elektronischen Bauteile (beispielsweise Halbleiter wie Transistoren), aber auch Schaltkontakte, sehr effektiv vor Überspannung.

3.2.4.2. Schutz vor Überhitzung

Mit zunehmender Spieldauer werden die Hubmagnete sehr warm. Speziell die ↓-Taste wird durch das lange Drücken während des Softdrops nach ungefähr 10 Minuten Spieldauer über 65°C heiss. Um hitzebedingten Schäden vorzubeugen, wurde der Versuchsaufbau mit zwei 12V-DC-Lüftern der Firma Enermax ergänzt. Die Lüfter starten, sobald der Roboter mit Spannung versorgt wird.

3.2.4.3. Schutz vor Abgleiten

Die Enden der Bowdenzüge sind sternförmig (siehe Abb. 3.4) und haben einen Durchmesser von 3 mm. Die Oberflächen der Tasten des Game Boys sind teils konvex. Während der Testphase glitten die Finger manchmal ab und drückten daneben (siehe Abschnitt 4.4).

Das Problem wurde mit Zylindern gelöst, welche über die Bowdenzüge gestülpt werden. Die grössere Druckfläche von 5 mm Durchmesser verhindert ein Abgleiten (siehe rotes Bauteil in Abb. 3.8).

3.2.4.4. Wegbegrenzung

Um die Performance der Aktoren weiter zu verbessern und die Tasten des Game Boys zu schonen, wurde der Hubweg der Hubmagnet mit Wegbegrenzer verkürzt (siehe blaues Bauteil in Abbildung 3.8). Der Wegbegrenzer verhindert, dass der Eisenkern des Hubmagneten in seine Ausgangsposition zurück gedrückt wird. Der Hubweg wird somit von 10 mm auf 3 mm reduziert.

3.2.4.5. Reibungsreduzierung

Um die Reibung der Bowdenzüge weiter zu reduzieren, wurden die beiden Enden des Führungsrohrs mit einem Senkbohrer 45° angesenkt. Somit wird ein Verkannten am Ein- und Austritt vom Innenrohr gegenüber dem Führungsrohr verhindert.

Um die Bowdenzüge möglichst senkrecht auf das Steuerkreuz des Game Boys zu führen, werden diese mit Isolierband gebündelt, was sich ebenfalls positiv auf die Gleiteigenschaften auswirkt.

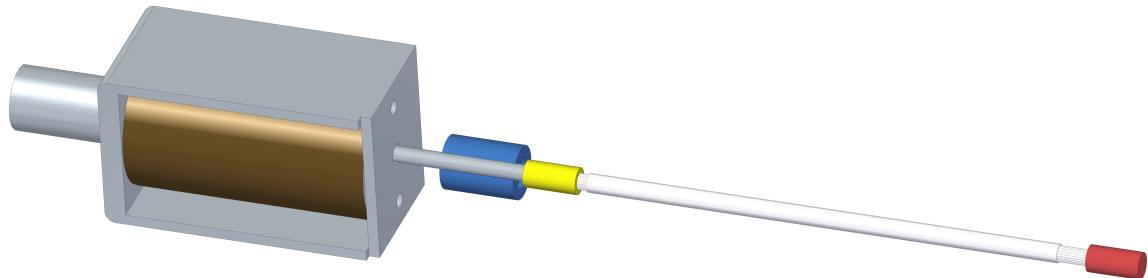


Abbildung 3.8.: Einer von acht Aktoren des Roboters.

3.2.5. Stückliste

- 8 × Kuhnke H3486-R-F-24V
- 1 × Tinkerforge Master Brick 2.1
- 2 × Tinkerforge Industrial Quad Relay Bricklet
- 8 × LL-Bowdenzug 1500x4/3mm inkl. Anschluss M3
- 8 × Verbindungsmuffe M3 x 20mm
- 1 × Microsoft Webcam LifeCam Studio Business
- 1 × Enermax T.B. Silence (80mm)
- 1 × 24V DC Netzteil 240Watt
- 1m² PVC Platte
- diverses Kleinmaterial

3.2.6. Kamera

Die bereits im Projekt 2 verwendete Kamera Logitech C200 Webcam hat eine Framerate von 15 FPS und eine Auflösung von 640 × 480 Pixel. Die Kamera lieferte zu wenig Bilder pro Sekunde und die schlechte Bildqualität führte zu vielen Fehlern bei der Bildanalyse.

Später wurde die Microsoft-Webcam LifeCam Studio eingesetzt. Sie verfügt über 30 FPS und eine Auflösung von maximal 1920 × 1080 Pixel. Das Objektiv erlaubt die Fokussierung auf den Nahbereich, was für die Aufnahme des Game-Boy-Displays sehr wichtig ist.

Zur Diskussion stand auch der Einsatz einer industriellen High-Speed-Kamera. Unter der Voraussetzung, die Konstruktion des Roboters möglichst preiswert und reproduzierbar zu halten, wurde diese Möglichkeit allerdings nicht weiter verfolgt.

Zum Einsatz kam auch die Kamera ELP-USBFHD01M-BL28. Sie ermöglicht Aufnahmen von bis zu 60 FPS. Diese Geschwindigkeit wurde durch Limitierungen seitens der Decoder-Library FFmpeg jedoch nicht erreicht.

Ein möglicher Ansatz wäre auch der Einsatz von mehreren Kameras. Dadurch könnte die Performance der Bildanalyse erhöht werden, weil das Sampling parallel statt sequntiell ausgeführt werden könnte. Aus Preisgründen wurde auch dieser Ansatz nicht geprüft.

3.2.6.1. Kamerahalterung

Zusätzlich zur Optimierung der Kamera wurde deren Halterung mehrmals überarbeitet. Eine zufriedenstellende Lösung ist eine sehr kompakte Halterung, die direkt am Mast des Roboters angebracht ist. Außerdem unterdrückt die kompakte Bauweise störende Vibrationen.

3.2.6.2. Framerate

Die Framerate limitiert die Geschwindigkeit, mit der neue Kamerabilder analysiert werden können.

Der Game Boy läuft mit einer Framerate von 59.73 FPS[8]. Theoretisch müsste die Kamera also eine Framerate von etwa 60 FPS haben, um kein Bild des Game-Boy-Displays zu verpassen. Eine noch höhere Framerate würde insofern etwas bringen, als dass die Reaktionszeit des Roboters dadurch gesenkt würde. Im schlimmsten Fall wird das Bild kurz vor dem Framewechsel des Game-Boy-Displays aufgenommen. In diesem Fall kann das neue Bild erst nach $\frac{1}{\text{Framerate}}$ Sekunden analysiert werden.

Weitere Überlegungen zum Einfluss der Framerate auf die Gesamtperformance sind im Abschnitt 3.6.6 zu finden.

3.3. Plattform

Bereits zu Beginn der Vorarbeit entschied sich das Projektteam, Visual Studio mit C# als Programmiersprache einzusetzen. Da die Software bereits während der Projektarbeit weit fortgeschritten war, wurde entschieden, weiter auf einer .NET-Plattform aufzubauen.

Es wurde zwar versucht, den C#-Code mit Mono auf ein Raspberry Pi (siehe Abschnitt 3.3.1.1) zu migrieren, dabei tauchten jedoch grössere Probleme auf und diese Idee wurde verworfen.

Stattdessen entstand die Idee, die Applikation auf einem kleinen PC laufen zu lassen. Hierzu war ein Mini-ITX-Mainboard der Firma ASUS vorgesehen (siehe Abschnitt 3.3.1.2). Das Mainboard hat sämtliche notwendigen Hardwarekomponenten direkt auf der Platine verbaut. Ebenfalls kann das Mini-ITX-Mainboard via Notebook-Netzteil mit Strom versorgt werden. Somit ist kein klassischer PC-Aufbau notwendig und der Roboter kann als alleinstehende Einheit funktionieren. Die Spielergebnisse auf dieser Plattform waren mangels Performance unbefriedigend.

Letztendlich erzielte das Lenovo ThinkPad T440p (siehe Abschnitt 3.3.1.4) mit einer Quad Core CPU die besten Resultate.

3.3.1. Varianten

3.3.1.1. Raspberry Pi 3

Modell Model B

Prozessor 1.2 GHz 64-bit quad-core ARMv8 CPU

Arbeitsspeicher 1 GB RAM

Kamera Raspberry Pi NoIR Camera V2 (8 MP)

Diese Plattform wurde nur ansatzweise getestet, da es Probleme bei der Migration der Software gab². Diese Variante wurde deshalb schnell verworfen, nicht zuletzt auch wegen der geringen Prozessorleistung des Raspberry Pi.

3.3.1.2. Mini-ITX-Mainboard

Prozessor Intel® Core™ i3-4350 Processor, 64 bit

Mainboard ASUS H81T Intel H81

Arbeitsspeicher 8 GB RAM

Diese Plattform wurde in Betrieb genommen und einige Spiele durchgeführt. Jedoch war die Spiel-Performance offensichtlich viel schlechter – sehr wahrscheinlich wegen der geringen Prozessorleistung. Es wurde deshalb auf weitere Tests verzichtet.

²Der Code aus der Vorarbeit wurde in C# geschrieben und hat eine Abhängigkeit auf die EmguCV-Library. Diese Library liess sich nicht fehlerfrei für den Prozessor des Raspberry Pi 3 kompilieren.



3.3.1.3. Lenovo ThinkPad T450s

Prozessor Intel® Core™ i7-5600U Prozessor, 64 bit

Arbeitsspeicher 12 GB RAM

Diese Plattform kam vor allem für die Entwicklung und das Testing der Software zum Einsatz. Des Weiteren wurden einige Messungen darauf gemacht (siehe Kapitel 4).

3.3.1.4. Lenovo ThinkPad T440p

Prozessor Intel® Core™ i7-4710MQ Prozessor, 64 bit

Arbeitsspeicher 16 GB RAM

Auf dieser Plattform wurden die meisten Tests mit dem Versuchsaufbau unternommen. Ebenfalls wurden sämtliche Rekordspiele auf dieser Plattform erzielt.

3.4. Softwarearchitektur

Dieser Abschnitt beschreibt den Aufbau der Software, die für den Roboter geschrieben wurde. Diagramme und weitere Details zur Implementierung sind im Anhang E zu finden.

Die Komponente, welche die KI enthält, wird nachfolgend als *Agent* bezeichnet.

3.4.1. Designziele

Testbarkeit Jede Software-Komponente soll isoliert und automatisiert getestet werden können.

Austauschbarkeit Lose Kopplung vereinfacht das Austauschen einer spezifischen Implementierung.

3.4.2. Angewandte Designpatterns

Dependency Injection Dependency Injection erleichtert die Test- und Austauschbarkeit der einzelnen Komponenten. Für alle wichtigen Klassen können so Unit-Tests geschrieben und das Testing automatisiert werden. In vielen Bereichen ist es zudem notwendig, dass die Implementierung einer bestimmten Komponente möglichst einfach und schnell ausgetauscht werden kann.

Als Dependency Injection Library wird *Simple Injector* eingesetzt (siehe Abschnitt A.4).

Inversion of Control Die Kontrolle über den Programmfluss liegt nicht beim Agenten, sondern bei der Engine. Der Agent stellt nur ein Interface zur Verfügung, worüber die Engine ihn aufrufen kann. Dies vereinfacht die Implementierung weiterer Agents, da der Entwickler keine Kenntnis über den Aufbau der Engine benötigt.

Der Agent kann als Plug-In (in Form einer DLL) in die Engine geladen werden. Um den Agenten kompilieren zu können, braucht es lediglich eine Referenz auf die Core-DLL. Die Engine hat ebenfalls keine Abhängigkeit auf den Agenten. Sie kann die Implementierung des Agenten zur Laufzeit laden.

Strategy Bei der Bewertungsfunktion des Agenten wird das Strategy Pattern eingesetzt. So kann die Funktion einfach ausgetauscht werden, wenn man z.B. verschiedene Algorithmen miteinander vergleichen will.

State Pattern Für die Implementierung des Agenten wird das State Pattern angewendet (siehe Abschnitt 3.6.5).

3.4.3. Testing

Neue Features wurden immer zuerst auf dem Emulator (siehe Abschnitt 3.5.3) getestet. Danach wurden Integrations- und Unit-Tests auf dem echten Roboter durchgeführt.

Neben Integrationstests wurde sehr stark mit Unit-Tests gearbeitet. Über 500 Unit-Tests wurden im Verlaufe des Projekts erstellt.

3.4.4. Module

Die Engine ist zur besseren Strukturierung in Module aufgeteilt. Im Folgenden ist jedes Modul kurz beschrieben.

GameBot.Core Enthält Interfaces zu sämtlichen modulübergreifenden Komponenten. Zudem stellt es grundlegende Datenstrukturen und Enumerations zur Verfügung.

GameBot.Emulation Enthält den Game-Boy-Emulator (siehe auch Abschnitt 3.5.3)

GameBot.Engine.Emulated Enthält die Implementierung der Engine unter Verwendung des Emulators.

GameBot.Engine.Physical Enthält die Implementierung der Engine für den realen Einsatz.

GameBot.Game.Tetris Enthält die Implementierung des Agenten für das Spiel Tetris. Dies ist die Referenzimplementierung eines Agenten für den Roboter.

GameBot.Game.Tetris.Ga Ausführbarer Teil. Zuständig für die Optimierung der Gewichtsfaktoren der Bewertungsfunktion mittels genetischem Algorithmus.

GameBot.Game.Tetris.Simulator Ausführbarer Teil. Ermöglicht die Simulation von Tetris in einer schlanken aber sehr performanten Routine.

GameBot.Robot.Ui Ausführbarer Teil. Enthält die GUI-relevanten Komponenten.

GameBot.Test Enthält sämtliche Unit-Tests.

3.4.5. Externe Abhängigkeiten

Folgende Libraries wurden in dieser Arbeit eingesetzt:

EmguCV 3.0.0 OpenCV-Wrapper für .NET

GAF 2.3.0 Framework für genetische Algorithmen

Moq.4.2.1510.2205 Mocking

NLog.4.3.10 Logging

NUnit.2.6.4 Unit-Testing

SimpleInjector.3.2.2 Dependency Injection

Tinkerforge.2.1.11 API für Tinkerforge



3.5. Engine

Die Engine übernimmt alle Aufgaben des Bots, welche nicht spielspezifisch sind. Der Roboter ist so in der Lage, ein beliebiges Game-Boy-Spiel zu spielen – vorausgesetzt ein Agent wurde dafür implementiert.

Die Hauptaufgabe der Engine ist die Abstraktion der Hardware. Namentlich sind dies Kamera (Input) und Aktoren (Output). Dem Agenten wird so ein einfaches Interface zur Verfügung gestellt, worüber er mit der physikalischen Welt kommunizieren kann. Ziel dieser Abstraktion ist es, die Außenwelt bei Bedarf simulieren zu können. Mehr dazu im Abschnitt 3.5.3.

Die Engine wird in einer Schleife ausgeführt. Pro Durchgang werden jeweils folgende Schritte sequentiell ausgeführt:

- Aufnahme des Kamerabildes
- Bildverarbeitung
- Aufruf des Agenten (Erfassung des Spielstands, Berechnung der auszuführenden Aktionen, Übersetzung der Aktionen in Tastendrücke)
- Ausführung der Tastendrücke

Beim Aufruf des Agenten wird diesem eine Referenz auf den Executor mitgegeben. Damit kann er Tastendrücke auslösen, welche die Engine dann synchron oder asynchron ausführt. Der Agent bestimmt selber, wann er seine Durchführung unterbricht und die Kontrolle an die Engine zurückgibt. Er muss dies spätestens dann tun, wenn er ein neues Bild von der Kamera benötigt.

Eine weitere Aufgabe der Engine ist das Zeitmanagement. Die Engine stellt einen Service zu Verfügung, über den der Agent die Simulationszeit abfragen kann. Die Simulationszeit ist die verstrichene Zeit, seit der Bot gestartet wurde.

In Abb. 3.9 ist der Aufbau der Engine mit den einzelnen Unterkomponenten dargestellt.

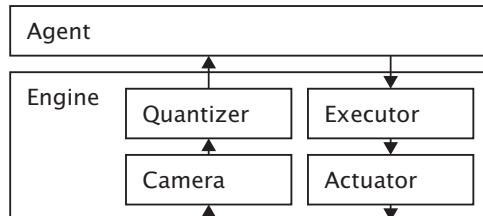


Abbildung 3.9.: Stark vereinfachte Darstellung der Engine und ihrer Komponenten.

3.5.1. Hardware-Ansteuerung

Die Aktoren werden über Tinkerforge angesteuert. Dafür muss auf dem Rechner brickd installiert sein – ein Daemon, welcher als Server dient. Über das Internet-Protokoll lassen sich schliesslich die Hardware-Komponenten steuern. Verwendet wird die Tinkerforge-API, welche per NuGet-Package eingebunden wird.

Die Kamera ist per USB 2 mit der Plattform verbunden. Die Engine steuert die Kamera per EmguCV an, genauer gesagt über die Capture-Komponente. EmguCV verwendet intern FFmpeg zur Decodierung des Videostreams. Die Bilder werden mit einer Framerate von 30 FPS erfasst. Das Auslesen übernimmt ein eigener Thread, damit Auslesen und Verarbeiten der Bilder parallel erfolgen können.

Abb. 3.10 zeigt schematisch den Aufbau der Engine und den Datenfluss bei Input und Output.

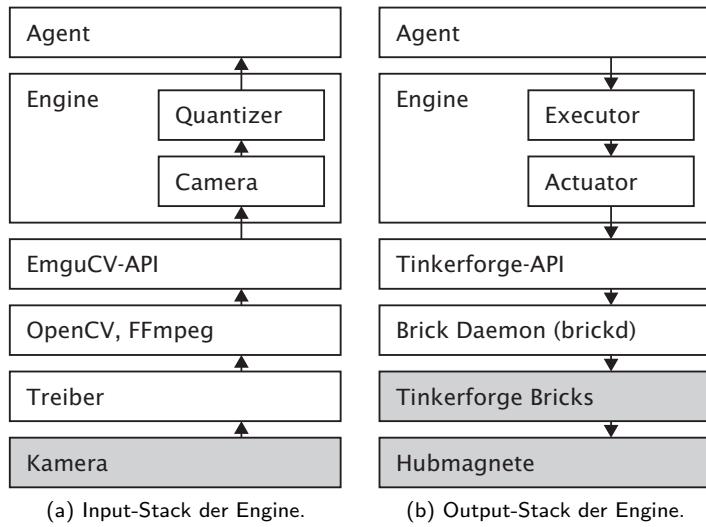


Abbildung 3.10.: Datenfluss der Engine bei Input und Output.

3.5.2. Bildverarbeitung

Damit der Agent die Bilder interpretieren kann, müssen die Rohdaten der Kamera zuerst verarbeitet werden.

Das Quellbild der Kamera ist farbig und hat eine Auflösung von 640×480 Pixel³.

In der Vorstudie wurde der Game Boy Color eingesetzt. Dessen Display hat einen sehr schlechten Kontrast, da keine Hintergrundbeleuchtung vorhanden ist. Für diese Arbeit wurde deshalb auf einen Game Boy Advance SP gewechselt, welcher eine Hintergrundbeleuchtung hat. Diese vereinfacht die Bildverarbeitung enorm, da die Spielsteine so besser erkannt werden können.

Da die Bildverarbeitung in Echtzeit ablaufen muss, können nur sehr effiziente Algorithmen eingesetzt werden. Es werden ausschliesslich lokale Operatoren verwendet. Die resultierende Operation muss zu grossen Teilen invariant gegenüber Rauschen und ungleichen Lichtverhältnissen sein.

3.5.2.1. Entzerrung

Das Zielbild soll rechteckig sein und exakt der Auflösung des Game-Boy-Displays entsprechen, also 160×144 Pixel. Für den Agenten ist es hilfreich, dass er die Bilder so normalisiert wie möglich erhält. So kann er diese einfach auswerten.

Als erster Schritt ist also eine Entzerrung nötig, um das perspektivisch verzerrte Bild des Displays in eine rechteckige Form zu bringen⁴ und gleichzeitig eine Größenanpassung zu erreichen (siehe Abb. 3.11). Für die Entzerrung wird die OpenCV-Funktion WarpPerspective angewendet. Diese Funktion verlangt eine Transformations-Matrix und eine Zielgröße. Die Transformationsmatrix kann mittels der Funktion GetPerspectiveTransform anhand von vier Keypoints berechnet werden. Die vier Keypoints sind die Ecken des Game-Boy-Displays im Quellbild.

Da die relative Position der Kamera zum Display nicht verändert wird, ist auch die Transformations-Matrix konstant und muss nur initial vorberechnet werden. Die Kalibrierung muss möglichst exakt sein, da zu grosse Abweichungen später bei der Bildanalyse Fehler verursachen können (siehe Abschnitt 4.2).

³Verschiedene Kameras kamen bei dieser Arbeit zum Einsatz. Dabei variierten jeweils Auflösung und Frame rate.

⁴Die perspektivische Verzerrung ergibt sich aus der schrägen Positionierung der Kamera. Das Display kann aber nicht senkrecht von der Kamera erfasst werden, da sich so die Kamera im Display spiegeln und später die Bildanalyse erschweren würde.

