DHBW
Duale Hochschule
Baden-Württemberg
Ravensburg
Campus Friedrichshafen

# Realisierung eines Vier-Gewinnt Roboters

## ggf. Untertitel mit ergänzenden Hinweisen

## Studienarbeit T3_3200

Studiengang Elektrotechnik

Studienrichtung Automation

Duale Hochschule Baden-Württemberg Ravensburg, Campus Friedrichshafen

von

Simon Gschell / Patrik Peters

| | |
|---|---|
| Abgabedatum: | 8. Juli 2025 |
| Bearbeitungszeitraum: | 01.01.2025 - 31.06.2025 |
| Matrikelnummer: | 123 456 |
| Kurs: | TEA22 |
| Betreuer: | Prof. Dr. ing Thorsten Kever |

# Erklärung

gemäß Ziffer 1.1.14 der Anlage 1 zu §§ 3, 4 und 5 der Studien- und Prüfungsordnung für die Bachelorstudiengänge im Studienbereich Technik der Dualen Hochschule Baden-Württemberg vom 29.09.2017 in der Fassung vom 24.07.2023.

Ich versichere hiermit, dass ich meine Studienarbeit T3_3200 mit dem Thema:

*Realisierung eines Vier-Gewinnt Roboters*

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Friedrichshafen, den 8. Juli 2025

---

Simon Gschell / Patrik Peters

# Kurzfassung

Diese Studienarbeit wurde im Rahmen der sechsten Akademiephase angefertigt. Im ersten Teil der Arbeit wurden verschiedene Spieltheorien miteinander verglichen. Aufbauend auf den Erkenntnissen der ersten Studienarbeit, lag der Schwerpunkt diesmal in der praktischen Umsetzung eines Vier-Gewinnt-Roboters. Das Ziel dieser Arbeit bestand darin, einen Roboter zu entwickeln, der eigenständig Spielzüge beim Spiel „Vier gewinnt“ ausführen kann und somit in der Lage ist, gegen einen anderen Roboter anzutreten.

Für die Realisierung des Projekts wurde ein LEGO Spike Prime Set verwendet. Die Konstruktion des Roboters besteht aus verschiedenen zusammengesetzten LEGO-Bauelementen. Vereinzelt wurden auch Elemente selbst entworfen und mittels 3D-Drucker angefertigt. Die Steuerung und Überwachung der Aktoren, wie zum Beispiel der Motoren und Sensoren, erfolgt mithilfe eines LEGO Spike Hub. Dieser Mikrocontroller verarbeitet die eingehenden Sensordaten und steuert die Bewegungen des Roboters entsprechend der programmierten Logik.

Die Programmierung erfolgte in der Sprache MircoPython in der LEGO Spike App. Das Kernstück des Programms ist der Alpha-Beta-Algorithmus, mit ihm wird der nächste optimale Zug berechnet. Durch die Kombination aus mechanischer Konstruktion und programmierter Software entstand ein funktionsfähiger Prototyp, der die gestellten Anforderungen erfüllt und einen Spielzug eigenständig ausführen kann.

# Abstract

This student research project was carried out during the sixth academy phase. In the first part of the project, various game theories were examined and compared. Building on the insights from that initial work, the focus this time was on the practical development of a Four-in-a-Row robot. The goal was to create a robot capable of making its own moves in the game "Connect Four,ällowing it to compete against another robot.

The project was implemented using a LEGO Spike Prime set. The robot itself was built from a range of LEGO components, with some parts specially designed and produced using a 3D printer. The motors and sensors are managed by a LEGO Spike Hub, which acts as the robot's microcontroller. This hub processes sensor data and directs the robot's movements according to the programmed logic.

Programming was done in MicroPython using the LEGO Spike app. At the heart of the software is the alpha-beta algorithm, which determines the best possible move at each turn. By combining mechanical design with custom software, the project resulted in a working prototype that meets the requirements and can play the game autonomously.

# Inhaltsverzeichnis

# 1 Einleitung

# 2 Grundlagen

In diesem Kapitel werden die zentralen Begriffe und Methoden ausführlich vorgestellt. So wird das notwendige Grundlagenwissen vermittelt, auf dem die weitere Ausarbeitung aufbaut.