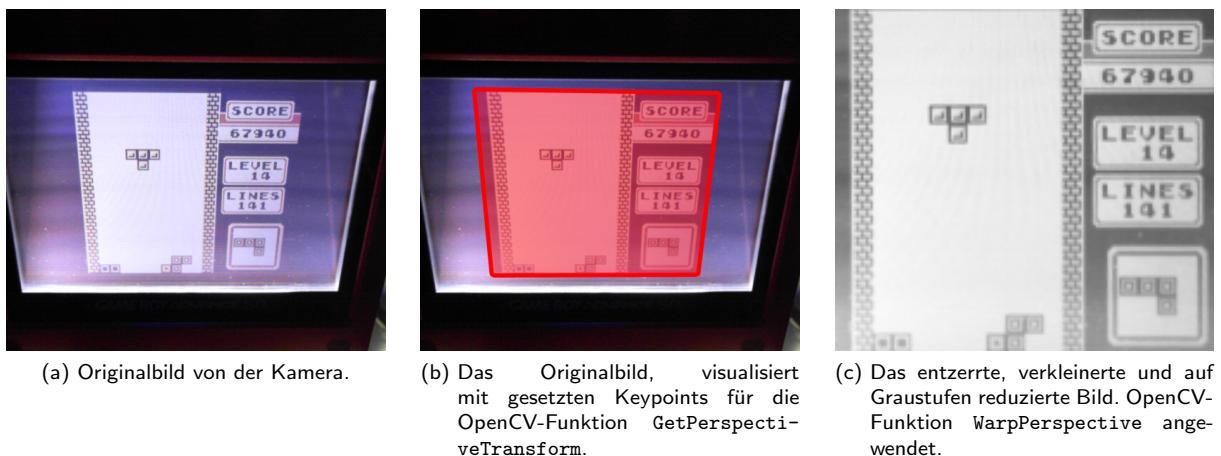
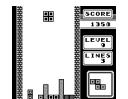


Abbildung 3.11.: Das Verfahren der Bildentzerrung.

3.5.2.2. Binarisierung

Der Informationsgehalt auf dem Bild kann zur einfacheren Auswertung weiter reduziert werden. Besonders weil das Spiel Tetris und auch die meisten anderen Game-Boy-Classic-Spiele sowieso nur vier Graustufen unterscheiden. Die Binarisierung erhöht auch den Kontrast des Bildes, was wiederum die Bildanalyse erleichtert. Zwei Verfahren wurden hier getestet: einfaches und adaptives Thresholding.

Das einfache Thresholding funktioniert nur, wenn die Lichtverhältnisse zeitlich und räumlich (innerhalb des Bildes) beinahe konstant sind. Sonst treten Artefakte auf, welche zu Informationsverlust führen (siehe Abb. 3.12).

Das adaptive Thresholding hingegen ist robuster gegenüber variablen Lichtverhältnissen. Die OpenCV-Funktion nimmt unter anderem zwei Parameter entgegen: blocksize und constant. Die Justierung dieser Parameter erfolgte manuell und jeweils pro Kamera und Game-Boy-Modell.

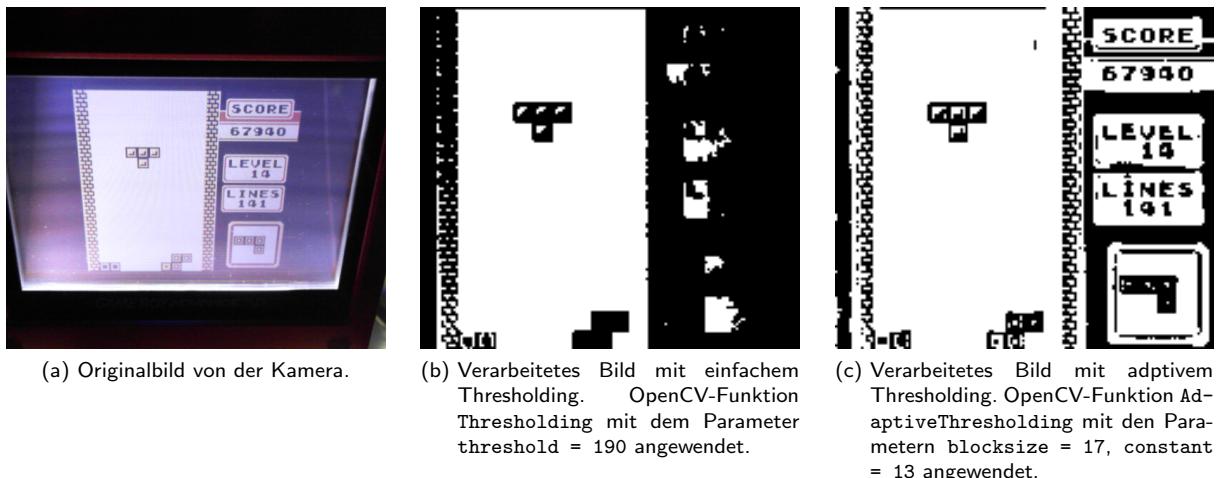


Abbildung 3.12.: Vergleich der beiden Thresholding-Verfahren.

3.5.2.3. Opening

Da die Spielsteine in der Game-Boy-Variante nicht alle ausgefüllt sind, ist es sehr schwierig, diese zu detektieren. Kommt noch ein Rauschen dazu, gehen dünne Linien bei der Binarisierung schnell verloren.

Es braucht also noch einen weiteren Filter, um die Tetriminos auszufüllen. Hier haben sich die morphologischen Operatoren als sehr hilfreich erwiesen – genauer: die Opening-Funktion (siehe Abb. 3.13). Als Kernel ist ein Quadrat mit Länge 7 optimal. Diese Länge reicht aus, einen Block auszufüllen (Innenabstand 6 Pixel), ist aber gerade noch zu klein, um zwei auseinanderliegende Blöcke fälschlicherweise zu verbinden (8 Pixel Abstand).

Zu viel Rauschen kann bei morphologischen Operatoren zu Fehlern führen. Es wurden Versuche mit der OpenCV-Funktion `FastNlMeansDenoisingColored` unternommen. Diese ist allerdings nicht geeignet, da der Rechenaufwand viel zu hoch ist. Dank dem adaptiven Thresholding kann das Rauschen genügend reduziert werden, sodass morphologische Operatoren stabile Resultate liefern.

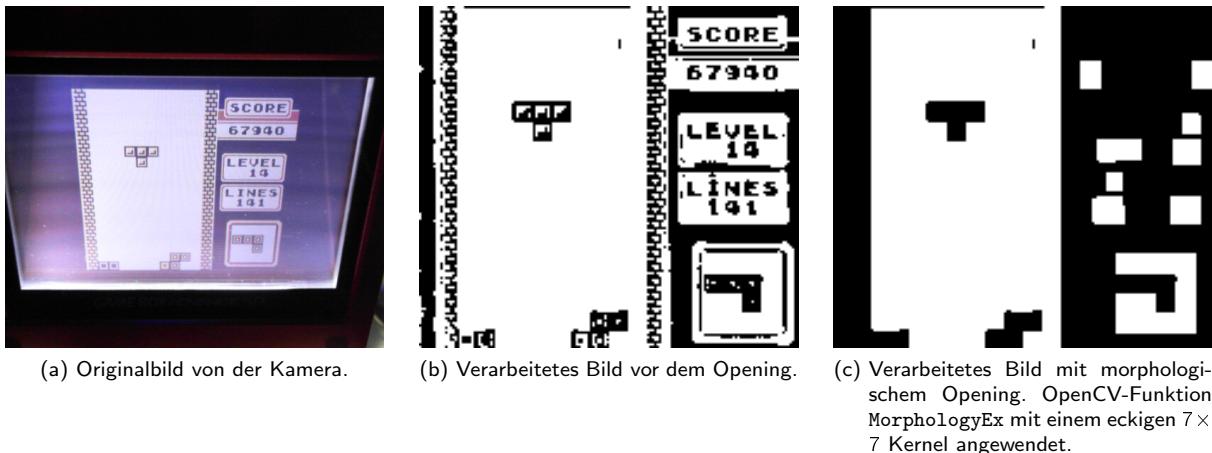


Abbildung 3.13.: Morphologisches Opening zum Ausfüllen der Blöcke im Spielfeld.

3.5.2.4. Voraussetzungen

Folgende Punkte müssen beachtet werden, damit später eine robuste Bildanalyse gewährleistet werden kann:

Genaue Kalibrierung

Keine Reflexionen oder Spiegelungen auf dem Display Kamera schräg zur optischen Achse positionieren, damit sich die Kamera nicht im Display spiegelt.

Gute Beleuchtung Display mit Hintergrundbeleuchtung oder eine indirekte externe Beleuchtung verwenden.

Abschirmung Wenn möglich, das Display von externen Lichteinflüssen abschirmen.

3.5.3. Emulator

Um den Agenten isoliert und trotzdem unter realen Bedingungen testen zu können, wurde ein Game-Boy-Emulator eingesetzt. Der Emulator – ursprünglich entwickelt von Michael Birken im Jahr 2008 – wurde so angepasst, dass er mit der Engine kompatibel ist. Der Emulator emuliert eine Variante des Prozessors Zilog Z80[22]. Dieser Prozessor ist auch im echten Game Boy verbaut.

Der Emulator von Michael Birken ist eine Klassenbibliothek – geschrieben in C# – und kann so praktisch in den eigenen Code integriert werden. Per API können also Tastendrücke ausgeführt und das Display-Bild ausgelesen werden.

Der grosse Vorteil des Emulators ist es, dass die KI unabhängig vom Roboter entwickelt und getestet werden konnte. Ebenfalls dank des Emulators konnten tausende von Spielen simuliert werden, um Daten über den Zufallsgenerator des Spiels gewinnen zu können (siehe Abschnitt 3.6.1.2). Der Emulator erlaubte außerdem die Verifizierung der Spiellogik von Tetris (siehe Abschnitt 3.6.1).

Da der Bildschirm des Emulators fehlerfrei ausgelesen werden kann, konnte die Engine auch bereits ohne eine funktionierende Bildverarbeitung getestet werden.



3.6. Agent

Der Agent ist die künstliche Intelligenz, die das Spiel Tetris selbstständig spielen kann. Die Aufgaben des Agenten sind die Bildanalyse, das Berechnen des Spielzugs und schlussendlich die Ausführung der Aktionen. Jede dieser Aufgaben wird im Folgenden in einem oder mehreren Abschnitten behandelt.

3.6.1. Spiellogik

Der Agent muss Kenntnis der Spiellogik haben, damit er weiß, welche Auswirkungen eine bestimmte Aktion auf den Spielstand hat.

Die Game-Boy-Variante von Tetris entspricht nicht der Tetris Guideline (siehe Abschnitt 2.2.3)⁵. Viele öffentlich verfügbare Algorithmen sind entsprechend den Richtlinien optimiert worden und können demnach nur mit Anpassungen oder mit Leistungseinbussen gebraucht werden.

Als Folge wurde die Spiellogik für diese Arbeit selber implementiert. Für eine eigene Implementierung spricht auch, dass der Code nach Bedarf optimiert werden kann. Eine hohe Performance ist für die Spiellogik deshalb notwendig, da die Breitensuche der KI davon exzessiven Gebrauch macht. Weitere Informationen dazu im Abschnitt 3.6.3.

3.6.1.1. Levels und Geschwindigkeit

Der Großteil des Spiels ist deterministisch (siehe Abschnitt 2.2.4). Viele Informationen über den Spielstand können also implizit bestimmt werden. Dafür ist aber detailliertes Wissen über die Spiellogik notwendig. Beispielsweise lässt sich berechnen, wie weit ein Spielstein gefallen ist, wenn man die ursprüngliche Position und den Level kennt. Den Level kann man berechnen, wenn man die Anzahl komplettierter Linien kennt.

Die Fallgeschwindigkeiten pro Level sind bekannt[8] (siehe Tabelle 3.1). Im sogenannten *Heart Mode* des Spiels entspricht die Geschwindigkeit immer dem aktuellen Level +10.

Tabelle 3.1.: Fallgeschwindigkeiten der Spielsteine pro Level.

Level	Frames pro Linie	Millisekunden pro Linie	Linien pro Sekunde
0	53	887.3	1.1
1	49	820.4	1.2
2	45	753.4	1.3
3	41	686.4	1.5
4	37	619.5	1.6
5	33	552.5	1.8
6	28	468.8	2.1
7	22	368.3	2.7
8	17	284.6	3.5
9	11	184.2	5.4
10	10	167.4	6.0
11	9	150.7	6.6
12	8	133.9	7.5
13	7	117.2	8.5
14	6	100.5	10.0
15	6	100.5	10.0
16	5	83.7	11.9
17	5	83.7	11.9
18	4	67.0	14.9
19	4	67.0	14.9
20	3	50.2	19.9

⁵Beispielsweise entspricht die Größe des Spielfeldes mit 10×18 Blöcken nicht dem Standard. Auch ein Harddrop ist in Game Boy Tetris nicht möglich.

3.6.1.2. Zufallsgenerator

Die Tetris Guideline definiert, wie ein Tetris-Zufallsgenerator implementiert sein sollte[5]. Die Auftretenswahrscheinlichkeit für die einzelnen Tetriminos ist laut dieser Richtlinie gleichverteilt, wenn auch nicht wirklich zufällig⁶. Die Entwickler von Game Boy Tetris haben allerdings einen speziellen Zufallsgenerator implementiert, welcher nicht alle Tetriminos mit der gleichen Wahrscheinlichkeit generiert[8]. Diesen Umstand kann man sich zu Nutze machen, um mit einer probabilistischen Suche die KI zu verbessern (siehe dazu Abschnitt 3.6.3.2).

Um die Wahrscheinlichkeiten der Tetriminos zu messen, wurden mithilfe der Engine über 18'000 Spielsteine auf dem Game-Boy-Emulator gespielt. Dabei wurden die generierten Tetriminos gezählt und ihre Wahrscheinlichkeiten konnten anschliessend statistisch ausgewertet werden. Das Resultat ist eine deutlich ungleichmässige Wahrscheinlichkeitsverteilung. Siehe dazu Tabelle 3.2. Die Wahrscheinlichkeiten entsprechen aber nicht dem Zufallsgenerator, wie er im Internet beschrieben ist[8].

Tabelle 3.2.: Wahrscheinlichkeiten der Tetriminos.

Tetrimino	Wahrscheinlichkeit
S	19.9%
T	17.4%
J	15.2%
O	14.3%
I	11.6%
L	11.1%
Z	10.5%

Interessant ist auch, dass die Wahrscheinlichkeitsverteilung sehr unregelmässig ist, wenn man jeweils die beiden letzten Tetriminos betrachtet. Die vorhergehenden Spielsteine haben also einen Einfluss auf den neu erzeugten Spielstein (siehe Abb. 3.14).

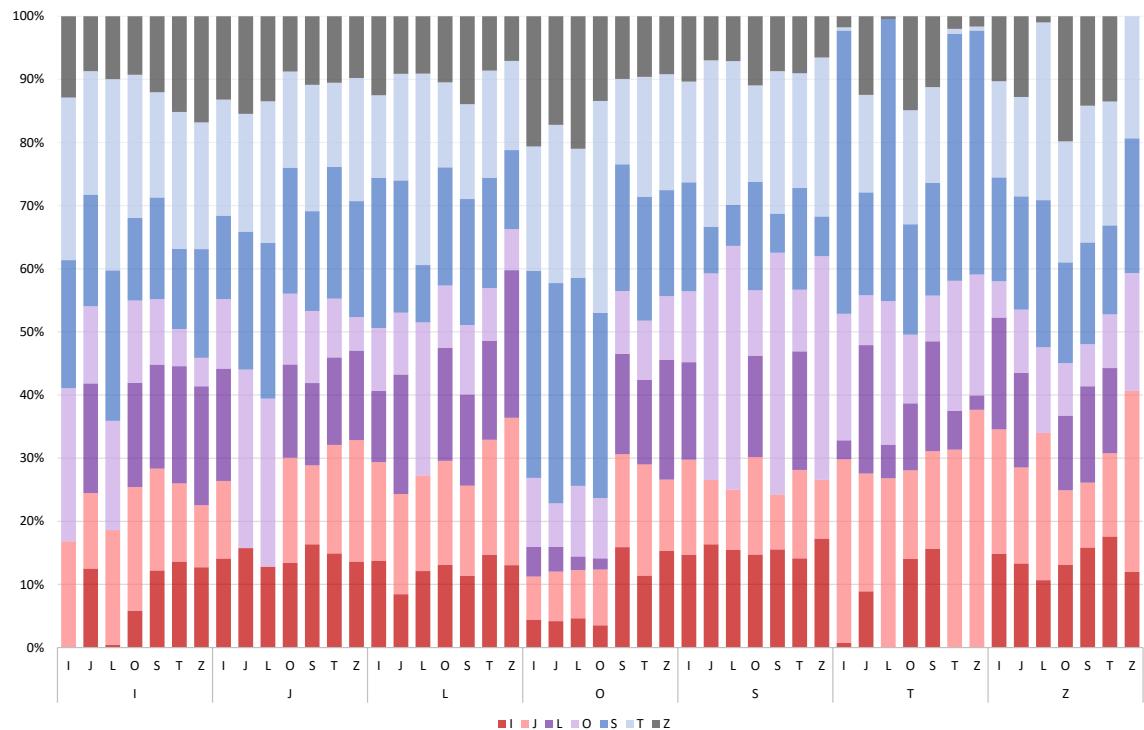


Abbildung 3.14.: Wahrscheinlichkeiten der Tetriminos in Abhängigkeit der zwei vorhergehenden Tetriminos. Die obere Zeile der x-Achsenbeschriftung gibt das letzte Tetrimino an, die untere Zeile das vorletzte Tetrimino.

⁶Der Random Generator der Tetris Guideline liefert periodisch alle sieben Tetriminos hintereinander, jedoch in zufälliger Permutation[5]. Zu einem gewissen Grad lässt sich also vorhersagen, welcher Spielstein als nächstes kommen wird.



3.6.1.3. Effiziente Datenstrukturen

Um eine maximale Performance zu erreichen, wurden gewisse Datenstrukturen auf Effizienz getrimmt.

Das Spielfeld beispielsweise ist als Array von Integer-Werten implementiert. Jede Spalte wird dabei von einem Wert representiert. Die einzelnen Bits stellen freie oder belegte Blöcke dar. Auf diese Weise können in $\mathcal{O}(1)$ (also in konstanter Zeit und unabhängig von der Höhe des Spielfelds) gewisse Merkmale einer Spalte abgefragt werden. Dies ermöglicht eine Lookup-Tabelle mit dem Spaltenwert als Schlüssel. Konkrete Merkmale sind beispielsweise Höhe einer Spalte oder Anzahl Löcher.

Effizient muss auch die Position eines Spielsteins nach dem Drop berechnet werden können. Dafür wurde für jedes Tetrimino und jede mögliche Rotation die untere Kontur (siehe Abb. 3.15) vorberechnet und deren relative Position abgespeichert. Pro Konturblock wird die Spaltenhöhe abgefragt und die Distanz berechnet. Die Dropdistanz ist das Minimum aller Distanzen unter den Konturblöcken.

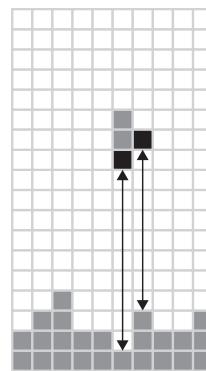


Abbildung 3.15.: Berechnung der Drop-Distanz anhand der unteren Kontur des Spielsteins (schwarze Blöcke).

3.6.2. Bildanalyse

Der Agent erhält ein bereits verarbeitetes Bild von der Engine (siehe Abschnitt 3.5.2). Um im nächsten Schritt die Aktionen bestimmen zu können, muss zuerst der aktuelle Spielstand bestimmt werden. Die Kamera bzw. die Bildverarbeitung ist nicht zu 100% zuverlässig. Es muss also eine gewisse Fehlertoleranz berücksichtigt werden.

Grundsätzlich geht der Agent von einem deterministischen System aus. Jede Aktion hat klar definierte Auswirkungen, woraus sich auch der neue Spielstand ableiten lässt. Es gibt folgende Ausnahmen:

- Der aktuelle Spielstein muss zu Beginn des Spiels ermittelt werden, da dieser zufällig erscheint.
- Der nächste Spielstein muss nach jeder Runde ermittelt werden, da dieser auch zufällig erscheint.

Versagt der Agent in einem dieser Punkte, entsteht ein inkonsistenter Spielstand und jede weitere Aktion vergrößert den Fehler nur. Dies muss also unbedingt verhindert werden. Verschiedene Massnahmen sind denkbar:

Sampling Der Agent analysiert immer mehrere Bilder, statt nur ein einziges. Dadurch wird die Wahrscheinlichkeit verringert, einen falschen Spielstein anzunehmen. Allerdings geht beim Sampling auch wertvolle Zeit verloren. Mehr dazu in Abschnitt 3.6.2.3.

Verifizieren des Spielfeldes Ein Spielstein kann auch mal falsch platziert werden. Entweder weil ein Tastendruck nicht ausgeführt wird, oder weil die Zeit nicht reicht, den Spielstein korrekt zu positionieren. In diesem Fall muss der Agent darauf reagieren können. Mehr dazu in Abschnitt 3.6.2.4.

Verifizieren jeder Aktion Um zu verifizieren, ob eine Aktion ausgeführt wurde, kann nach jeder Aktion eine Bildanalyse durchgeführt werden. So kann jede einzelne Aktion überprüft und bei Bedarf wiederholt werden. Dieses Verfahren hat sich als ineffizient herausgestellt (siehe Abschnitt 3.6.4).

Zum Ermitteln der Spielsteine wurden zwei verschiedene Ansätze implementiert: die blockbasierte Methode (siehe Abschnitt 3.6.2.1) und die spielsteinbasierte Methode (siehe Abschnitt 3.6.2.2). Beide Verfahren benötigen einen Suchbereich, um eine effiziente Bildanalyse garantieren zu können. Der Agent kann die Position des zu ermittelnden Spielsteins abhängig von der Zeit berechnen und den Suchbereich davon ableiten.

3.6.2.1. Blockbasierte Methode

Bei diesem Ansatz werden nicht direkt die Spielsteine erkannt, sondern das Bild wird zuerst auf ein Raster von 8×8 Pixel reduziert. Dank dem morphologischen Opening (siehe Abschnitt 3.5.2.3) sind die einzelnen Felder dieses Rasters mehrheitlich weiß oder schwarz. Innerhalb des Spielfeldes oder des Vorschaubildes bedeutet schwarz „Feld ist ein Block eines Spielsteins“. Weiß bedeutet „Feld ist frei“.

Um ein Tetrimino im Bild zu finden, müssen jeweils vier Blöcke überprüft werden. Jeder Block kann mit einer gewissen Wahrscheinlichkeit p_{b_k} erkannt werden. p_{b_k} entspricht 1 – dem mittleren Helligkeitswert der 8×8 Pixel. Die Wahrscheinlichkeit für das Tetrimino p_t ist der Mittelwert aller Wahrscheinlichkeiten p_b .

$$p_t = \frac{1}{4} \sum_{k=1}^4 p_{b_k}$$

Es wird ein Schwellenwert t definiert, der erreicht werden muss, damit ein Tetrimino als erkannt gilt.

$$t \leq p_t$$

Das Verfahren ist sehr effizient und robust gegenüber leichten Kalibrierungsfehlern. Die Ergebnisse dazu sind unter Abschnitt 4.2 zu finden.

3.6.2.2. Spielsteinbasierte Methode

Der spielsteinbasierte Ansatz entspricht einem Template Matching. Als Pattern wird das gesuchte Tetrimino verwendet. Gemacht werden nur die Konturen und gewisse Details, deswegen wird zusätzlich eine Maske zum Template verwendet (siehe Abb. 3.16). Das Template wird jeweils in einer Region von 3×3 Pixeln um die Suchposition gematcht. Dies macht das Verfahren robuster gegenüber leichten Kalibrierungsfehlern.

Wie bei der blockbasierten Analyse wird mit Wahrscheinlichkeiten und einem Schwellenwert gearbeitet.

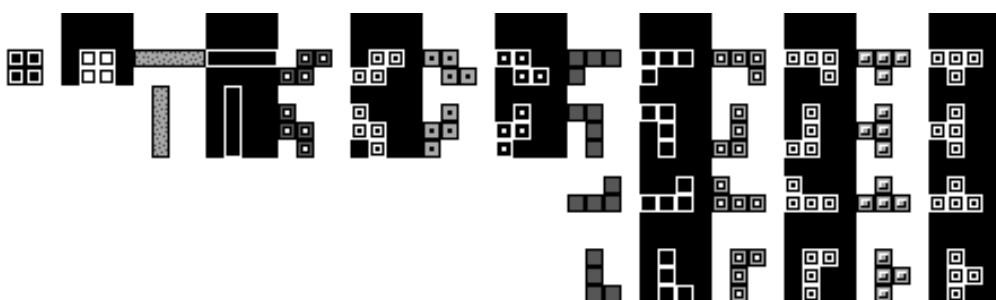


Abbildung 3.16.: Die Templates inklusive Masken sämtlicher Spielsteine und ihrer zulässigen Rotationen.

Die spielsteinbasierte Analyse ist fehleranfällig bei Helligkeitsveränderungen und weniger effizient als die blockbasierte Analyse. Deshalb wurde dieses Verfahren verworfen. Die Ergebnisse dazu sind unter Abschnitt 4.2 zu finden.



3.6.2.3. Sampling

Der Agent analysiert jeweils mehrere Bilder, statt nur eines. Dies verringert die Wahrscheinlichkeit, ein falsches Tetrimino zu erkennen. Hingegen steigt die Zeit für die Bildanalyse linear mit der Anzahl Samples n an. Je nach Framerate geht dabei viel Zeit verloren (siehe Abschnitt 3.2.6.2). Die Tabelle 3.3 zeigt die reduzierte Fehlerrate anhand verschiedener Sampling-Größen n .

Die Sampling-Größe n gibt an, wie viele Bilder maximal analysiert werden. Es braucht $\frac{n+1}{2}$ gleiche Samples, damit das Sampling früher beendet wird. Bei $n = 5$ wird das Sampling also bereits nach drei analysierten Bildern beendet, wenn die ersten drei Bilder das gleiche Tetrimino liefern. Für diese Arbeit hat sich die Sampling-Größe $n = 3$ als vernünftigen Wert erwiesen.

Tabelle 3.3.: Auswirkungen auf die Fehlerrate e in Abhängigkeit der Sampling-Größe n .

Fehlerrate e	Fehlerraten mit Sampling		
	$n = 1$	$n = 3$	$n = 5$
0%	0.000%	0.000%	0.000%
1%	1.000%	0.030%	0.001%
2%	2.000%	0.118%	0.008%
3%	3.000%	0.265%	0.026%
4%	4.000%	0.467%	0.060%
5%	5.000%	0.725%	0.116%
10%	10.000%	2.800%	0.856%

3.6.2.4. Verifizieren des Spielfeldes

Das Spielfeld wird nicht in jeder Runde komplett neu ermittelt, da dies zu fehleranfällig wäre und schlecht validiert werden kann. Dies ist aber auch gar nicht nötig, um den internen Spielstand zu verifizieren. Es reicht, den „Horizont“⁷ zu überprüfen.

Im Beispiel von Abb. 3.17 wurde der Spielstein falsch platziert und dies merkt der Agent, wenn er den Horizont (schwarz markierte Blöcke) auswertet.

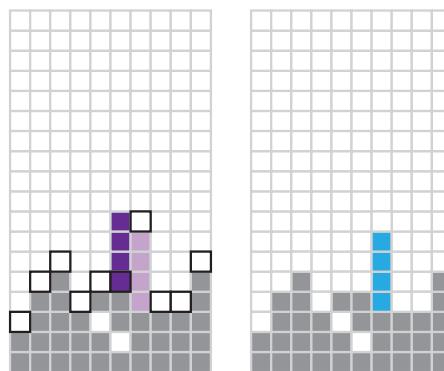


Abbildung 3.17.: Verifizierung des Spielfeldes des Kamerabildes (links) anhand des internen Spielstandes (rechts).

Wird ein Übereinstimmungsfehler festgestellt, muss gezwungenermaßen das gesamte Spielfeld neu ermittelt werden.

3.6.3. Breitensuche

Da das Spiel NP-vollständig ist (siehe Abschnitt 2.2.4), muss die optimale Lösung angenähert werden. Dafür wird der Lösungsansatz einer Breitensuche kombiniert mit einer Bewertungsfunktion verwendet.

⁷Gemeint sind alle Blöcke direkt über den obersten Blöcken jeder Spalte.

Die Breitensuche untersucht alle gültigen Spielzüge, welche mit dem aktuellen und dem nächsten Tetrimino möglich sind. Jeder Blattknoten des Suchbaumes wird anhand einer Bewertungsfunktion (siehe Abschnitt 3.6.3.1) bewertet. Die Suche gibt den Spielzug zurück, welcher zur bestbewerteten Spielstellung führt. Der Spielzug wird zum Schluss noch in Aktionen – also Tastendrücke – übersetzt. Der Agent führt diese anschliessend aus (siehe Abschnitt 3.6.4).

Der Agent führt die Breitensuche bis zur Tiefe 2 aus (siehe Abb. 3.18). In der ersten Stufe werden alle gültigen Posen, sprich Position und Orientierung, für das aktuelle Tetrimino untersucht. In der zweiten Stufe die des Vorschau-Tetriminos. Die Tabelle 3.4 zeigt die möglichen Posen aller Tetriminos. Der Verzweigungsfaktor des Suchbaumes ist also im Mittel $\frac{162}{7} \approx 23$. Bei einer Suchtiefe von 2 ergibt das $(\frac{162}{7})^2 \approx 536$ zu bewertende Spielstellungen.

Das Spiel erlaubt es, Tetriminos seitwärts in eine Lücke zu befördern. So können Löcher später elegant wieder korrigiert werden. Um die KI simpel zu halten, wurde auf diese Möglichkeit bewusst verzichtet (siehe Abschnitt 1.6). Der Agent kann Tetriminos nur positionieren und anschliessend fallen lassen.

Tabelle 3.4.: Alle möglichen Tetrimino-Posen auf einem Spielfeld mit Breite 10.

Tetrimino	Rotationen	gültige Posen
O	1	9
I	2	17
S	2	17
Z	2	17
L	4	34
J	4	34
T	4	34

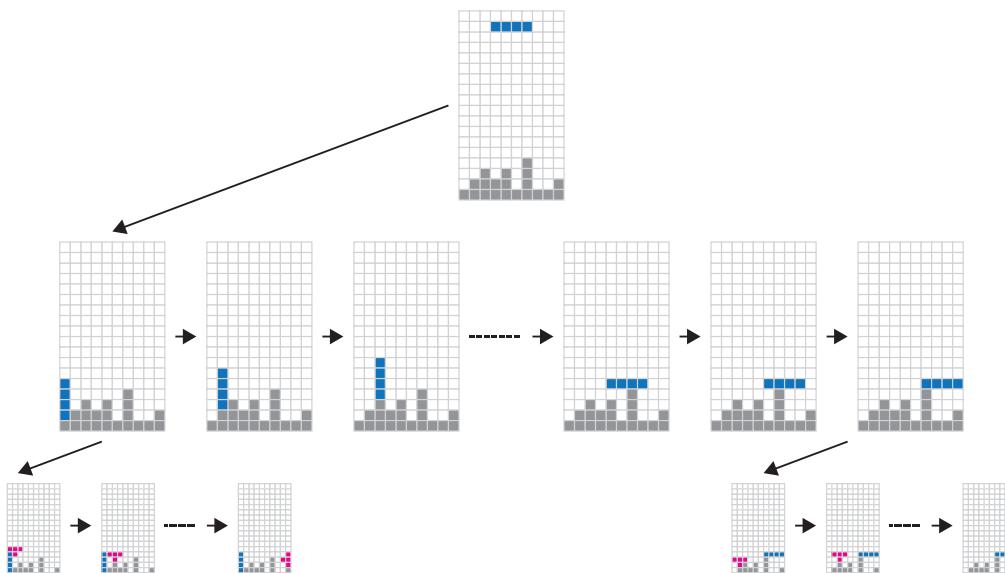


Abbildung 3.18.: Vereinfachte Darstellung der Breitensuche mit Tiefe 2 und den Spielsteinen I und T.

3.6.3.1. Bewertungsfunktion

Die Bewertungsfunktion macht eine Aussage über die Güte eines Spielstandes. Dies ist nur eine *Annäherung* an den realen Wert⁸.

Für die Bewertung der Spielstellungen wurden zwei bestehende Bewertungsfunktionen implementiert und verglichen. Die erste stammt von Yiyuan Lee. Er hat diese 2013 in seinem Blog gepostet[28]. Die zweite wurde von Max Bergmark entwickelt und 2015 veröffentlicht[21]. Beide Funktionen basieren auf folgenden Merkmalen⁹:

⁸Wollte man den realen Wert berechnen, müsste man die Breitensuche mit einer quasi unendlichen Tiefe ausführen, um sagen zu können, welche Spielstände mit welcher Wahrscheinlichkeit zu einer Niederlage führen können.

⁹Diese Liste ist überhaupt nicht abschliessend. Die Wahl der Merkmale ist einer der wichtigsten Schritte beim Design einer guten Bewertungsfunktion. Über 30 unterschiedliche Merkmale wurden in Publikationen zum Thema Tetris-KI erwähnt[31].



- Höhe des gefüllten Spielfeldes
- Komplettierte Linien
- Löcher
- Unebenheit

Yiyuan Lee berechnet in seiner Bewertungsfunktion für alle Merkmale je eine unabhängige Kennzahl und kombiniert diese mittels einer gewichteten Summe.

$$v(S) = a \cdot \text{AggregateHeight}(S) + b \cdot \text{CompleteLines}(S) + c \cdot \text{Holes}(S) + d \cdot \text{Bumpiness}(S)$$

Dabei ist $a = -0.510066$, $b = 0.760666$, $c = -0.356630$ und $d = -0.184483$. Die Faktoren hat Yiyuan Lee mittels genetischem Algorithmus optimiert.

Max Bergmark konzentriert sich auf die Merkmale Löcher und Unebenheit und gewichtet die Merkmale in Abhängigkeit der Höhe.

Pro Bewertungsfunktion wurden 100 Spiele simuliert und die komplettierten Linien sowie die erzielte Punktzahl aufgezeichnet. Die Resultate sind im Abschnitt 4.3 zu finden. Die Bewertungsfunktion von Yiyuan Lee schneidet besser ab und wurde deshalb für die KI dieser Arbeit eingesetzt.

Abb. 3.19 zeigt zwei Möglichkeiten, die beiden Spielsteine S und I zu platzieren. Darunter ist jeweils der konkrete Wert der Bewertungsfunktion angegeben. Das Merkmal der Unebenheit ist in diesem Beispiel entscheidend für die Wahl der „optimalen“ Platzierung.

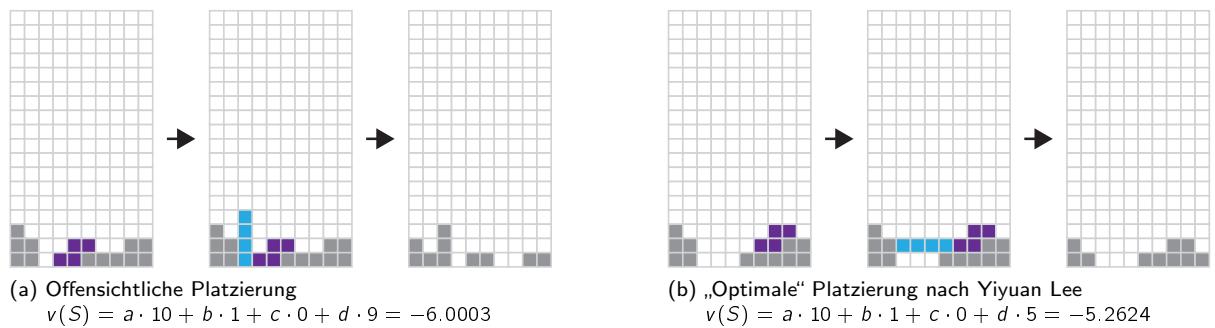


Abbildung 3.19.: Unterschiedliche Möglichkeiten, die Spielsteine S und I zu platzieren. Inklusive Bewertung des resultierenden Spielstandes.

3.6.3.2. Vorhersage

Um die Resultate der Breitensuche zu verbessern, kann die Suchtiefe erhöht werden. Da aber keine Informationen für das übernächste und alle weiteren Tetriminos zur Verfügung stehen, muss man einen probabilistischen Ansatz wählen. Man untersucht also die möglichen Züge für jedes der sieben Tetriminos. Pro Tetrimino wird der bestmögliche Wert mit der Auftretenswahrscheinlichkeit des Tetriminos verrechnet. Die Auftretenswahrscheinlichkeiten wurden experimentell ermittelt, wie in Abschnitt 3.6.1.2 beschrieben.

Der Agent kann auf diese Weise weiter in die Zukunft blicken. Der Rechenaufwand steigt allerdings exponentiell für jede weitere Stufe n der Vorhersage an: 162^n (siehe Tabelle 3.5). Ab der zweiten Stufe der Vorhersage dauert die Berechnung bereits mehrere Sekunden und ist für einen Echtzeit-Bot nicht mehr brauchbar. In den höheren Levels ist bereits eine Stufe der Vorhersage nicht mehr schnell genug.

Tabelle 3.5.: Zeitkomplexität der Breitensuche in Abhängigkeit der Suchtiefe n .

<i>Suchtiefe n</i>	<i>Dauer in ms</i>	<i>Dauer in h</i>
1	0.1	$\ll 0$
2	0.3	$\ll 0$
3, 1 Stufe der Vorhersage	46.7	$\ll 0$
4, 2 Stufen der Vorhersage	7178.8	$\ll 0$
5, 3 Stufen der Vorhersage	1162964.1	0.3
6, 4 Stufen der Vorhersage	188400188.4	52.3

3.6.3.3. Gentischer Algorithmus

Mittels genetischem Algorithmus wurde versucht, die Bewertungsfunktion von Yiyuan Lee durch Optimierung der Gewichtsfaktoren noch weiter zu verbessern. Vorgegangen wurde annäherungsweise nach Yiyuan Lees Methode[28]. Abweichungen von seinem Verfahren sind in diesem Abschnitt beschrieben.

Die Populationsgrösse entspricht 100 Individuen. Die Wahrscheinlichkeit für ein Crossing-over beträgt 30%, die Mutationswahrscheinlichkeit ist 2% pro Gen. Tournament Selection wurde zur Wahl der Eltern für das Crossing-over verwendet. Als Fitnessfunktion wurden jeweils 100 Spiele mit je 100 Spielsteinen durchgeführt, die Anzahl komplettierter Linien entspricht der Fitness des Individuums. Gemacht wurden 5 Durchläufe mit je 100 Generationen (siehe Resultate in Tabelle 3.6).

Tabelle 3.6.: Resultate der Optimierung der Gewichtsfaktoren mittels genetischem Algorithmus.

<i>Settings</i>	<i>Factor Height</i>	<i>Factor Lines</i>	<i>Factor Holes</i>	<i>Factor Bumpiness</i>	<i>Result Lines</i>
Yiyuan Lee	-0.51007	0.76067	-0.35663	-0.18448	3536
GA 1	-0.54039	0.27020	-0.79558	-0.04503	569
GA 2	-0.76359	0.37331	-0.50906	-0.13575	1124
GA 3	-0.56190	0.65337	-0.50310	-0.06534	677
GA 4	-0.52732	0.79098	-0.13877	-0.27754	2501
GA 5	-0.90616	0.14828	-0.37894	-0.11533	854

Die Bemühungen ergaben keine Verbesserungen der ursprünglichen Gewichtsfaktoren von Yiyuan Lee. Gründe dafür können sein: die geringe Populationsgrösse, zu wenig Generationen, zu ungenaue Fitnessfunktion oder vorzeitige Konvergenz.

3.6.4. Befehlsausführung

Nach der Breitensuche kennt der Agent die Aktionen, die auszuführen sind. Er gibt die Aktionen als Tastendrücke der Engine weiter, welche diese wiederrum an die Hardware weiterleitet.

Die Aktoren sind nicht zu 100% zuverlässig, da beim Drücken der Game-Boy-Tasten immer eine gewisse Fehleranfälligkeit vorhanden ist. Es braucht also eine Sicherungsschicht. Sie erkennt Fehler und kann Aktionen in solchen Situationen wiederholen bzw. den Spielstand korrigieren¹⁰.

Es wurden zwei Ansätze ausprobiert: die pessimistische und die optimistische Befehlsausführung. Sie werden in den Abschnitten 3.6.4.1 und 3.6.4.2 erläutert.

3.6.4.1. Pessimistische Befehlsausführung

Jede Aktion wird nach dem Ausführen überprüft. Der Befehl kann n mal wiederholt werden. So gewinnt man Zuverlässigkeit, verliert jedoch Zeit. Der Agent muss für die Verifizierung jeder Aktion ein neues Bild analysieren.

Dieses Verfahren hat sich für einen Echtzeit-Bot als zu ineffizient herausgestellt.

¹⁰Die Engine kann die Ausführung der Aktionen nicht überprüfen, da dafür Wissen über das konkrete Spiel notwendig ist. Der Agent muss also diese Aufgabe übernehmen.



3.6.4.2. Optimistische Befehlausführung

Bei diesem Ansatz wird von beinahe fehlerfrei funktionierenden Aktoren ausgegangen. Der Agent führt die Aktionen aus, ohne diese nachzuprüfen. Bevor aber der nächste Spielstein platziert wird, wird zuerst das Spielfeld verifiziert und wenn nötig neu eingelesen. Für das Verifizieren wird nur der „Horizont“ des Spielfeldes betrachtet (siehe Abschnitt 3.6.2.4).

Dieser Ansatz hat sich für diese Arbeit bewährt und als sehr effizient herausgestellt.

3.6.4.3. Parallele Befehlausführung

Da der Roboter pro Taste einen Aktor hat, kann er mehrere Aktionen gleichzeitig ausführen. So können Rotation und Translation parallel ausgeführt werden. Dadurch kann die Zeit von bis zu zwei Tastendrücken gewonnen werden¹¹.

3.6.5. Zustandsautomat

Der Agent ist als endlicher Automat implementiert (siehe Abb. 3.20). Je nach Aufgabe, die der Agent auszuführen hat, wechselt der Automat in einen anderen Zustand. Im Folgenden sind die möglichen Zustände kurz erläutert:

Ready Der Agent wartet bis das Spiel beginnt.

Analyze Der Agent prüft das Spielfeld auf Fehler, analysiert die Kamerabilder und macht eine Breitensuche aller möglichen Spielzüge.

Execute Der Agent führt die Aktionen aus, welche aus dem berechneten Spielzug resultieren.

Game over Der Agent ist inaktiv.

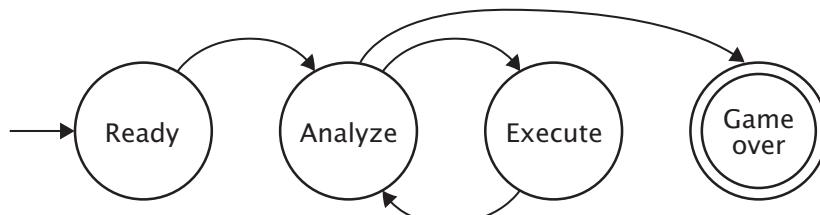


Abbildung 3.20.: Zustandsdiagramm des Agenten.

3.6.6. Zeitkomplexität

Dieser Abschnitt behandelt theoretische Überlegungen zur Obergrenze der möglichen zeitlichen Optimierungen¹².

Die drei wichtigen Prozesse *Bilderaffassung und -analyse*, *Breitensuche* und *Befehlausführung* sind jeweils voneinander abhängig, müssen also zwingend sequentiell stattfinden. Die benötigte Gesamtzeit pro Spielzug Δt_{total} lässt sich also folgendermassen berechnen:

$$\Delta t_{\text{total}} = \Delta t_{\text{analysis}} + \Delta t_{\text{search}} + \Delta t_{\text{execution}}$$

Die Dauer für die Bilderaffassung und -analyse ist abhängig von der Anzahl Samples n , der Framerate f , der Zeit für die Analyse des aktuellen Spielsteins Δt_{cp} und der Zeit für die Analyse des nächsten Spielsteins Δt_{np} ¹³.

¹¹Ein Spielstein muss höchstens zwei Mal rotiert werden, da Rotationen sowohl im als auch gegen den Uhrzeigersinn möglich sind. Bei paralleler Ausführung können also maximal die Tastendrücke der Rotation gespart werden.