## 2.1 Vier-Gewinnt

Das Spiel Vier-Gewinnt wird auf einem Spielfeldraster mit sechs Zeilen und sieben Spalten gespielt. Zum Spielbeginn erhält jeder Spieler 21 Spielchips, entweder Rote oder Gelbe. Ziel des Spiels ist es, möglichst schnell vier Chips der eigenen Farbe in eine Reihe zu bringen – waagrecht, senkrecht oder diagonal. Die Spieler werfen abwechselnd ihre Chips in das Spielfeld, bis entweder ein Spieler das Ziel erreicht oder alle 42 Felder belegt sind [Has20].

## 2.2 Alpha-Beta-Pruning-Algorithmus

Alpha-Beta-Pruning ist ein Verfahren, das bei Spielen wie Schach, Dame oder Vier Gewinnt eingesetzt wird, um den optimalen nächsten Zug zu bestimmen. Ziel des Algorithmus ist es, die Suche im Spielbaum effizienter zu machen. Im Unterschied zum Minimax-Algorithmus werden beim Alpha-Beta-Pruning gezielt Teilbäume weggelassen, die für das Endergebnis keine Rolle spielen. Während der Tiefensuche durch

den Spielbaum arbeitet der Algorithmus mit zwei Schrankenwerten, dem Alpha ($\alpha$) und dem Beta ($\beta$). Zu Beginn wird Alpha auf $-\infty$ und Beta auf $+\infty$ gesetzt. An jedem MAX-Knoten wird das Maximum aus dem bisherigen Alpha und den Werten der Nachfolgeknoten ausgewählt. Das bedeutet, Alpha wird immer dann erhöht, wenn ein nachfolgender Knoten einen höheren Wert liefert als das aktuelle Alpha. Am MIN-Knoten hingegen wird Beta jeweils auf das Minimum aus dem bisherigen Beta und den Werten der Nachfolgeknoten gesetzt. Sobald an einem Knoten die Bedingung $\alpha \geq \beta$ erfüllt ist, wird der restliche Teilbaum nicht weiter betrachtet. In diesem Fall hat der MIN bereits eine bessere Alternative gefunden, sodass MAX diesen Zweig des Baums nicht mehr wählen würde [Ado09].

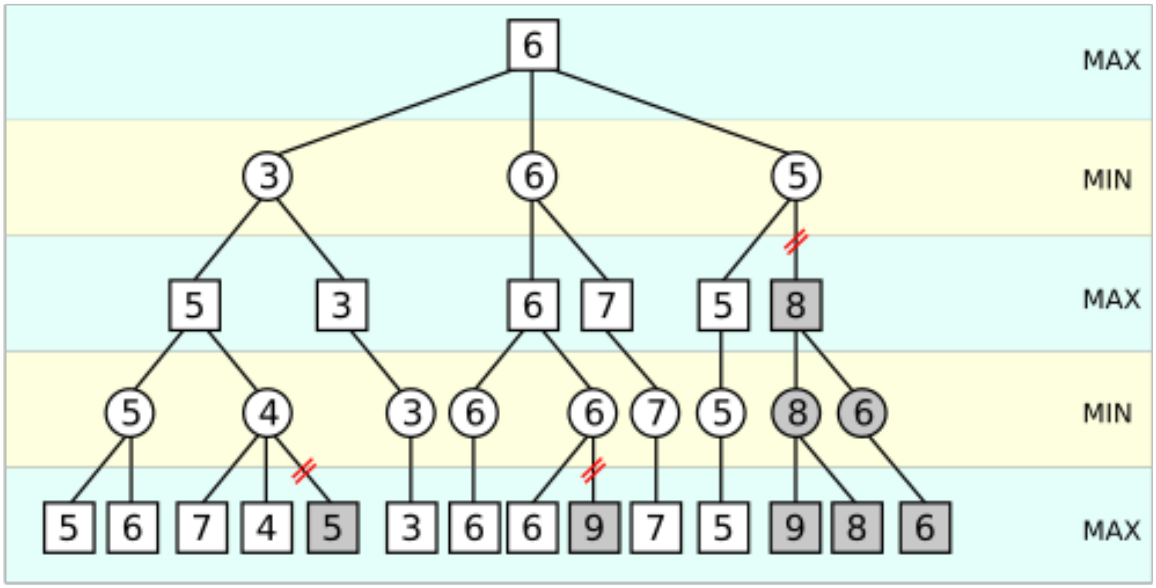


**Abbildung 2.1:** Alpha-Beta Spielbaum

## 2.3 Mikropython

MicroPython ist eine speziell für Mikrocontroller angepasste Version der Programmiersprache Python. Im Gegensatz zur Desktop-Variante lässt sich MicroPython-Code direkt auf Hardware mit begrenzten Ressourcen ausführen[SS23][PLJ23]. Im Unterschied zu Standard-Python 3 bringt MicroPython allerdings nur einen Teil der

gewohnten Python-Standardbibliotheken mit.
Wodurch es weniger Speicherplatz benötigt. Zudem besitzt MicroPython einen eigenen Interpreter, der direkt auf einem Mikrokontroller ausgeführt werden kann. Dadurch eignet sich MicroPython besonders gut für die Programmierung des LEGO Spike Hubs[Bel24].