¹²Vorausgesetzt, der Roboter wurde anhand der hier beschriebenen Methode gebaut. Bei abweichender Implementierung gelten diese Überlegungen unter Umständen nicht mehr.

¹³Die Formel stimmt nur, wenn man davon ausgeht, dass das Resultat aller gesampelten Bilder gleich ist. Im anderen Fall müsste man mit n rechnen, statt mit $\frac{n-1}{2}$

$$\Delta t_{\text{analysis}} = \frac{n-1}{2} \cdot \max \left(\frac{1}{f}, \Delta t_{\text{cp}} + \Delta t_{\text{np}} \right)$$

Die Dauer für die Breitensuche ist direkt abhängig vom Verzweigungsfaktor b , den Suchtiefen k (bekannte Spielsteine) und p (Tiefe der Vorhersage) und der Zeit Δt_{score} zum Bewerten eines Spielstandes.

$$\Delta t_{\text{search}} = b^k (7b)^p \Delta t_{\text{score}} = b^{(k+p)} 7^p \Delta t_{\text{score}}$$

Die Dauer für die Befehlsausführung hängt ab von der Anzahl Rotationsbefehlen c_{rot} , der Anzahl Translationsbefehlen c_{trans} , der Beschaltungszeit Δt_{hit} und der Rückstellzeit $\Delta t_{\text{release}}$ der Aktoren.

$$\Delta t_{\text{execution}} = \max(c_{\text{rot}}, c_{\text{trans}}) \cdot (\Delta t_{\text{hit}} + \Delta t_{\text{release}})$$



4. Messungen und Ergebnisse

Dieses Kapitel beschreibt die Messverfahren für die zu Beginn definierten Messgrößen (siehe Abschnitt 1.4) und führt die Ergebnisse auf. Die Resultate haben keine statistische Signifikanz.

4.1. Performance der Bildverarbeitung

4.1.1. Messverfahren

Die Performance der Bildverarbeitung ist definiert durch die Zeit, die benötigt wird, um sämtliche benötigten Operatoren auf ein Kamerabild anzuwenden. Damit die Resultate miteinander vergleichbar sind, wurden immer Bilder mit einer Auflösung von 640×480 verwendet.

Pro Messung wurden jeweils 1000 Bilder verarbeitet und die per PC-Timer gemessene Zeit gemittelt.

Verwendetes Gerät für die Messungen: siehe Abschnitt 3.3.1.3.

4.1.2. Ergebnisse

Die Bildverarbeitung pro Bild kann auf dem Testgerät unter einer Millisekunde ausgeführt werden. Der rechenintensivste Operator ist die Entzerrung, sie nimmt 61% der Rechenzeit in Anspruch.

Für Messdaten siehe Tabelle 4.1.

Tabelle 4.1.: Messergebnisse für die Bildverarbeitungszeit in Millisekunden (pro Operator und total).

	Zeit für 1000 Bilder (ms)	Zeit für 1 Bild (ms)	Prozentualer Anteil
<i>Entzerrung</i>	275	0.275	61%
<i>Binarisierung</i>	123	0.123	27%
<i>Opening</i>	52	0.052	12%
<i>Total</i>	451	0.451	100%

4.2. Zuverlässigkeit der Bildanalyse

4.2.1. Messverfahren

Die Zuverlässigkeit der Bildanalyse ist definiert durch den prozentualen Anteil der korrekt erkannten Tetris-Spielsteine auf bereits verarbeiteten Kamerabildern (Auflösung jeweils 160×144 Pixel).

Getestet wurde jeweils die Zuverlässigkeit dieser drei Methoden:

- Erkennen des aktuellen, unbekannten Spielsteins
- Erkennen des aktuellen, bekannten Spielsteins
- Erkennen des nächsten Spielsteins

Pro Messung und Methode wurden jeweils 40 unterschiedliche Kamerabilder verwendet. Die sichtbaren Spielsteine wurden zuvor manuell ausgewertet. Um die Ergebnisse besser differenzieren zu können, wurden zwei Störfaktoren eingeführt¹:

Rauschen Künstliches Rauschen (schwarz-weiss) wird mittels halbtransparenter Überblendung (Faktor 0 bis 0.8) über das Bild gelegt.

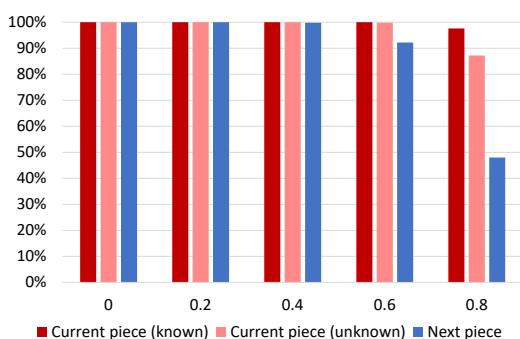
Fehlkalibrierung Die Keypoints für die Entzerrung des Bildes werden absichtlich falsch platziert. Mit einer zufälligen Abweichung pro Keypoint von bis zu 8 Pixel.

Verwendetes Gerät für die Messungen: siehe Abschnitt 3.3.1.3.

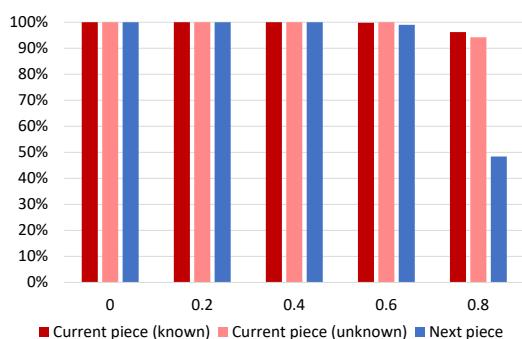
4.2.2. Ergebnisse

Beide Verfahren erreichen eine Zuverlässigkeit von 100%, sofern der Rauschfaktor nicht grösser als 0.2 und der Kalibrierungsfehler nicht grösser als 1 Pixel ist. Die blockbasierte Methode ist zuverlässiger bei Fehlkalibrierung (siehe Abb. 4.2), die spielsteinbasierte Methode ist zuverlässiger bei Rauschen (siehe Abb. 4.1).

Für Messdaten siehe Tabellen 4.2 und 4.3.



(a) Blockbasierte Methode



(b) Spielsteinbasierte Methode

Abbildung 4.1.: Zuverlässigkeit der Erkennung in Prozent in Abhängigkeit von Rauschen auf dem Bild (0 bedeutet kein Rauschen, 1 bedeutet totales Rauschen).

Tabelle 4.2.: Blockbasierte Methode. Zuverlässigkeit der Erkennung in Prozent in Abhängigkeit von Fehlkalibrierung in Pixel und Rauschen.

Störung	Aktueller Spielstein (bekannt)	Aktueller Spielstein (nicht bekannt)	Nächster Spielstein
0	100.0%	100.0%	100.0%
1	100.0%	100.0%	100.0%
2	100.0%	100.0%	100.0%
4	100.0%	98.6%	98.2%
8	95.8%	63.8%	41.8%
0	100.0%	100.0%	100.0%
0.2	100.0%	100.0%	100.0%
0.4	100.0%	100.0%	99.8%
0.6	100.0%	99.8%	92.2%
0.8	97.6%	87.2%	48.0%

¹Da die Testbilder unter optimalen Verhältnissen aufgenommen wurden, lieferte das Messverfahren meist 100% Zuverlässigkeit. Die Bedingungen mussten also mit Störfaktoren künstlich verschlechtert werden, um die Ergebnisse überhaupt noch vergleichen zu können.

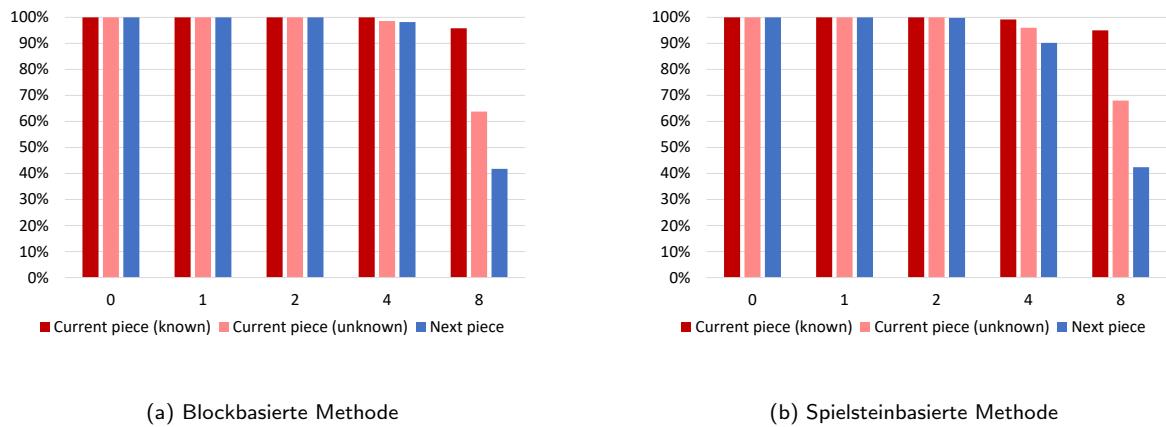


Abbildung 4.2.: Zuverlässigkeit der Erkennung in Prozent in Abhängigkeit von Fehlkalibrierung in Pixel.

Tabelle 4.3.: Spielsteinbasierte Methode. Zuverlässigkeit der Erkennung in Prozent in Abhängigkeit von Fehlkalibrierung in Pixel und Rauschen.

Störung	Aktueller Spielstein (bekannt)	Aktueller Spielstein (nicht bekannt)	Nächster Stein
0	100.0%	100.0%	100.0%
1	100.0%	100.0%	100.0%
2	100.0%	100.0%	99.8%
4	99.2%	96.0%	90.2%
8	95.0%	68.0%	42.4%
0	100.0%	100.0%	100.0%
0.2	100.0%	100.0%	100.0%
0.4	100.0%	100.0%	100.0%
0.6	99.8%	100.0%	99.0%
0.8	96.2%	94.2%	48.4%

4.3. Spielstärke der KI

4.3.1. Messverfahren

Die Stärke der KI gibt an, wieviele Linien abgebaut werden können bzw. welche Punktzahl erreicht wird, bevor das Spiel endet. Dafür werden jeweils 100 Spiele auf dem Emulator durchgeführt und die Anzahl kompletter Linien bzw. die Punktzahl gemittelt. Beim Simulieren wird auch berücksichtigt, dass der Spielstein weiter fällt, je mehr Aktionen ausgeführt werden (75 Millisekunden verstreichen pro Tastendruck, Rotation und Translation werden gleichzeitig ausgeführt). Gestartet wird jeweils in Level 9.

Das Messergebnis ist eine Kennzahl für die Qualität der Bewertungsfunktion bzw. der Strategie der KI.

Verwendetes Gerät für die Messungen: siehe Abschnitt 3.3.1.3.

4.3.2. Ergebnisse

Die Bewertungsfunktion von Yiyuan Lee schneidet besser ab. In jedem Szenario kann die KI sowohl mehr Zeilen abbauen, als auch eine höhere Punktzahl erreichen.

Berechnet wurden Mittelwert und Standardabweichung der kompletzierten Linien sowie der erreichten Punktzahl. Einmal unter der Berücksichtigung, dass der Spielstein mit konstanter Geschwindigkeit fällt (obere Werte in der Tabelle), und einmal ohne (untere Werte in der Tabelle), siehe Tabelle 4.4.

Tabelle 4.4.: Vergleich der Tetris-Bewertungsfunktionen anhand jeweils 100 simulierten Spielen.

	Yiyuan Lee		Max Bergmark	
	Linien	Punktzahl	Linien	Punktzahl
\bar{x}	6736.7	900241.7	3205.5	890606.0
σ	7540.2	224947.0	2615.9	236741.3
\bar{x}	21918.9	983052.7	6039.3	970509.1
σ	24084.9	100347.6	6192.9	116671.2

4.4. Zuverlässigkeit der Aktoren

4.4.1. Messverfahren

Während den ersten Tests mit dem Prototypen des Roboters fiel sofort auf, dass die Aktoren nicht alle Befehle ausführten, welche die KI ihnen übermittelte. Es war also ein Verfahren nötig, um Probleme bei den Aktoren aufzuzeigen und zu isolieren.

Ziel dieses Tests ist es, die Aktoren auf ihre Zuverlässigkeit zu prüfen. Der Test besteht aus einer statisch programmierten Sequenz.

Dieser Test gibt keinen Rückschluss auf die Langlebigkeit der Aktoren.

Aufbau

Der Test wird mittels der Aktoren des Roboters durchgeführt. Der Game Boy wird mit dem Spiel Tetris gestartet und der Singleplayer-Modus (A-Type) angewählt. Im Level-Auswahlmenü wird die Testroutine ausgeführt.

Verwendetes Gerät für den Test: siehe Abschnitt 3.3.1.4.

Testroutine

In der Testroutine führen die Aktoren folgende Sequenz aus: \leftarrow , \downarrow , \rightarrow , \uparrow , A, \rightarrow , \leftarrow , B. Nach der Durchführung dieser Sequenz wird die Testroutine für 2 Sekunden unterbrochen. Es gibt 20 Testdurchläufe.

Am Ende der Testroutine sollten die Tasten \downarrow , \uparrow , A und B 20 mal und die Tasten \leftarrow und \rightarrow 40 mal betätigt worden sein.

Auswertung

Nach dieser Sequenz muss der Cursor nach korrekter Ausführung wieder am Ausgangspunkt stehen. Steht der Cursor nicht wieder am Ausgangspunkt, wurde mindestens ein Tastendruck nicht richtig ausgeführt. Durch Sichtprüfung nach jeder Sequenz wird festgehalten, ob die Sequenz fehlerfrei abgearbeitet wurde oder nicht.

Die Messgrösse berechnet sich wie folgt:

$$\text{Zuverlässigkeit} = \frac{\text{erfolgreiche Testdurchläufe}}{\text{Testdurchläufe total}}$$

4.4.2. Ergebnisse

Die ersten Ergebnisse waren nicht erfolgreich (siehe Tabelle 4.5). Die Finger des Roboters rutschten teilweise von den Game-Boy-Tasten ab. Dieses Problem wurde mit den Fingerhülsen (siehe Abschnitt 3.2.4.3) behoben.

Auch die zweiten Ergebnisse waren nicht zufriedenstellend (siehe Tabelle 4.6). Der Finger für die Taste \leftarrow traf zu weit ins Zentrum des Steuerkreuzes und drückte somit \leftarrow und \downarrow gleichzeitig. Die Finger wurden mit einer neuen Führung exakter ausgerichtet.

Beim dritten Anlauf waren zehn Testserien erfolgreich (siehe Tabelle 4.7). Der Test wurde anschliessend nicht mehr durchgeführt. Bei Bedarf kann dank der Spieldaten (Logfiles) festgestellt werden, ob ein Problem auf die Aktoren zurückzuführen ist.



Tabelle 4.5.: Ergebnisse der Zuverlässigkeit der Aktoren (Serie 1).

Messung	Erfolgreich	Aktion ←, →	Aktion ↑, ↓, A, B
1	nein	40	20
2	nein	80	40
3	nein	120	60

Tabelle 4.6.: Ergebnisse der Zuverlässigkeit der Aktoren (Serie 2).

Messung	Erfolgreich	Aktion ←, →	Aktion ↑, ↓, A, B
1	ja	40	20
2	nein	80	40
3	ja	120	60
4	nein	160	80
5	nein	200	100

4.5. Performance der Aktoren

4.5.1. Messverfahren

Ziel dieses Tests ist es, die Geschwindigkeit der Aktoren zu ermitteln.

Aufbau

Der Test wird mittels der Aktoren des Roboters durchgeführt. Der Game Boy wird mit dem Spiel Tetris gestartet und der Singleplayer-Modus (A-Type) angewählt. Im Level-Auswahlmenü wird die Testroutine ausgeführt.

Verwendetes Gerät für den Test: siehe Abschnitt 3.3.1.4.

Testroutine

In der Testroutine führen die Aktoren folgende Sequenz aus: ←, ↓, →, ↑, A, →, ←, B. Die Sequenz wird 10 mal wiederholt.

Vorgehen

Die Werte der folgenden zwei Parameter werden nach jeder erfolgreichen Testroutine reduziert, bis die Routine nicht mehr erfolgreich durchgeführt werden kann:

Beschaltungszeit Zeit der elektrischen Beschaltung eines Hubmagneten in Millisekunden.

Rückstellzeit Zeit für die Rückstellung eines Hubmagneten mittels Federkraft in Millisekunden.

Auswertung

Wurden nicht alle Aktionen erfolgreich durchgeführt, war die Beschaltungszeit respektive die Rückstellzeit der Hubmagneten zu kurz. Somit werden die Parameter der letzten erfolgreichen Testroutine übernommen. Das Resultat wird in Aktionen pro Minute (APM) ausgewertet.

$$\text{Performance} = \frac{60000 \text{ ms}}{\text{Beschaltungszeit} + \text{Rückstellzeit}}$$

4.5.2. Ergebnisse

Zu beachten gilt, dass alle acht Aktoren die gleichen Parameter haben.

Die Aktoren erreichen 857 Aktionen pro Minute (APM) (siehe Tabelle 4.8). Bei der Auswertung der Testserien wurde auch festgestellt, dass der Aktor für die Taste ← das Problem war. Somit ist 857 Aktionen pro Minute (APM) das Ergebnis des langsamsten Aktors.

Tabelle 4.7.: Ergebnisse der Zuverlässigkeit der Aktoren (Serie 3).

Messung	Erfolgreich	Aktion \leftarrow, \rightarrow	Aktion $\uparrow, \downarrow, A, B$
1	ja	40	20
2	ja	80	40
3	ja	120	60
4	ja	160	80
5	ja	200	100
6	ja	240	120
7	ja	280	140
8	ja	320	160
9	ja	360	180
10	ja	400	200

Tabelle 4.8.: Ergebnisse der Performance der Aktoren.

Messung	Beschaltungszeit (ms)	Rückstellzeit (ms)	Erfolgreich	APM
1	140	140	ja	214
2	80	80	ja	375
3	40	40	ja	750
4	20	20	nein	1500
5	30	30	nein	1000
6	35	35	nein	857
7	37	37	ja	811
8	37	35	nein	833
9	35	37	ja	833
10	33	37	ja	857
11	32	37	nein	870
12	33	36	nein	870

4.6. Spielstärke des Roboters

4.6.1. Messverfahren

Diese Messgröße gibt an, wie viele Linien der Roboter abbauen kann bzw. welche Punktzahl er erreicht, bevor das Spiel endet. Dafür spielt der Roboter jeweils 10 Spiele auf dem echten Game Boy. Die Anzahl kompletter Linien bzw. die Punktzahl wird gemittelt. Gestartet wird einmal in Level 9 und einmal in Level 9 mit Heart-Mode.

Das Messergebnis ist eine Kennzahl für die Leistung des Roboters insgesamt.

Verwendetes Gerät für die Messungen: siehe Abschnitt 3.3.1.4.

4.6.2. Ergebnisse

Folgende Ergebnisse wurden vom Roboter am Ende dieser Arbeit erreicht (siehe Tabelle 4.9).

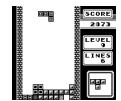
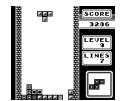


Tabelle 4.9.: Erreichte Punktezahlen des finalen Roboters.

<i>Spiel</i>	<i>Linien</i>	<i>Punkte</i>
1	625	508171
2	476	377923
3	1216	999999
4	694	605587
5	641	543543
6	909	771634
7	504	408965
8	411	333723
9	833	700815
10	680	583411
\bar{x}	699	583377
σ	238	201522



5. Diskussion

5.1. Mensch gegen Maschine

Als Referenz diente hier die Classic Tetris World Championship, bei der sich die besten Tetris-Spieler der Welt messen (siehe Abschnitt 2.2.7). Beispielsweise reichte Jonas Neubauer im Finale der Meisterschaft 2016 im dritten und letzten Spiel gegen Jeff Moore ein Punktestand von 764'030 um Tetris-Weltmeister zu werden. Ein Max-Out ist also sehr selten und wird meist in den Qualifikationsrunden erreicht.

Mit durchschnittlich 634'476 Punkten (siehe Abschnitt 4.6) ist der Roboter noch zu wenig kompetitiv, um an der Classic Tetris World Championship in den Final-Spielen mithalten zu können. Das Problem liegt unter anderem in der Spielstrategie. Master-Tetris-Spieler verwenden die Stacking-Strategie (siehe Abschnitt 2.2.6.1), um die Punktzahl zu maximieren. Zudem haben sie die Möglichkeit, Tetriminos seitlich in eine Lücke zu schieben, was der Roboter nicht kann.

Trotz dieser Schwächen erreichte der Roboter am 5. Januar 2017 eine Max-Out-Score (siehe Abb. 5.1). Mit 1'216 abgebauten Linien ein beeindruckendes Resultat. Die 999'999 Punkte wurden bereits mit 1'133 Linien erreicht. Mit der Stacking-Strategie wäre dieser Punktestand mit deutlich weniger Linien erreicht worden.

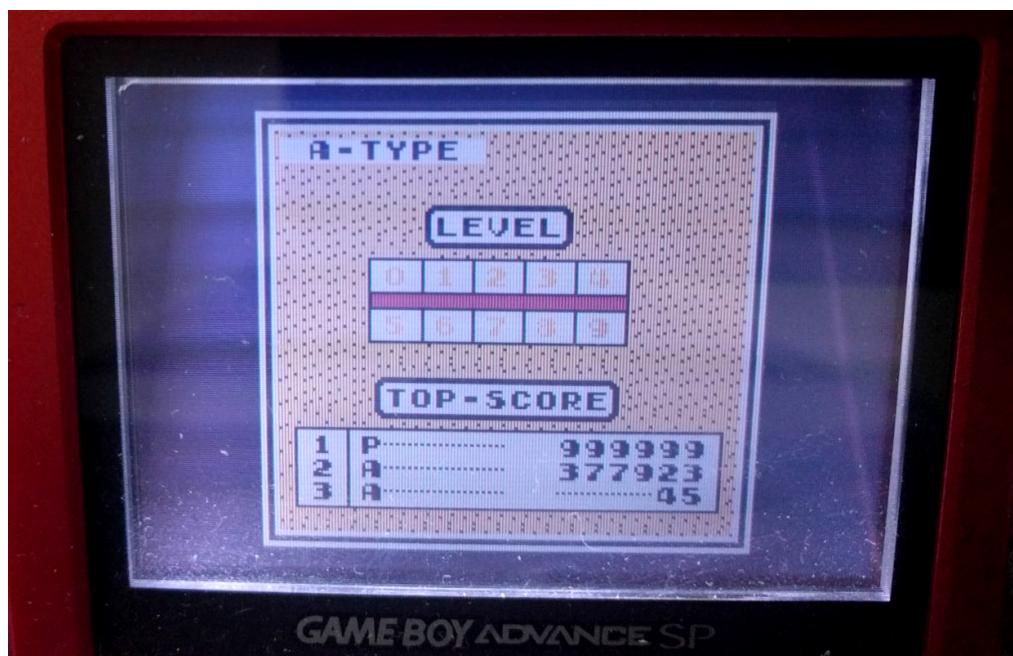


Abbildung 5.1.: Screenshot der Highscore-Übersicht mit der erreichten Max-Out-Score.

Um einen fairen Vergleich zwischen Mensch und Maschine machen zu können, müsste der Roboter allerdings auf dem NES spielen und an den Turnieren der Classic Tetris World Championship teilnehmen.

Die Stärken des Roboters sind seine Ausdauer, das zielsichere Vorgehen und das extrem schnelle Ausführen von Aktionen. Die Stärken des Menschen sind seine Kreativität und die enorme Lernfähigkeit des Gehirns. Profi-Tetris-Spieler üben dennoch Jahre, um dann in wenigen Millisekunden entscheiden zu können, wie ein Spielstein platziert werden soll.

5.2. Limiten

5.2.1. Kamera

Die Framerate der Kamera limitiert die Reaktionszeit des Bots (siehe Abschnitt 3.2.6.2). Ist die Framerate zu tief, verpasst der Bot wertvolle Bilder oder kann auf diese erst verzögert reagieren. Die Framerate ist einerseits gegeben durch Beschränkungen der Kamera (Auslesegeschwindigkeit des Sensors, Buffering) und des Übertragungsmediums (USB 2). Andererseits begrenzt die CPU die Geschwindigkeit bei der Decodierung der Frames. Je nach eingesetzten Treibern (ffmpeg300) und Software-Libraries (OpenCV) wird die Framerate zusätzlich limitiert.

5.2.2. Bildverarbeitung und -analyse

Die Bildverarbeitung eines Frames dauert nicht einmal eine Millisekunde, wenn ausschliesslich lokale Operatoren verwendet werden. Diese Geschwindigkeit ist möglich, da nur mit Bildern von sehr kleiner Auflösung gearbeitet wird (640×480 und 160×144 Pixel). Würde man die Bildverarbeitung auf einer Grafikkarte ausführen, könnte man zusätzliche Verbesserungen in der Geschwindigkeit erzielen. Der Engpass liegt aber eindeutig nicht bei der Bildverarbeitung.

Bei der Bildanalyse muss ein Kompromiss zwischen Geschwindigkeit und Zuverlässigkeit gemacht werden. Momentan werden mindestens zwei Bilder gesampelt, um die nötige Zuverlässigkeit erbringen zu können. Dies geht auf Kosten der Geschwindigkeit. Das Problem liegt bei den Überblendungen zweier Frames auf dem Game-Boy-Display. Wird das Bild im falschen Moment aufgenommen, sind zwei Frames gleichzeitig sichtbar. Bei der Bildanalyse kann nicht mehr exakt unterschieden werden, von welchem Frame der Spielstein erkannt wurde. Um diese Übergänge zu erkennen, müssten komplexere Methoden angewandt werden. Das Sampling ist einfacher, hat aber den Nachteil, dass mehrere Frames analysiert werden müssen und so wertvolle Zeit verloren geht.

5.2.3. Künstliche Intelligenz

Die Herausforderung der KI besteht im Meistern eines *NP*-vollständigen Spiels in Echtzeit. Dies ist eine sehr schwierige Aufgabe. Nicht zuletzt wegen der hohen Komplexität von Tetris. Der Verzweigungsfaktor liegt bei ungefähr 23, wenn die n nächsten Spielsteine bekannt sind. Sonst ist der Verzweigungsfaktor sogar 162^1 .

Aufgrund des exponentiellen Anstiegs der Rechenzeit mit jeder Erhöhung der Suchtiefe ist die maximale Suchtiefe sehr begrenzt. Die Limite bildet die Prozessorleistung. Durch Parallelisieren der Breitensuche oder einem Pruning des Verzweigungsfaktors könnte man die Rechenzeit reduzieren.

Ausserdem ist die verwendete Bewertungsfunktion nicht perfekt, sondern nur eine Annäherung. Nur vier Merkmale werden für die Bewertung nach Yiyuan Lee betrachtet. Würde man weitere Merkmale berücksichtigen, könnten die Resultate wahrscheinlich noch verbessert werden. Die Gewichtung der einzelnen Merkmale liesse sich mittels genetischen Algorithmen oder neuronalen Netzen weiter optimieren. Zusätzlich könnten Spielstandbewertungen vorberechnet und in einer Datenbank gespeichert werden. So könnte man die Performance weiter steigern.

5.2.4. Plattform

Die Tests mit den unterschiedlichen Plattformen haben deutlich gezeigt, dass eine grosse Abhängigkeit zwischen der Hardware-Leistung und der Spielstärke des Roboters besteht. Dies ist darauf zurückzuführen, dass viele Abläufe extrem CPU-lastig sind: Decodieren der Kamerabilder, Bildverarbeitung und -analyse, Breitensuche. Der limitierende Faktor ist hier offensichtlich die Prozessorleistung.

Es ist anzunehmen, dass mit einer noch schnelleren Plattform durchschnittlich nochmals bessere Resultate erzielt werden könnten.

¹Zum Vergleich: Schach hat einen Verzweigungsfaktor von ca. 35[14].



5.2.5. Aktoren

Bei den Hubmagneten ist der limitierende Faktor die Magnetisierung der Spulen. Diese dauert 16 ms und benötigt eine Leistung von 144 Watt (siehe Datenblatt D.1.1). Zu Beginn macht der Hubmagnet keine Bewegung. Erst nach 16 ms ist das Magnetfeld genug stark um den Eisenkern anzuziehen.

Durch den Einsatz kleinerer Spulen mit schnellerem Ansprechverhalten wäre es möglich, die Anzugszeit noch weiter zu verkürzen. Dadurch würden die Hubmagnete allerdings noch schneller überhitzen. Dieses Problem wurde beim ersten Prototypen bereits nach wenigen Tastendrücken festgestellt. Auch die grossen Kuhnke Hubmagnete neigen zur Überhitzung. Diesem Problem konnte allerdings mit Lüftern entgegengewirkt werden. Mit den Hubmagneten wurde also eine Kompromisslösung zwischen Hitzeentwicklung und Geschwindigkeit gefunden.

Wahrscheinlich könnte die Performance mit den gleichen Hubmagneten noch ein wenig gesteigert werden, indem man jeden Aktor individuell statt gemeinsam parametrisiert. Grund für diese Annahme ist, dass die in der Mitte platzierten Bowdenzüge eine geringere Reibung als die an den Ecken platzierten Bowdenzüge haben. Somit müssten die Hubmagnete in der Mitte schneller ansprechen, sprich müssten weniger lang als 32 ms beschalten werden.



6. Schlussfolgerungen

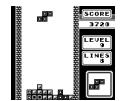
Den Autoren ist es gelungen, einen Roboter zu entwickeln, der Tetris auf dem Game Boy spielen und sogar im höchsten Level eine beträchtliche Punktzahl erreichen kann. Auf dem Weg dorthin sind sie an viele Grenzen gestossen.

Bei der Informationserfassung sind Reflexionen und schlechter Kontrast äusserst störend. Ein Roboter hat es also leichter, auf einem Game Boy mit Hintergrundbeleuchtung zu spielen. Die Reaktionszeit des Roboters hängt vor allem von der Framerate der Kamera ab. Die Verarbeitung und Analyse der Bilder kann in wenigen Millisekunden ausgeführt werden.

Obwohl Tetris regeltechnisch ein sehr einfaches Spiel ist, ist das Finden des optimalen Spielzuges eine äusserst komplexe Aufgabe. Mittels raffinierter Heuristiken ist es dennoch möglich, gute Spielzüge in geringer Zeit zu berechnen.

Die Ausführung der Tastendrücke ist vor allem durch physikalische Gesetze beschränkt. Aktionen in kinetische Energie umzuwandeln bedeutet Energieverbrauch und Zeit. Mit starken Hubmagneten ist man den allermeisten menschlichen Fingern überlegen.

Trotz allen Hürden bei der Optimierung des Roboters, die letzte Grenze bildet das Spiel selbst. Auch wenn man Tetris perfekt und unendlich schnell spielt, gibt es Sequenzen von Spielsteinen, die einen zwingen, das Spiel zu verlieren[1]. Dieser Umstand liefert die absolute Obergrenze, wie gut ein Tetris-Spieler überhaupt sein kann – egal ob Mensch oder Maschine.



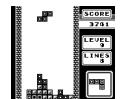
Selbständigkeitserklärung

Wir bestätigen, dass wir die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der im Literaturverzeichnis angegebenen Quellen und Hilfsmittel angefertigt haben. Sämtliche Textstellen, die nicht von uns stammen, sind als Zitate gekennzeichnet und mit Quellenangabe versehen.

Ort, Datum: Bern, 19.01.2017

Namen Vornamen: Patrik Marti Mathias Winkler

Unterschriften:



Glossar

NP-vollständig Begriff aus der Komplexitätstheorie für die schwierigsten Probleme in der Klasse *NP*.

Agent Autonomes Computerprogramm, das seine Umwelt über Sensoren wahrnehmen und mittels Aktoren darauf einwirken kann.

Aktionen pro Minute (APM) Ausführungsgeschwindigkeit von Aktionen in einem Videospiel.

API Schnittstelle zur Anwendungsprogrammierung.

Arduino Physical-Computing-Plattform mit quellöffentner Soft- und Hardware.

Block out Beschreibt beim Spiel Tetris den Zustand, wenn ein Spielstein aufgrund des überfüllten Spielfeldes nicht mehr erzeugt werden kann.

Classic Tetris World Championship Offizielle Tetris-Weltmeisterschaft.

Crossing-over Ein genetischer Operator, der die Gene der Eltern neu kombiniert, um damit die Gene der Nachfolger zu erzeugen.

EmguCV .NET-Wrapper für OpenCV, geschrieben in C#.

Endlicher Automat Ein Modell für Objekte mit einer endlichen Anzahl definierter Zustände.

FFmpeg Sammlung von Programmbibliotheken zur Aufnahme, Codierung und Decodierung von Audio- und Videomaterial.

Game Link Cable Kabel, womit sich zwei Game-Boy-Geräte miteinander zu verbinden lassen.

Genetischer Algorithmus Von der Evolution inspirierter Algorithmus zum Lösen von Optimierungsproblemen.

Harddrop Das aktive und sofortige Fallenlassen und Fixieren eines Tetriminos im Spiel Tetris.

Internet-Protokoll Verbreitetes Netzwerkprotokoll.

KI Künstliche Intelligenz.

Library Programmbibliothek.

Lookup-Tabelle Vorberechnete statische Werte, welche mit einem Schlüssel schnell nachgeschlagen werden können.

Max-Out-Score Ein Begriff für die maximale Punktzahl (999'999), die im Spiel Tetris erreicht werden kann.

NES Alte Videospielkonsole von Nintendo.

NuGet Paketverwaltung für die Softwareentwicklung unter .NET.

OpenCV Eine Programmbibliothek mit Algorithmen unter anderem für die Bildverarbeitung.

Raspberry Pi Einplatinencomputer, entwickelt von der Raspberry Pi Foundation.

Softdrop Das aktive Fallenlassen eines Tetriminos im Spiel Tetris.

Template Matching Eine Technik in der digitalen Bildverarbeitung um Teile eines Bildes zu finden, die einem Template entsprechen.

Tetrimino Spielstein, wie er in Tetris genannt wird. Benannt nach der geometrischen Figur *Tetromino*.

Tetris Guideline Die Spezifikation für die Normierung von Tetris-Spielen.

The Tetris Company Die Firma, der die Lizenzrechte von Tetris gehört.

Tinkerforge Plattform verschiedener Mikrocontrollerbausteine, die unterschiedliche Module ansteuern können.

Tournament Selection Eine Methode, in einem genetischen Algorithmus Individuen aus der Population auszuwählen.

Vorzeitige Konvergenz Frühzeitiges Konvergieren einer Population gegen ein nur lokales Optimum in einem genetischen Algorithmus.



Literaturverzeichnis

- [1] "A deadly piece sequence," http://harddrop.com/wiki/A_deadly_piece_sequence, besucht: 27.12.2016.
- [2] "Classic Tetris World Championship – History," <http://thectwc.com/history>, besucht: 03.01.2017.
- [3] "Input lag measurements," <http://boards.dingonity.org/gpd-android-devices/input-lag-measurements/>, besucht: 03.01.2017.
- [4] "Most ported videogame,"
<http://www.guinnessworldrecords.com/world-records/most-ported-computer-game/>, besucht: 28.12.2016.
- [5] "Random Generator," http://tetris.wikia.com/wiki/Random_Generator, besucht: 28.12.2016.
- [6] "Tetris Concept," <http://www.tetrisconcept.com/>, besucht: 01.01.2017.
- [7] "Tetris Guideline," http://tetris.wikia.com/wiki/Tetris_Guideline, besucht: 28.12.2016.
- [8] "Tetris Harddrop," [http://harddrop.com/wiki/Tetris_\(Game_Boy\)](http://harddrop.com/wiki/Tetris_(Game_Boy)), besucht: 28.12.2016.
- [9] "Tetris (NES, Nintendo)," [https://tetris.wiki/Tetris_\(NES,_Nintendo\)#Differences_in_the_PAL_release](https://tetris.wiki/Tetris_(NES,_Nintendo)#Differences_in_the_PAL_release), besucht: 28.12.2016.
- [10] "TGM 3 Tetris Arika, Invisible Tetris," <https://www.youtube.com/watch?v=jwC544Z37qo>, besucht: 28.12.2016.
- [11] "Wikipedia – Aktor," <https://de.wikipedia.org/wiki/Aktor>, besucht: 06.01.2017.
- [12] "Wikipedia – Elektromagnet," <https://de.wikipedia.org/wiki/Elektromagnet>, besucht: 06.01.2017.
- [13] "Wikipedia – Game Boy," https://de.wikipedia.org/wiki/Game_Boy,
https://en.wikipedia.org/wiki/Game_Boy, besucht: 03.01.2017.
- [14] "Wikipedia – Game complexity,"
https://en.wikipedia.org/wiki/Game_complexity#Complexities_of_some_well-known_games, besucht: 28.12.2016.
- [15] "Wikipedia – List of best-selling Game Boy video games,"
https://en.wikipedia.org/wiki/List_of_best-selling_Game_Boy_video_games, besucht: 28.12.2016.
- [16] "Wikipedia – List of Tetris variants," https://en.wikipedia.org/wiki/List_of_Tetris_variants, besucht: 28.12.2016.
- [17] "Wikipedia – Schrittmotor," <https://de.wikipedia.org/wiki/Schrittmotor>, besucht: 06.01.2017.
- [18] "Wikipedia – Servo," <https://de.wikipedia.org/wiki/Servo>, besucht: 06.01.2017.
- [19] "Wikipedia – Tetris," <https://de.wikipedia.org/wiki/Tetris>, besucht: 28.12.2016.
- [20] "Wikipedia – Tetris (Game Boy)," [https://en.wikipedia.org/wiki/Tetris_\(Game_Boy\)](https://en.wikipedia.org/wiki/Tetris_(Game_Boy)), besucht: 28.12.2016.
- [21] M. Bergmark, "Tetris: A Heuristic Study," Bachelorthesis, KTH, School of Engineering Sciences (SCI), 5 2015, <http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-168306>.
- [22] M. Birken, "Game Boy Emulator," <http://meatfighter.com/gameboy>, besucht: 28.12.2016.
- [23] D. L.-N. Erik D. Demaine, Susan Hohenberger, "Computing and Combinatorics: 9th Annual International Conference, COCOON 2003 Big Sky, MT, USA, July 25–28, 2003 Proceedings," B. Z. Tandy Warnow, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, ch. Tetris is Hard, Even to Approximate, pp. 351–363, http://dx.doi.org/10.1007/3-540-45071-8_36.
- [24] C. Fahey, "Tetris – Colin Fahey," <http://colinfahey.com/tetris/tetris.html>, besucht: 22.12.2016.

- [25] R. Heise, "How to play TETRIS – TETRIS stacking tutorial,"
http://www.ryanheise.com/tetris/tetris_stacking.html, besucht: 03.01.2017.
- [26] M. Holland, „Die Story ist so gross“: Tetris-Film wird zur Trilogie,” [https://www.heise.de/newsticker/meldung/Die-Story-ist-so-gross-Tetris-Film-wird-zur-Trilogie-3251022.html](http://www.heise.de/newsticker/meldung/Die-Story-ist-so-gross-Tetris-Film-wird-zur-Trilogie-3251022.html), 6 2016, besucht: 22.12.2016.
- [27] C. Kazis, “Tetris-Effekt: Wenn unser Denken dominiert wird,”
<http://www.srf.ch/kultur/wissen/tetris-effekt-wenn-unser-denken-dominiert-wird>, 11 2016, besucht: 22.12.2016.
- [28] Y. Lee, “Tetris AI – The (Near) Perfect Bot,”
<https://codemyroad.wordpress.com/2013/04/14/tetris-ai-the-near-perfect-player/>, 4 2013, besucht: 28.12.2016.
- [29] J. Neubauer, “Youtube – NES Tetris - 999,999,” <https://www.youtube.com/watch?v=qYDPMHkHh8s>, besucht: 05.01.2017.
- [30] F. Shump, “Lego Robot Plays Tetris While Planning to Destroy Mankind,”
<http://walyou.com/lego-robot-plays-tetris/>, 4 2010, besucht: 22.12.2016.
- [31] C. Thiery and B. Scherrer, “Building Controllers for Tetris,” *International Computer Games Association Journal*, vol. 32, pp. 3–11, 2009. [Online]. Available: <https://hal.inria.fr/inria-00418954>



Abbildungsverzeichnis

2.1. Game Boy Advance SP	5
2.2. Tetris Multiplayer-Modus mit Linien des Gegners am unteren Ende des Spielfeldes.	7
2.3. Beispiel eines Spielfeldes bei der Spielstrategie <i>Stacking</i>	8
2.4. Jeff Moore und Jonas Neubauer beim Finale der Classic Tetris World Championship 2016.	8
3.1. Versuchsaufbau des finalen Roboters ohne Computer.	11
3.2. Skizze und Foto des ersten Prototypen des Roboters.	12
3.3. Skizze des Konzepts mit Gelenken.	13
3.4. Querschnitt des verwendeten Bowdenzuges. Querschnitt (links) und 3D-Modell (rechts).	13
3.5. Screenshot aus Solid Edge (links) und das Bauteil, erstellt vom 3D-Drucker (rechts).	14
3.6. Roboter als 3D-Modell.	15
3.7. Elektronischer Schaltplan des Roboters.	16
3.8. Einer von acht Aktoren des Roboters.	17
3.9. Stark vereinfachte Darstellung der Engine und ihrer Komponenten.	21
3.10. Datenfluss der Engine bei Input und Output.	22
3.11. Das Verfahren der Bildentzerrung.	23
3.12. Vergleich der beiden Thresholding-Verfahren.	23
3.13. Morphologisches Opening zum Ausfüllen der Blöcke im Spielfeld.	24
3.14. Wahrscheinlichkeiten der Tetriminos in Abhängigkeit der zwei vorhergehenden Tetriminos. Die obere Zeile der x-Achsenbeschriftung gibt das letzte Tetrimino an, die untere Zeile das vorletzte Tetrimino.	26
3.15. Berechnung der Drop-Distanz anhand der unteren Kontur des Spielsteins (schwarze Blöcke).	27
3.16. Die Templates inklusive Masken sämtlicher Spielsteine und ihrer zulässigen Rotationen.	28
3.17. Verifizierung des Spielfeldes des Kamerabildes (links) anhand des internen Spielstandes (rechts).	29
3.18. Vereinfachte Darstellung der Breitensuche mit Tiefe 2 und den Spielsteinen I und T.	30
3.19. Unterschiedliche Möglichkeiten, die Spielsteine S und I zu platzieren. Inklusive Bewertung des resultierenden Spielstandes.	31
3.20. Zustandsdiagramm des Agenten.	33
4.1. Zuverlässigkeit der Erkennung in Prozent in Abhängigkeit von Rauschen auf dem Bild (0 bedeutet kein Rauschen, 1 bedeutet totales Rauschen).	36
4.2. Zuverlässigkeit der Erkennung in Prozent in Abhängigkeit von Fehlkalibrierung in Pixel.	37
5.1. Screenshot der Highscore-Übersicht mit der erreichten Max-Out-Score.	43
A.1. Bildschirmausschnitt aus dem Issue-Tracking-System von Bitbucket.	59
A.2. Zeiplan der Arbeit.	60
C.1. Zusammenbauzeichnung aller Bauteile.	66
C.2. Plan Aktor Montageplatte.	67
C.3. Plan Bowdenzug-Führung oben.	68
C.4. Plan Bowdenzug-Führung unten.	69
C.5. Plan Game-Boy-Matrise.	70
C.6. Plan Grundplatte.	71
C.7. Plan Kamerahalterung.	72
C.8. Plan Lüfterhalterung.	73
C.9. Plan Mast.	74
C.10. Plan Montagewinkel.	75
C.11. Plan Tastenlayout Träger.	76
C.12. Plan Tastenlayout.	77

D.1. Datenblatt des Kuhnke H3268-R-F24V Hubmagnet.	79
E.1. Klassendiagramm des Assemblies <i>GameBot.Core</i>	81
E.2. Klassendiagramm des Assemblies <i>GameBot.Game.Tetris</i>	81



Tabellenverzeichnis

3.1. Fallgeschwindigkeiten der Spielsteine pro Level.	25
3.2. Wahrscheinlichkeiten der Tetriminos.	26
3.3. Auswirkungen auf die Fehlerrate e in Abhangigkeit der Sampling-Grosse n	29
3.4. Alle moglichen Tetrimino-Posen auf einem Spielfeld mit Breite 10.	30
3.5. Zeitkomplexitat der Breitensuche in Abhangigkeit der Suchtiefe n	32
3.6. Resultate der Optimierung der Gewichtsfaktoren mittels genetischem Algorithmus.	32
4.1. Messergebnisse fur die Bildverarbeitungszeit in Millisekunden (pro Operator und total).	35
4.2. Blockbasierte Methode. Zuverlassigkeit der Erkennung in Prozent in Abhangigkeit von Fehlkalibrierung in Pixel und Rauschen.	36
4.3. Spielsteinbasierte Methode. Zuverlassigkeit der Erkennung in Prozent in Abhangigkeit von Fehlkalibrierung in Pixel und Rauschen.	37
4.4. Vergleich der Tetris-Bewertungsfunktionen anhand jeweils 100 simulierten Spielen.	38
4.5. Ergebnisse der Zuverlassigkeit der Aktoren (Serie 1).	39
4.6. Ergebnisse der Zuverlassigkeit der Aktoren (Serie 2).	39
4.7. Ergebnisse der Zuverlassigkeit der Aktoren (Serie 3).	40
4.8. Ergebnisse der Performance der Aktoren.	40
4.9. Erreichte Punktezahlen des finalen Roboters.	41
A.1. Budget und Kosten.	62



A. Projektmanagement

A.1. Projektmethode

Die Autoren sind bei ihrer Arbeit iterativ vorgegangen. Komponente für Komponente wurde geplant, entwickelt und getestet. Bei Anpassungen und Optimierungen wurde dieses Vorgehen ebenfalls angewandt. So konnte agil auf Anpassungen und neue Anforderungen reagiert werden.