## 2.4 LEGO Spike Hub:

Der LEGO Spike Hub ist das Herzstück des LEGO Spike Prime Sets. Er dient als programmierbare Steuereinheit mit sechs LPF2 input/output ports, an die alle LEGO-Sensoren und -Motoren angeschlossen werden können. Im Inneren arbeitet ein eigener Prozessor (100 MHz ARM Cortex-M4), unterstützt von 320 KB RAM und 1 MB Flash-Speicher. Die Programmierung des LEGO Spike Hubs erfolgt in der Sprache MicroPython. LEGO stellt dafür eine eigene Entwicklungsumgebung (IDE) bereit, mit dieser der Hub einfach programmiert werden kann. Hierfür kann der Hub über USB oder via Bluetooth mit dem Computer verbunden werden. Die Steuereinheit wird über einen wiederaufladbarer Lithium-Ionen-Akku mit Strom und Spannung versorgt [LEG20b].

Weitere technische Merkmale des LEGO Spike Hubs sind:

- Individuell anpassbaren Lichtmatrix (5x5)
- Lautsprecher
- Taster mit integrierter Statusleuchte
- Tasten für die Navigation und Steuerung durch Menüs am Hub
- Lautsprecher

- sechsachsiger Gyrosensor

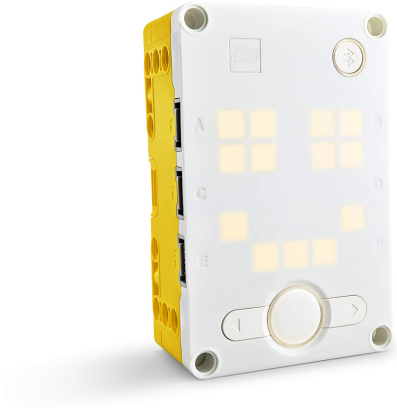

**Abbildung 2.2:** LEGO Spike Hub

## 2.5 Sensorik

### 2.5.1 LEGO Spike Kraft- oder Tuchsensor

Dieser Sensor erkennt, ob er gedrückt wurde, und misst dabei gleichzeitig die auf ihn ausgeübte Kraft. Mit einer Abtastrate von 100 Hz erfasst er Kräfte im Bereich von 2,5 bis 10 Newton und arbeitet dabei mit einer Genauigkeit von $\pm 0,65$ Newton. Der gemessene Wert wird als Prozentwert ausgegeben, wobei 100% einem Tastendruck von 10 Newton entsprechen. Typischerweise wird der Sensor zum Erkennen von Hindernissen oder als Start- bzw. Stopptaste eines Roboters eingesetzt. Der Kraft- oder Tuchsensor wird direkt am LEGO Spike Hub angeschlossen [LEG20c].

**Abbildung 2.3:** LEGO Spike Kraft- oder Tuchsensor

### 2.5.2 LEGO Spike Farbsensor

Dieser elektronische Farbsensor wurde speziell für LEGO Spike entwickelt. Seine Abtastrate beträgt 1 kHz und er kann direkt am Hub angeschlossen werden. Der Sensor kann bis zu acht verschiedene Farben erkennen, darunter Schwarz, Blau, Rot, Weiß, Braun, Gelb, Pink und Grün. Außerdem misst er sowohl die Intensität des reflektierten Lichts als auch die des Umgebungslichts [LEG20a][LEG20c].

Für die Farberkennung erfasst der Sensor die Farbwerte sowohl im RGB- (Rot, Grün, Blau) als auch im HSV-Farbraum (hue = Farbton, saturation = Sättigung, value = Helligkeit). Die Messergebnisse werden als Ganzzahlen ausgegeben [LEG20a].

Zur Reflexionsmessung sendet der Sensor weißes Licht auf eine Oberfläche und misst das zurückgeworfene Licht. Diese Funktion wird häufig für Linienführung eingesetzt [Bet25][LEG20a].