A.1.1. Issue-Tracking

Als Issue-Tracking-System wurde Bitbucket eingesetzt. So konnte das Team Anforderungen und Tasks einfach verwalten (siehe Abb. A.1).

The screenshot shows the Bitbucket Issues page for the repository 'realgameboys / GameBot / GameBot'. On the left is a sidebar with various icons for C#, Teams, Projects, Repositories, Snippets, and a search bar. The main area displays a list of 11 issues, each with a title, priority (T), progress (P), status, votes, assignee, created date, and updated date. The issues are:

Title	T	P	Status	Votes	Assignee	Created	Updated
#1: Unit-Tests für die Bildverarbeitung erstellen	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	RESOLVED	winki NA	2016-10-15	2016-10-17	
#2: Erkennung der Tetrominos anhand Pattern Matching	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	RESOLVED	winki NA	2016-10-15	2016-10-17	
#3: Kamera besorgen	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	RESOLVED	Patrik Marti	2016-10-15	2016-11-22	
#4: Kamerahalterung erstellen	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	RESOLVED	Patrik Marti	2016-10-15	2016-11-22	
#5: Testbilder machen	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	RESOLVED	Patrik Marti	2016-10-15	2016-11-21	
#6: Gummiblöppen auf den Bowdenzug	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	RESOLVED	Patrik Marti	2016-10-15	2016-11-22	
#7: Positionierung der Finger beim Steuerkreuz nach aussen verlegen	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	RESOLVED	Patrik Marti	2016-10-15	2016-11-22	
#8: Probabilistisches Finden eines Tetrominos auf dem Spielfeld	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	RESOLVED	winki NA	2016-10-15	2016-10-15	
#9: Sicherungsschicht bei den Aktuatoren	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	RESOLVED	winki NA	2016-10-15	2016-10-15	
#10: Zufallsverteilung Tetrominos genau analysieren	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	RESOLVED	winki NA	2016-10-15	2016-12-10	
#11: Netzwerk-Kamera evaluieren	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	CLOSED	Patrik Marti	2016-10-15	2016-11-22	

Abbildung A.1.: Bildschirmausschnitt aus dem Issue-Tracking-System von Bitbucket.

A.2. Projektverlauf

Die Projektarbeit richtete sich nach folgendem Zeitplan (siehe Abb. A.2).

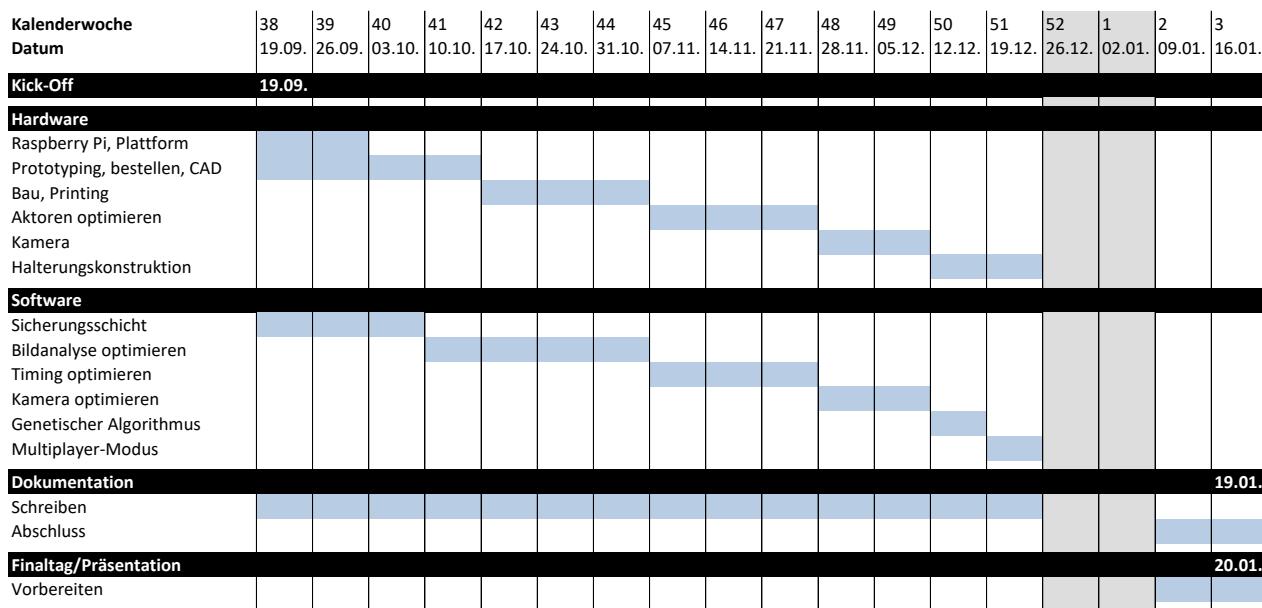


Abbildung A.2.: Zeiplan der Arbeit.

A.3. Projektorganisation

A.3.1. Beteiligte Personen

Studierende

Patrik Marti, patrik.marti@students.bfh.ch

Mathias Winkler, mathias.winkler@students.bfh.ch

Betreuer

Prof. Marcus Hudritsch, marcus.hudritsch@bfh.ch

Experte

Dr. Harald Studer, harald.studer@optimo-medical.com

A.3.2. Projektmeetings

Die Projektmeetings der Studenten mit dem Betreuer fanden einmal wöchentlich per Skype statt. Zusätzlich gab es einige persönliche Treffen in Biel.

Das Treffen mit dem Experten fand am 4. November 2016 in Biel statt.

A.3.3. Arbeitsweise

Die Studenten koordinierten sich untereinander jeweils wöchentlich über den Stand der Arbeiten und das weitere Vorgehen. In einer ersten Phase arbeiteten die Studenten jeweils selbstständig an ihren Tasks. Im späteren Verlauf der Arbeit trafen sich die Studenten oft, um gemeinsam den Roboter zu testen und die einzelnen Komponenten zu optimieren.

Konkrete Anforderungen lagen zu Beginn der Arbeit nicht vor. Die Studenten analysierten eigenständig, welche Tasks für das weitere Vorgehen relevant sind. Nach jeweils kurzer Absprache mit dem Betreuer wurden diese Tasks dann in Angriff genommen.



A.4. Hilfsmittel

- Visual Studio 2015 als Entwicklungsumgebung, Programmiersprache C#
- Python 3 für Auswertungsscripts
- git zur Versionierung
- Bitbucket¹ als git-Repository und Issue-Tracking-System
- NuGet² zur Paketverwaltung
- Tinkerforge Deamon brickd zum Ansteuern der Tinkerforge-Bricks
- L^AT_EX zur Erstellung des Berichts (MiK^TeX 2.9, Texmaker 4.5)
- NClass zur Erstellung von Klassendiagrammen³
- C# Game-Boy-Emulator⁴
- Siemens Solid Edge ST9 als Konstruktionssoftware
- Umtimaker 2+ 3D-Drucker
- Microsoft Visio 2010
- Microsoft Office (Excel)
- Fehlmann Säulenbohrmaschine mit Kreuztischen
- Diverses Werkzeug

A.5. Artefakte

Folgende Artefakte werden zusammen mit dieser Dokumentation in elektronischer Form abgegeben:

- Quellcode inklusive History
- Code-Dokumentation
- Messdaten
- Solid Edge Baugruppe des Roboters

A.6. Finanzen

Siehe Tabelle A.1.

¹<https://bitbucket.org>

²<https://www.nuget.org>

³<http://nclass.sourceforge.net/>

⁴<http://meatfighter.com/gameboy/>

Tabelle A.1.: Budget und Kosten.

<i>Bezeichnung</i>	<i>Stückpreis (CHF)</i>	<i>Stück</i>	<i>Total (CHF)</i>
Raspberry Pi Noir, Infrared Camera	40.90	1	40.90
Samsung EVO microSDHC mit Adapter (32GB, Class 10)	18.10	1	18.10
Raspberry Pi 3 Starter Kit (Onboard, Speziell)	75.10	1	75.10
Diverses Kleinmaterial	210.65	1	210.65
Microsoft Webcam LifeCam Studio Business	69.00	1	69.00
Kuhnke H3268-R-F24V Hubmagnet 5mm	86.20	8	689.60
Game Boy Advance Bundle - 2 Konsolen GBA SP	124.90	1	124.90
Filament Innofil 3D PLA-0023B075 PLA 2.85 mm Grau 750 g	34.95	2	69.90
PVC schwarz 500 x 1000 x 15 (keine Quittung)	50.00	1	50.00
LL-Bowdenzug 1500x4/3mm inkl. Anschluss M3	5.70	8	45.60
Enermax T.B. Silence (80mm)	11.00	2	22.00
<i>Total</i>			1'415.75
<i>Budget</i>			1'200.00
<i>Differenz</i>			-215.75



B. Korrespondenz

B.1. Jonas Neubauer

Facebook-Nachricht vom 22.11.2016 an Jonas Neubauer

Guten Abend Herr Neubauer

Mein Name ist Patrik Marti und zusammen mit meinem Studienkollegen Mathias Winkler studieren wir an der Berner Fachhochschule (BFH) Informatik. Für unsere Bachelor Thesis entwickeln wir einen Roboter, welcher Tetris auf einem Nintendo GameBoy SP spielen kann. Da wir beide nicht die talentiertesten Tetris Spieler sind, haben wir bereits jetzt keine Chance gegen unseren GameBot. Jetzt suchen wir einen wirklich kompletiven Gegner, welcher es gegen unsere Projektarbeit aufnimmt. Aus unserer Sicht gibt es wohl keinen besseren Gegner als der amtierende CTWC Weltmeister, also Sie.

Es würde mich ausserordentlich freuen, wenn wir Sie für diese Challenge begeistern könnten. Falls Sie Interesse haben, erreichen sie mich via Facebook oder Mail: patrik.marti@students.bfh.ch

Auf YouTube finden sie ein mittlerweile nicht mehr ganz aktuelles Video des GameBot
https://www.youtube.com/results?search_query=Patrik+MArti+gamebot

B.2. Tetris European Championship

Facebook-Nachricht an die Tetris European Championship vom 05.12.2016

Hello my name is Patrik Marti and together with my colleague Mathias Winkler we study at the Bern University of Applied Sciences.
We built and programmed a robot, that is able to play Tetris on a Game Boy SP. Now we are looking for a strong opponent who challenges our Robot.
For your support we would be very grateful. If you are interested please contact me one Facebook or eMail: patrik.marti@students.bfh.ch

B.3. Yiyuan Lee

E-Mail an Yiyuan Lee vom 24.11.2016

Hi Yiyuan

I'm Mathias and together with my friend Patrik we are currently working on our bachelor thesis. We are working on a physical bot that can play Tetris on a real Game Boy. During my research I came upon your blog article:
<https://codemyroad.wordpress.com/2013/04/14/tetris-ai-the-near-perfect-player/>.
Big compliment for your awesome work!
The current version of our bot is using your scoring function and I haven't found something better till now. Although Game Boy Tetris uses a slightly different rule set as the standard Tetris you described.

I just wanted to thank you for your great article and I'm also interested if you have improved your scoring function since then?
Also, what software or framework did you use for the genetic algorithm part?

On YouTube you will find a video of our bot in action:

<https://youtu.be/7zsxP6M78LA?t=41>
We are still improving it! :)

Best regards
Mathias

Antwort von Yiyuan Lee vom 25.11.2016

Hi Mathias,

I haven't made any changes to the scoring function apart from normalizing the weights some time ago, but it shouldn't affect performance at all.

The genetic tuning was done in Javascript and executed in Google Chrome, although I did not upload the source file for that and haven't been able to recover it since it was quite a few years ago.

I should be adding them in within 3 to 4 months time for completeness, after I am done serving military conscription.

Nonetheless, it should be quite straightforward to use the existing source files to write a genetic tuner - the AI is parameterizable with weights that are involved in the process.

All the best for your thesis! Do drop me an email if you need any help and I'll be glad to reply :)

Best Regards,
Yiyuan



C. Konstruktionspläne

	1	2	3	4		
A						
B						
C						
D	Pos.	Menge	Benennung	Zeichnungsnummer	Material	
1	1	Grundplatte	00011	PVC		
2	2	Montage Winkel	00005	Aluminium		
3	2	Mast	00006	PVC		
4	1	Bowdenzug Führung 2	00010	PVC		
5	1	Bowdenzug Führung 1	00001	PVC		
6	1	Game Boy Matrize	00009	PLA		
7	1	Tasten Layout 2.2	00002	PLA		
8	1	Tasten Layout 2.1 Halter	00003	PLA		
9	4	M6 Gewindestange		Stahl		
10	1	Montage Platte	00004	Stahl		
11	1	Lüfter Halterung	00007	PLA		
12	1	Kamera Halterung	00008	PLA		
E	Allgemeintoleranz		Maßstab 1:6.67	Werkstoff	Gewicht 4.163 kg	
F	Erstellt durch Martip	Genehmigt von Martip	Sachnummer 10001			
	Title, Zusätzlicher Titel Game Bot	Dokumentenart Zusammenbauzeichnung	And. 02	Ausgabedatum 2017-01-17	Spr. de	Blatt 1/1
SOLID EDGE SOLID EDGE ACADEMIC COPY						

Abbildung C.1.: Zusammenbauzeichnung aller Bauteile.

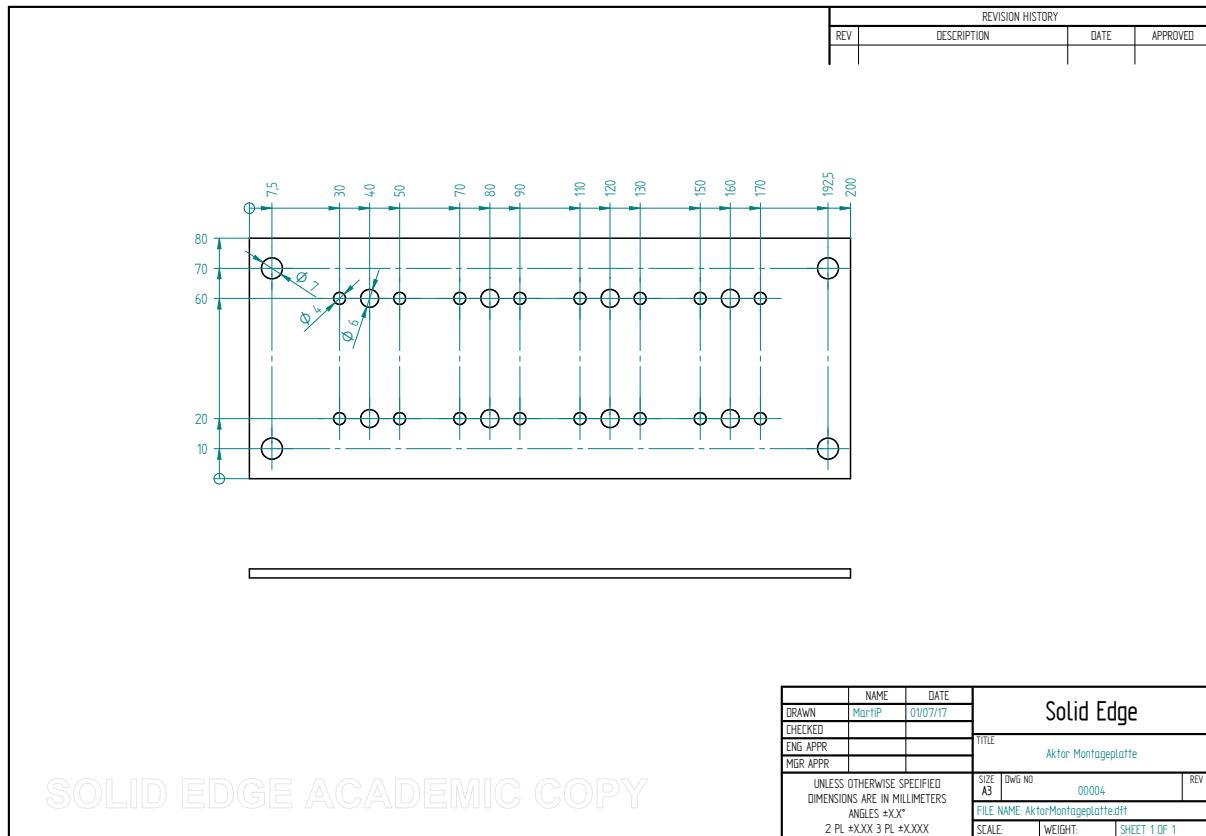


Abbildung C.2.: Plan Aktor Montageplatte.

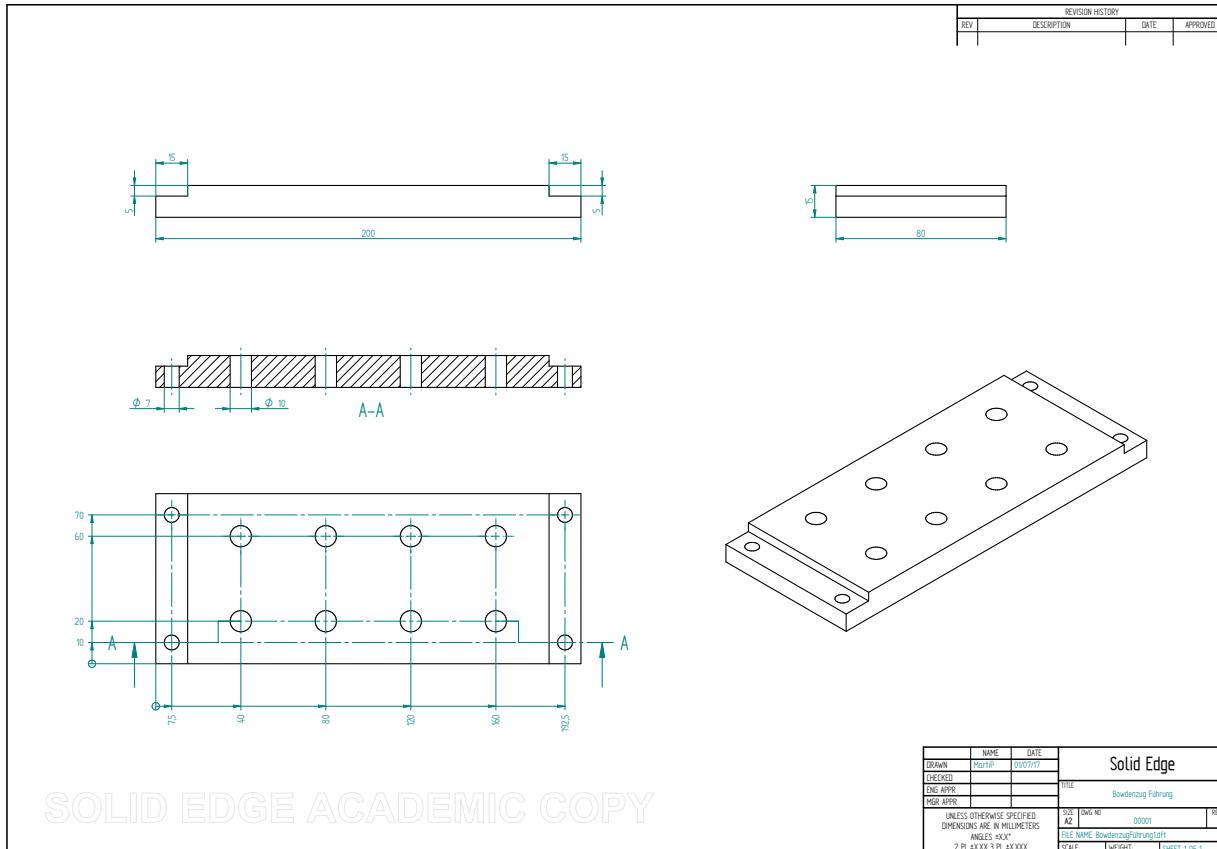


Abbildung C.3.: Plan Bowdenzug-Führung oben.

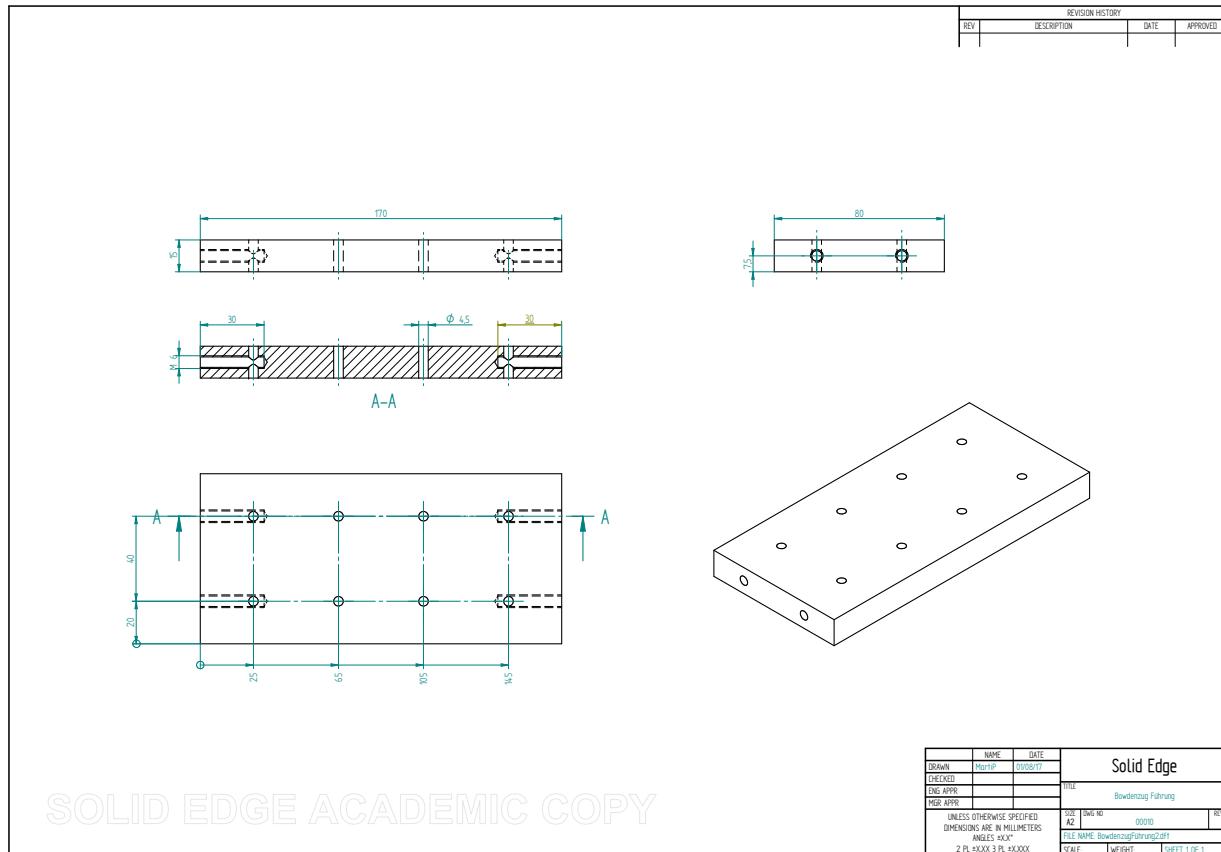
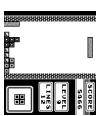


Abbildung C.4.: Plan Bowdenzug-Führungen unten.



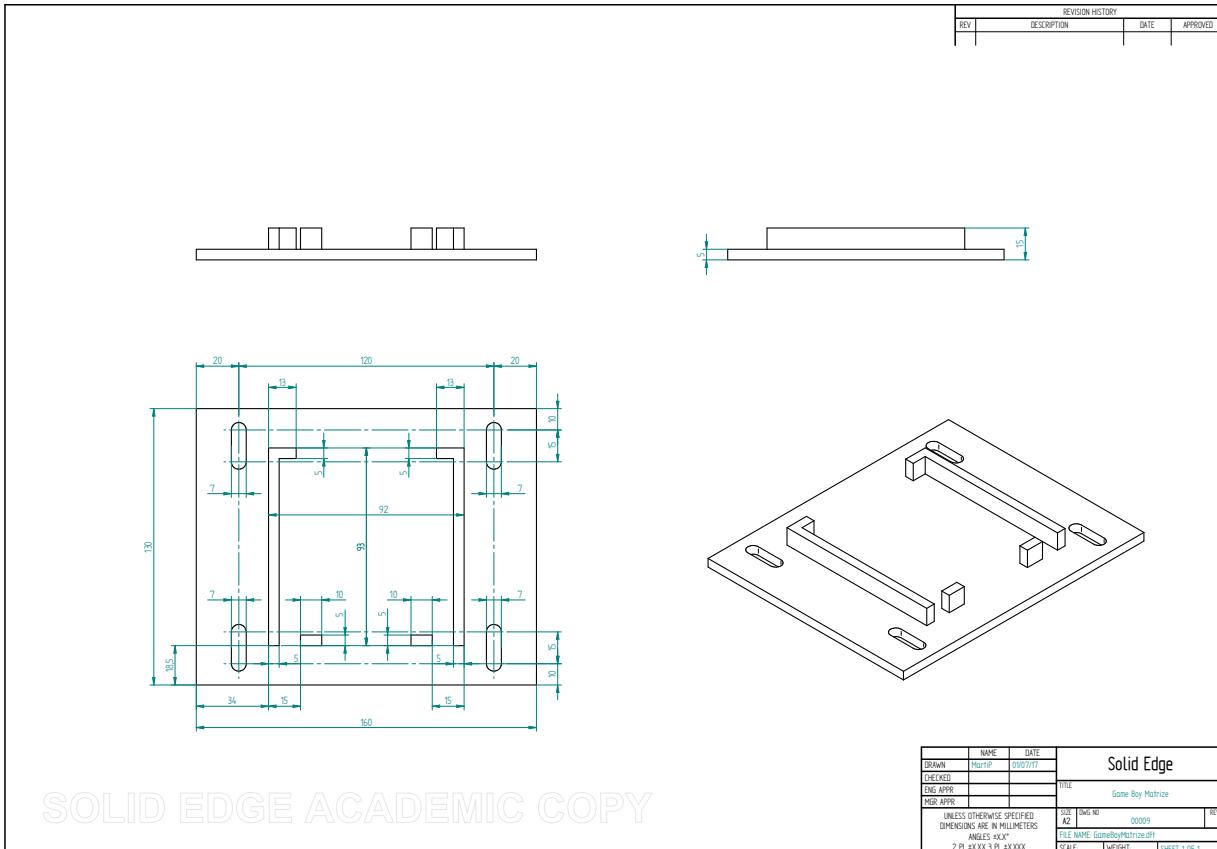


Abbildung C.5.: Plan Game-Boy-Matrize.

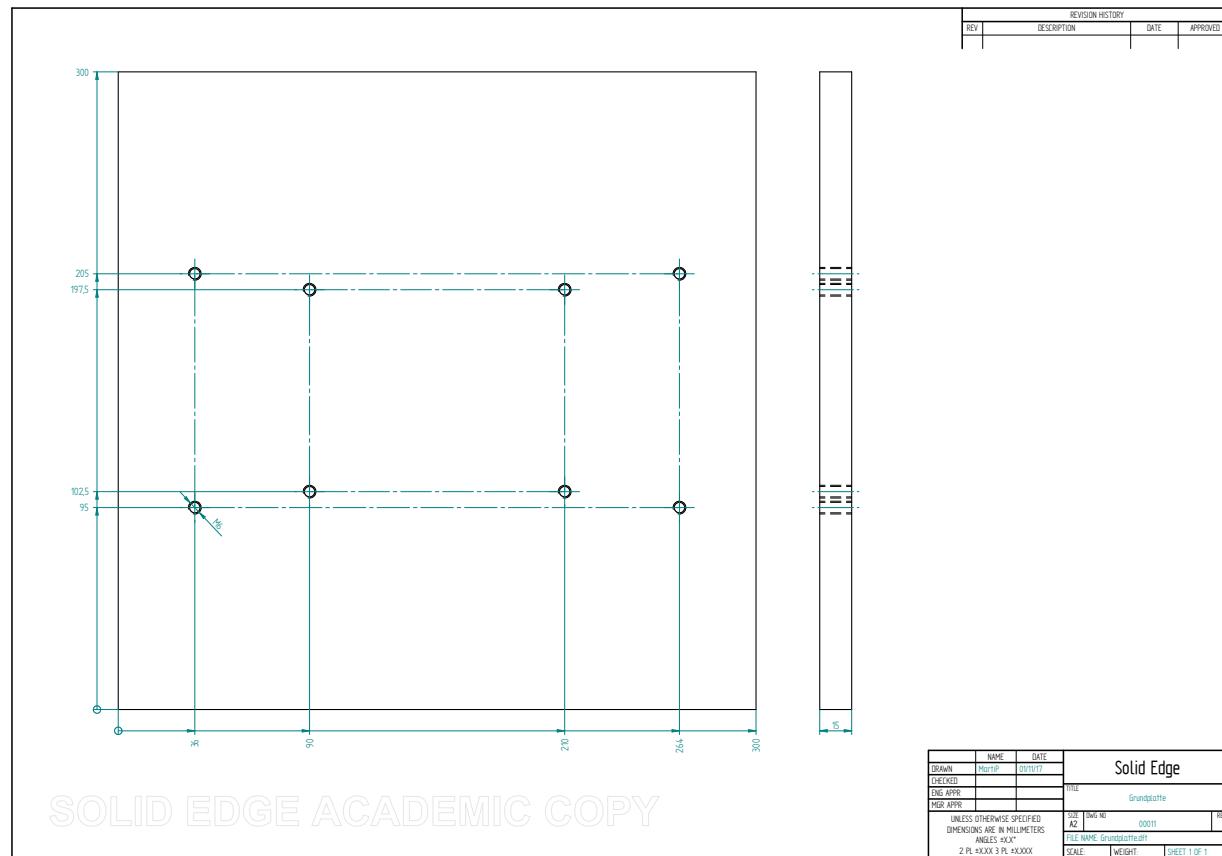


Abbildung C.6.: Plan Grundplatte.

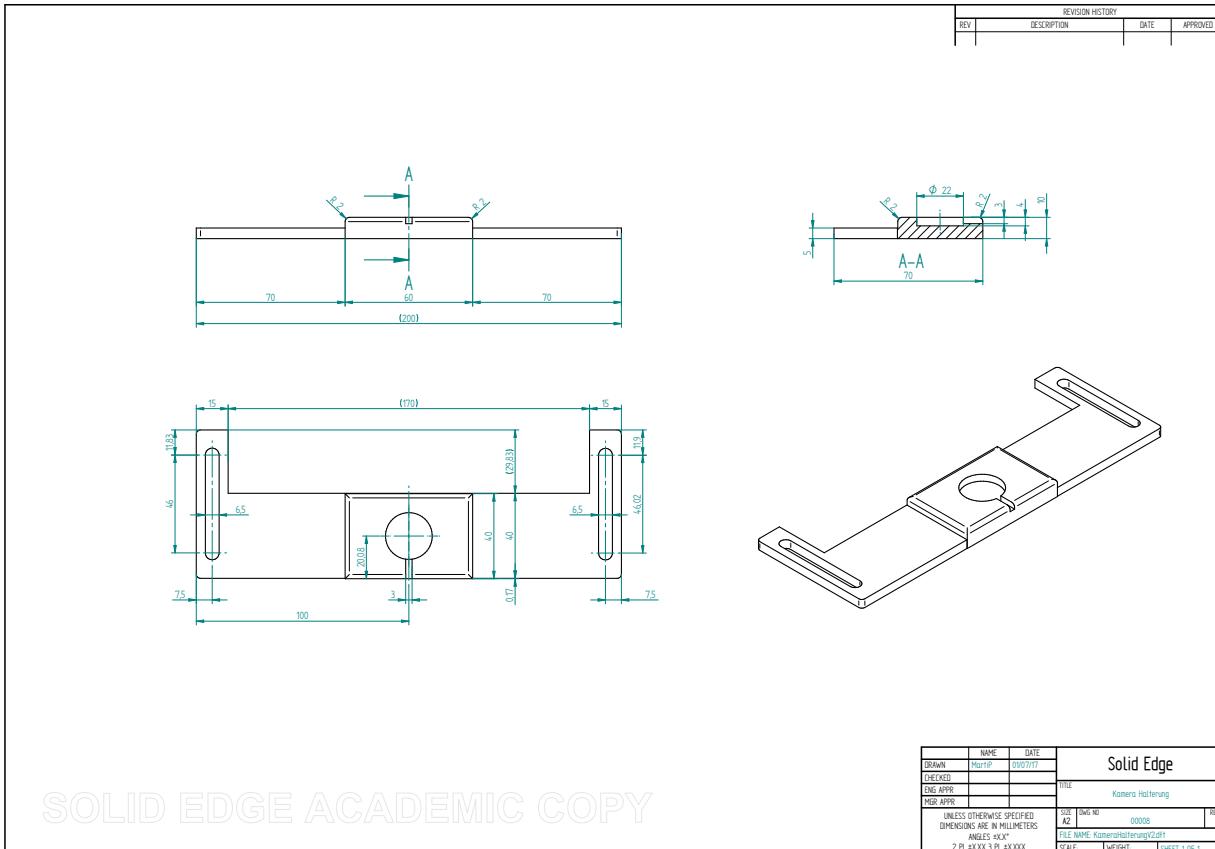


Abbildung C.7.: Plan Kamerahalterung.

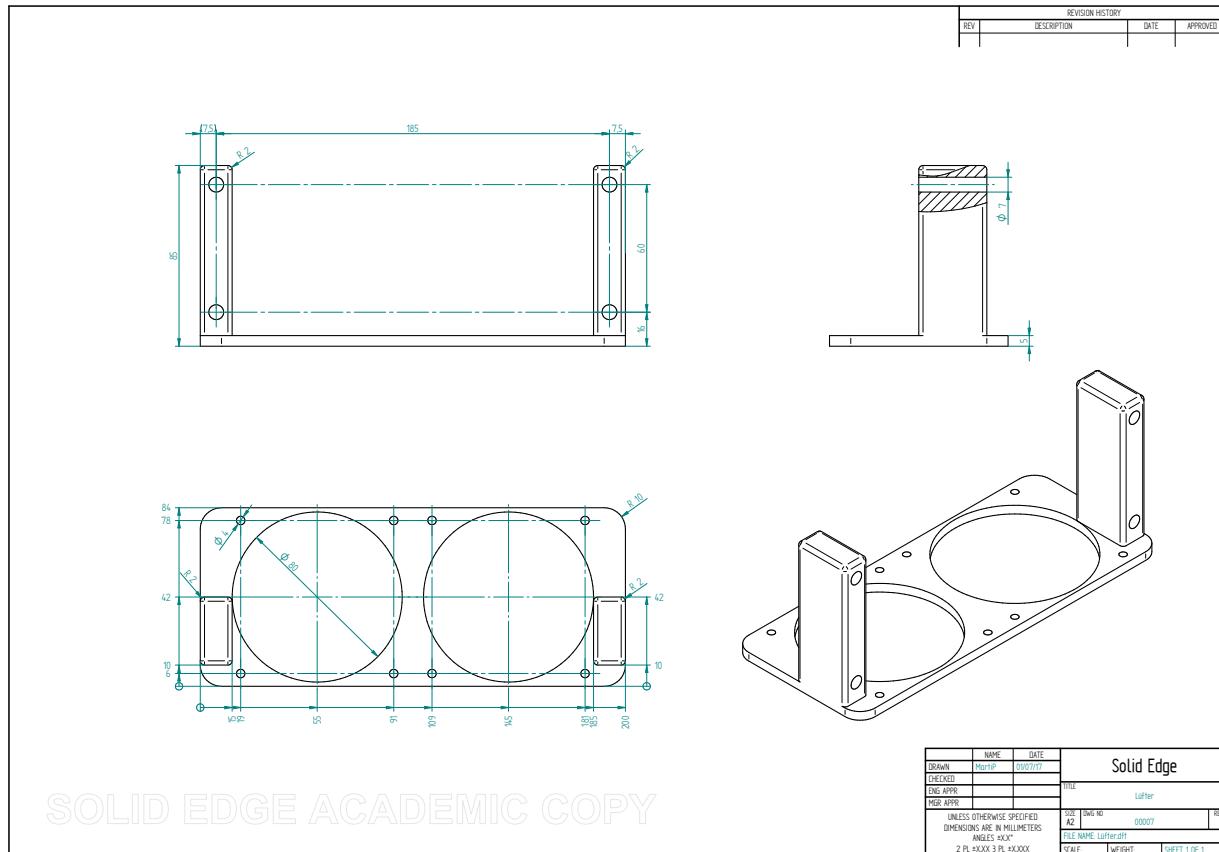


Abbildung C.8.: Plan Lüfterhalterung.

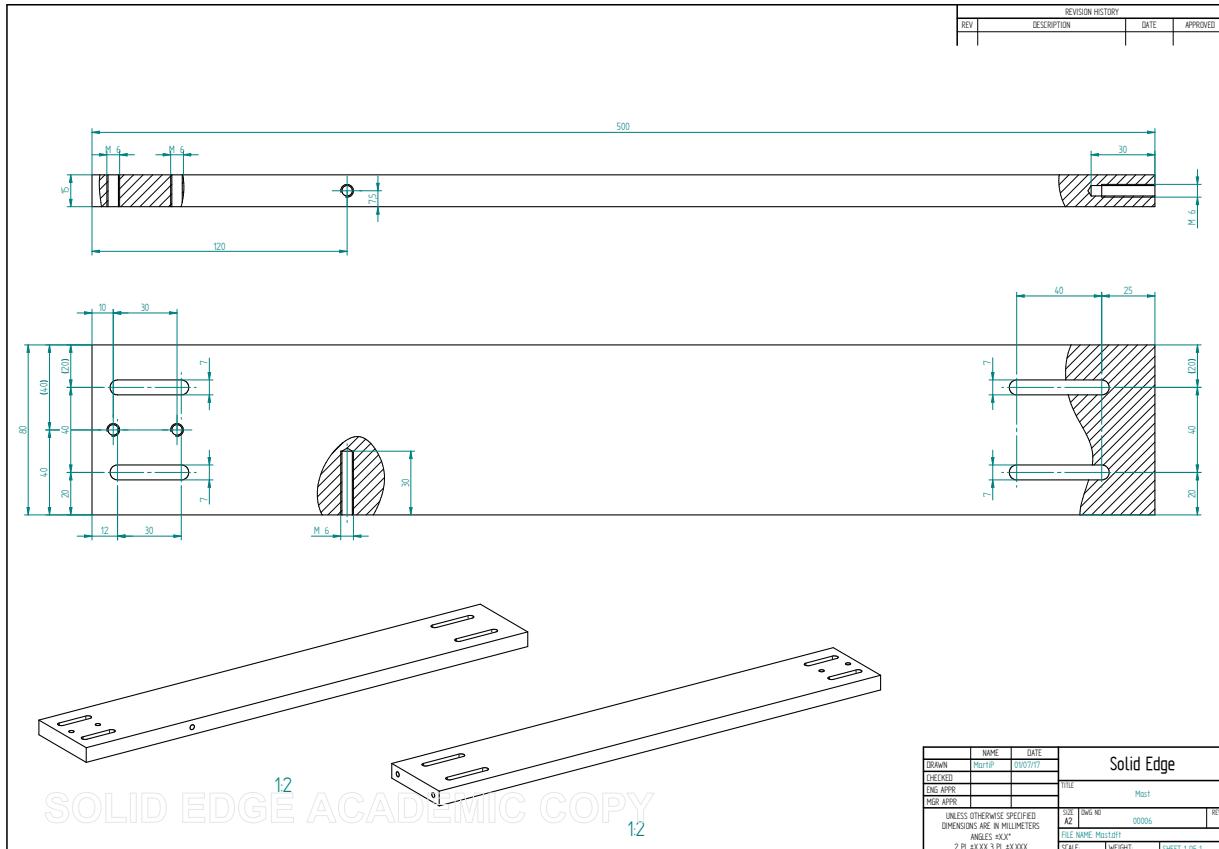


Abbildung C.9.: Plan Mast.

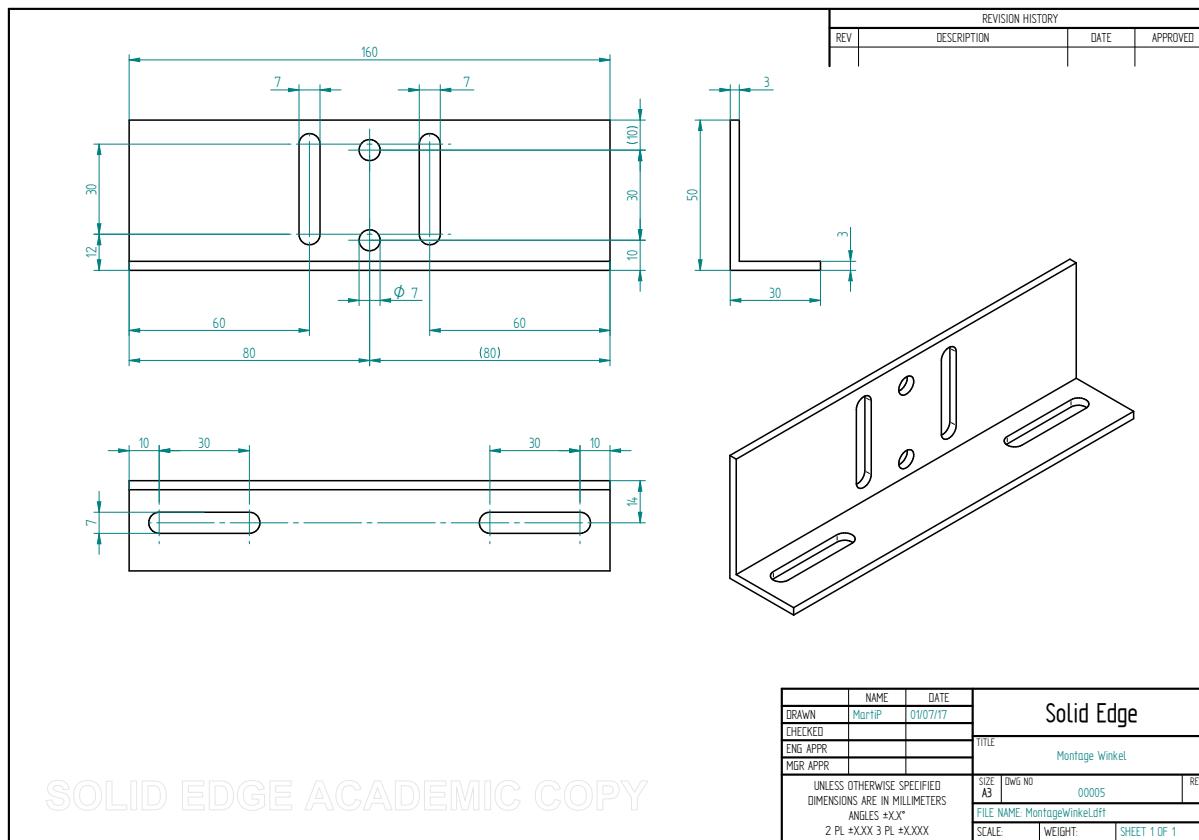


Abbildung C.10.: Plan Montagewinkel.

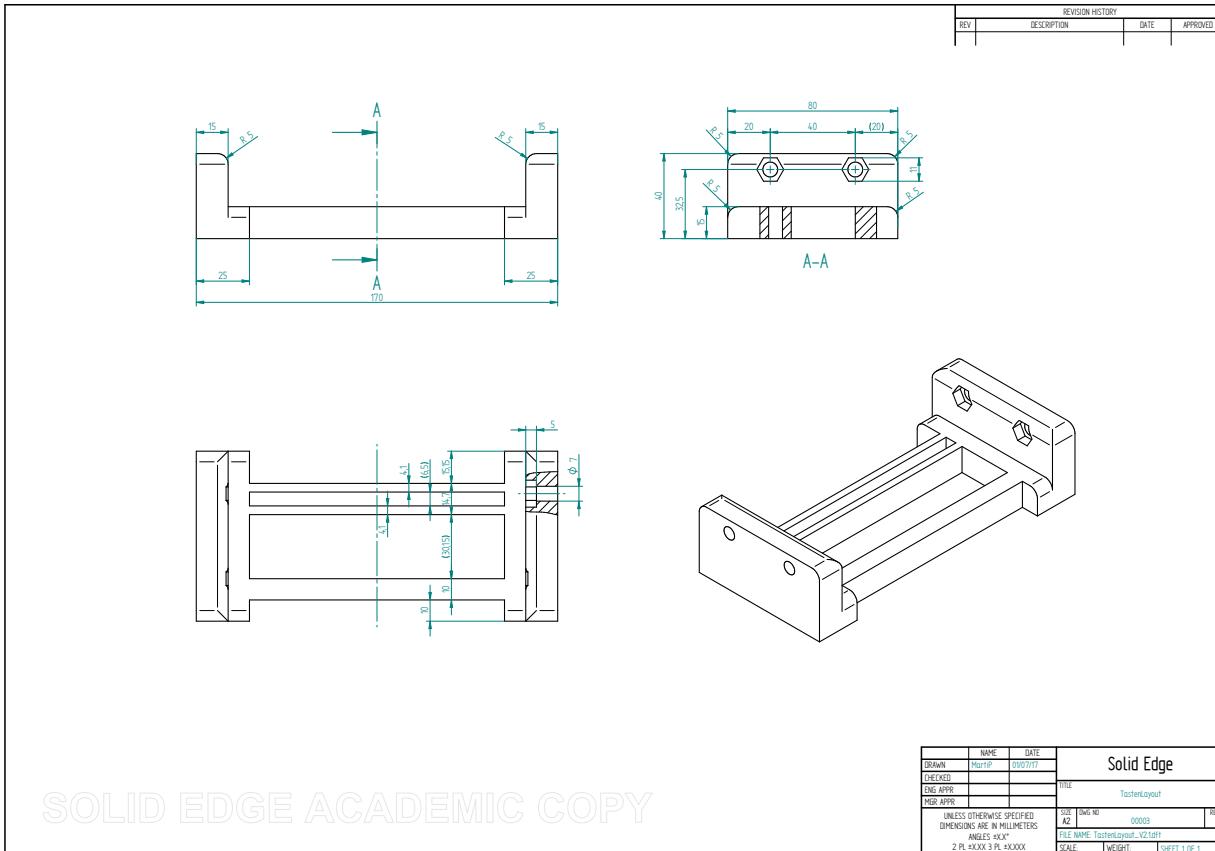


Abbildung C.11.: Plan Tastenlayout Träger.

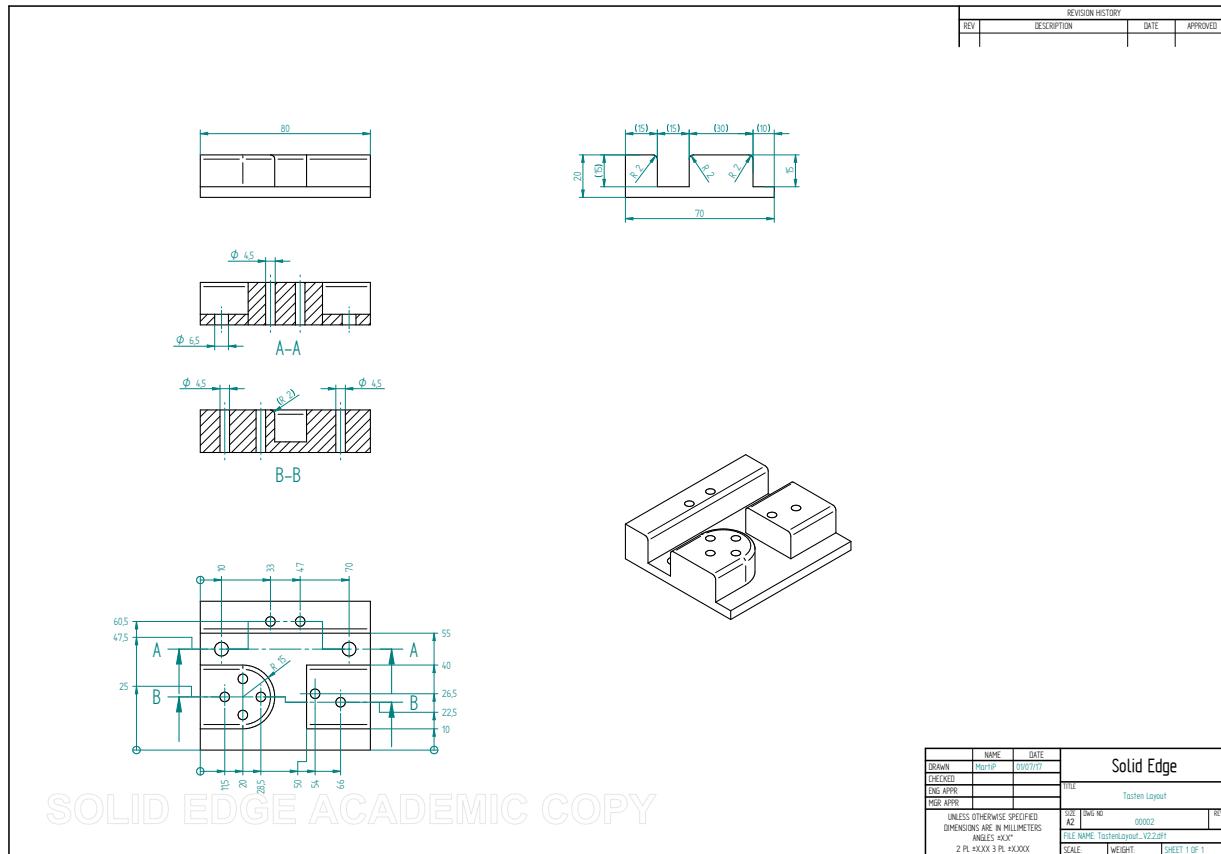
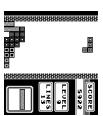


Abbildung C.12.: Plan Tastenlayout.





D. Datenblätter

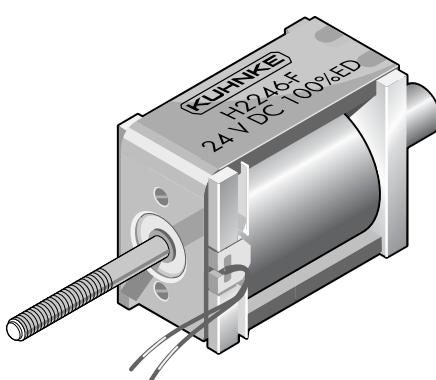
D.1. Hubmagnete

D.1.1. Kuhnke H3268-R-F24V Hubmagnet

Gewicht:	
Magnet:	ca. 65 g
Anker:	ca. 13 g
Standard:	
Spannung:	24 V DC
Litze:	10 cm
Thermische Klasse:	B ($T_{grenz} = 130^\circ\text{C}$)
Isolationsgruppe nach:	VDE 0110 B 75
Prüfspannung:	2500 V (eff)

Hohe Lebensdauer durch Ankerlagerung im Kunststoffspulenkörper.

* Auf Anfrage ist dieser Magnet auch mit wartungsfreier Ankerlagerung (Gleitlager) für höchste Lebensdauer lieferbar.



Weight:	
Complete solenoid:	appr. 65 g
Armature:	appr. 13 g
Standard:	
Voltage:	24 V DC
Flying leads:	10 cm
Thermal stability:	B (max. permissible temperature = 130 °C)
Insulation group according to:	VDE 0110 B 75
Test voltage:	2500 V (eff)

Long life expectancy due to armature bearing in plastic bobbin.

* On request, the solenoid can also be supplied with service-free armature bearing (plain bearing) for maximum durability.

Zul. rel. Einschaltzeit (ED) ⁴⁾	%	100	45	25	15	5	%	Perm. duty cycle (ED) ⁴⁾
Nennaufnahme Pn	W	5,2	10,2	19	29,5	75	W	Nominal coil power Pn
Anzugszeit (ED)	ms	24				7	ms	Actuation time (ED)

⁴⁾ Bei Montage auf eine Kühlfläche von mindestens 45 cm² ist die 1,3fache ED zulässig

⁴⁾ If solenoid is mounted directly onto a flat metal surface of at least 45 cm², the duty cycle can be extended up to 1.3 x nominal rating

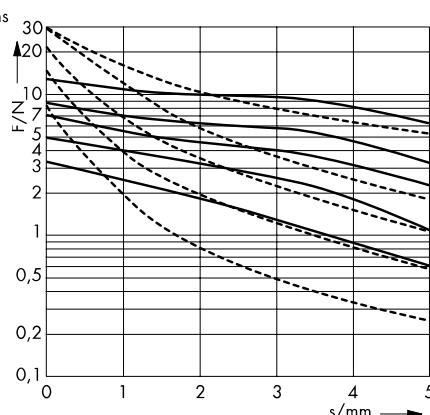
Kraft-Weg-Diagramm F = f (s)

— Konusanker
- - - Flachanker

Kraft bei waagerechter Bewegungsrichtung und bei 90 % Nennspannung und betriebswarmer Wicklung

Hub s = 0 entspricht dem angezogenen, bestromten Zustand

Kraft-Weg-Kennlinien sind ohne Feder gemessen



Force vs. Stroke diagramm F = f (s)

— Conical face armature
- - - Flat face armature

Force measured when operating in horizontal position, at 90 % rated voltage and with winding at operating temperature

stroke s = 0 corresponds to armature in fully home position

Force vs. stroke characteristics measured without return spring

Abbildung D.1.: Datenblatt des Kuhnke H3268-R-F24V Hubmagnet.



E. Code

E.1. Klassendiagramme

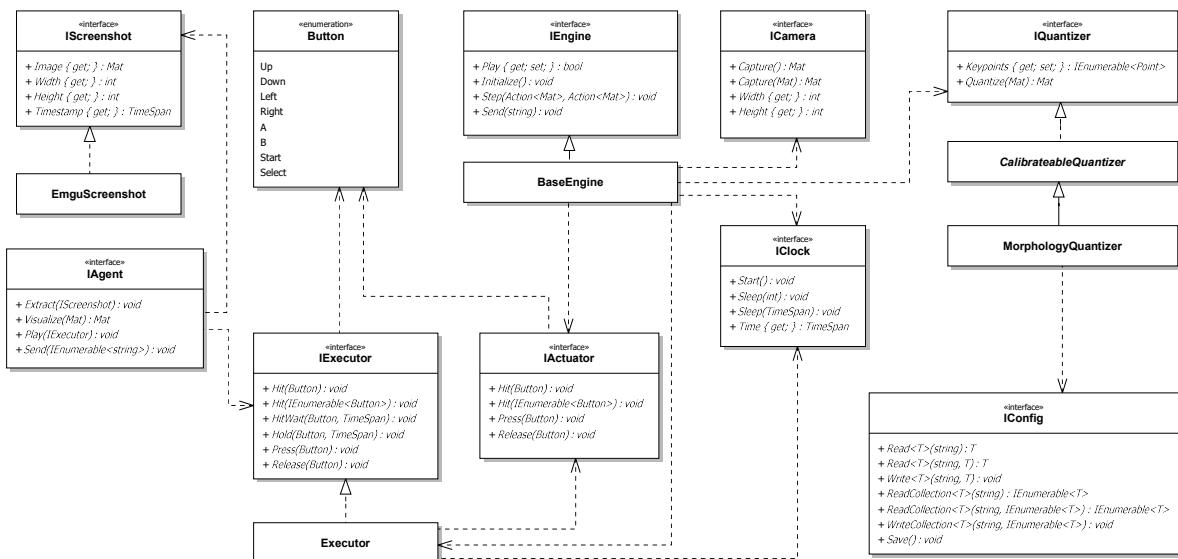


Abbildung E.1.: Klassendiagramm des Assemblies *GameBot.Core*.

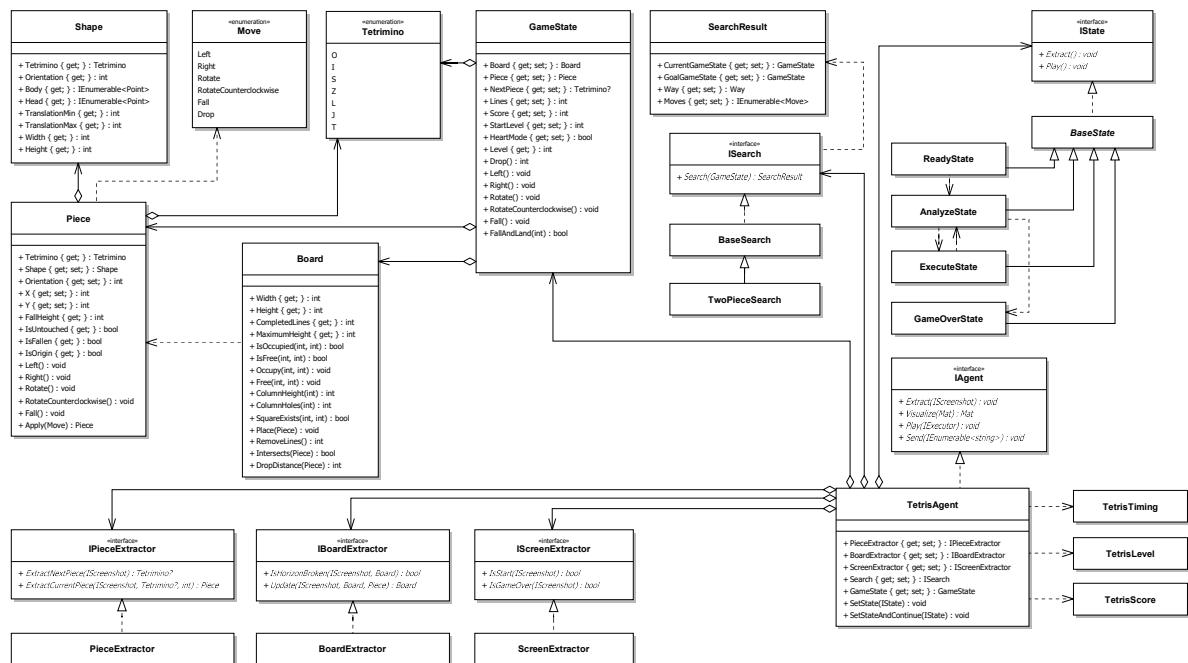


Abbildung E.2.: Klassendiagramm des Assemblies *GameBot.Game.Tetris*.

E.2. Befehle in der Applikation

Hier sind die Befehle aufgelistet, die in der Applikation GameBot.Robot.Ui möglich sind.

Escape Applikation beenden.

Linke Maustaste Einen Keypoint für die Kamerakalibrierung hinzufügen.

K Keypoints für die Kamerakalibrierung zurücksetzen.

P Agent starten.

S Agent stoppen.

R Agent zurücksetzen.

L Roboter wählt konfiguriertes Level aus (im Levelauswahlmenü).

H Roboter schreibt sein Kürzel in die Highscore-Tabelle (im Highscore-Menü).

M Roboter aktiviert den Heart-Mode (Startbildschirm).

G Roboter startet ein neues Spiel (Game-Over-Bildschirm).

Pfeiltaste oben Aktor für die Game-Boy-Taste ↑.

Pfeiltaste unten Aktor für die Game-Boy-Taste ↓.

Pfeiltaste links Aktor für die Game-Boy-Taste ←.

Pfeiltaste rechts Aktor für die Game-Boy-Taste →.

Y Aktor für die Game-Boy-Taste A.

X Aktor für die Game-Boy-Taste B.

Enter Aktor für die Game-Boy-Taste Start.

Backspace Aktor für die Game-Boy-Taste Select.

E.3. Konfigurationsdatei

Der folgende Codeausschnitt stammt aus der Datei App.config und enthält wichtige Konfigurationswerte für die Software.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

  <appSettings>

    <!-- Should the extracted game state be visualized from the graphical engine? -->
    <add key="Game.Tetris.Visualize" value="true" />

    <!-- The level to start from (0 - 9). -->
    <add key="Game.Tetris.StartLevel" value="9" />
    <!-- Play in heart mode? -->
    <add key="Game.Tetris.HeartMode" value="false" />

    <!-- In multiplayer mode, the bot checks for spawned lines on the bottom. -->
    <add key="Game.Tetris.Multiplayer" value="false" />

    <!-- Should the board be checked after every new piece? -->
    <add key="Game.Tetris.Check.Enabled" value="false" />

  </appSettings>
</configuration>
```



```
<!-- Number of image samples in the extraction state. 1 means no sampling. -->
<add key="Game.Tetris.Extractor.Samples" value="3" />

<!-- The probability that must be reached to accept an image for the next piece. -->
<add key="Game.Tetris.Extractor.ThresholdNextPiece" value="0.7" />
<!-- The probability that must be reached to accept an image for the current piece. -->
<add key="Game.Tetris.Extractor.ThresholdCurrentPiece" value="0.7" />
<!-- The probability that must be reached to accept an image for a moved piece. -->
<add key="Game.Tetris.Extractor.ThresholdMovedPiece" value="0.5" />

<!-- The time in ms that is added to calculate the search height. The search height
     must be high enough. -->
<add key="Game.Tetris.Timing.MoreTimeToAnalyze" value="300" />
<!-- This is the average time that passes before a drop can be executed. -->
<add key="Game.Tetris.Timing.LessFallTimeBeforeDrop" value="30" />
<!-- The time in ms that is subtracted after the drop. We do not want to miss important
     frames. Value can be between 0 and 33 ms. -->
<add key="Game.Tetris.Timing.LessWaitTimeAfterDrop" value="33" />

<!-- The path for the game boy rom. Only relevant in the Emulated engine mode. -->
<add key="Emulator.Rom.Path" value="Roms/tetris.gb" />

<!-- The engine mode. Possible values "Emulated" and "Physical". -->
<add key="Robot.Engine.Mode" value="Emulated" />

<!-- The camera source. 0 for integrated camera, 1 for external camera. -->
<add key="Robot.Camera.Index" value="1" />
<!-- Must the image from the camera be rotated by 180 degrees? -->
<add key="Robot.Camera.RotateImage" value="true" />
<!-- Width of the captured camera image. -->
<add key="Robot.Camera.FrameWidth" value="640" />
<!-- Height of the captured camera image. -->
<add key="Robot.Camera.FrameHeight" value="480" />
<!-- Fps of the captured camera image. -->
<add key="Robot.Camera.Fps" value="30" />

<!-- Keypoints to calculate the transformation matrix. Order: top left, top right,
     button left, button right. -->
<add key="Robot.Quantizer.Transformation.KeyPoints" value="0,0,160,0,0,144,160,144" />

<!-- The subtracted constant of the adaptive threshold. -->
<add key="Robot.Quantizer.Threshold.Constant" value="13" />

<!-- The block size of the adaptive threshold. -->
<add key="Robot.Quantizer.Threshold.BlockSize" value="17" />

<!-- Settings for the Tinkerforge API -->
<add key="Robot.Actuator.Host" value="localhost" />
<add key="Robot.Actuator.Port" value="4223" />
<add key="Robot.Actuator.UidMaster" value="6JKbWn" />
<add key="Robot.Actuator.UidRelay1" value="mTA" />
<add key="Robot.Actuator.UidRelay2" value="mTC" />

<!-- The time in ms the Actuator waits between press and release using a hit -->
<add key="Robot.Actuator.Hit.Time" value="35" />
<!-- The time in ms the Actuator waits after a hit -->
<add key="Robot.Actuator.Hit.DelayAfter" value="40" />
```

```
<!-- Frames per second of the live cam stream -->
<add key="Robot.Ui.CamFramerate" value="20" />

<!-- The log level to write (possible values are: "Info", "Warn", "Error") -->
<add key="Robot.Ui.LogLevel" value="Info" />

</appSettings>

</configuration>
```