**Abbildung 2.4:** LEGO Technic Farbsensor

## 2.6 Aktorik

### 2.6.1 LEGO Spike Winkelmotor

Der LEGO Spike Winkelmotor nicht nur ein einfacher Elektromotor, aufgrund eines integrierten Drehsensors kann er nicht nur die Drehrichtung, sondern auch die relative und absolute Position (in Grad) sowie die Drehgeschwindigkeit erfassen. Eine vollständige Umdrehung wird dabei in 360 einzelne Zählimpulse unterteilt, wobei die Genauigkeit des Motors bei ±3 Grad liegt. Muss der Motor ein Drehmoment von mehr als 5 Ncm aufbringen, blockiert er. Mit einer Abtastrate von 100 Hz erfasst der Motor die Position sowohl beim automatischen als auch im manuellen Betrieb. Der LEGO Spike Winkelmotor eignet sich somit nicht nur für Bewegungsaufgaben, sondern auch zur Positionsbestimmung [LEG20c].

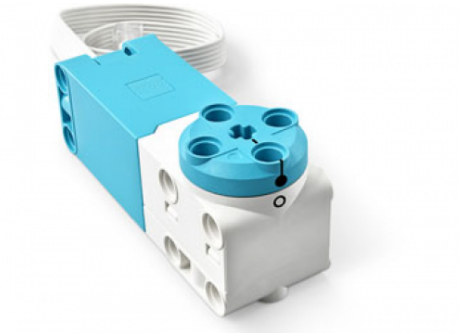

**Abbildung 2.5:** LEGO Spike Winkelmotor

# 3 Vorgehen

Im folgendem Kapitel wird auf die Planung des Vier-Gewinnt-Roboters eingegangen. Zunächst werden die Anforderungen definiert. Anschließend werden verschiedene Konzepte genauer betrachtet und miteinander verglichen.

## 3.1 Aufgabenpräzisierung

Für das Projekt soll mit LEGO Spike Prime ein Roboter entwickelt werden, der Spielzüge im Spiel Vier-Gewinnt vollkommen eigenständig gegen einen menschlichen Gegner oder einen anderen Roboter spielen kann. Der Roboter muss dazu in der Lage sein, die auf dem Spielfeld platzierten gelben und roten Spielsteine zuverlässig zu erkennen und deren Positionen zu erfassen, um das gesamte Spielfeld systematisch auswerten zu können.
Nach Abschluss des Scanvorgangs berechnet der Roboter mithilfe eines passenden Algorithmus die nächste optimale Position. An der ermittelten Stelle platziert er anschließend eigenständig den nächsten Spielstein. Es ist wichtig, dass der Roboter flexibel ist und sowohl mit gelben als auch mit roten Steinen spielen kann. Außerdem muss es möglich sein, dass der Roboter das Spiel entweder eröffnet oder als zweiter Spieler startet.
Für jeden Spielzug gilt eine maximale Zeitvorgabe von 90 Sekunden, die nicht überschritten werden darf. Der Roboter zieht sich nach jedem Zug komplett vom Spielfeld zurück und wartet darauf, dass der Gegner seinen Spielzug komplett abgeschlossen hat. Erst danach startet der Roboter erneut mit dem Scannen des Spielfelds und der darauf folgenden Berechnung des nächsten Zuges.

## 3.2 Anforderungen an den Roboter

In der Tabelle 3.1 sind die Anforderungen an den Vier-Gewinnt-Roboter in einer Anforderungsliste zusammengetragen. Dabei wird zwischen Forderungen und Wünschen unterschieden. Anforderungen, die mit ***F*** gekennzeichnet sind, müssen unbedingt umgesetzt werden. Während hingegen Anforderungen, die mit ***W*** markiert sind, als Wünsche zu verstehen sind und nicht zwingend im System realisiert werden müssen.

**Tabelle 3.1:** Anforderungstabelle für einen Vier-Gewinnt-Roboter

| **Nr.** | **Anforderung an das System** | **F/W** |
|---|---|---|
| - | komplettes Spielfeld scannen | F |
| - | das Ende des Spiels erkennen | F |
| - | autonom fahren | F |
| - | Begrenzungen des Spielfelds erkennen | F |
| - | Steine selber platzieren | F |
| - | immer nur ein Stein pro Spielzug | F |
| - | Abwarten, bis der Gegner seinen Zug beendet hat | F |
| - | nach jedem Zug rechts vom Spielfeld wegfahren | F |
| - | maximal 90 Sekunden pro Spielzug | F |
| - | optimalen Spielzug berechnen | F |
| - | es sollte möglich sein sowohl mit Gelb als auch Rot zu spielen | F |
| - | sowohl als Erster als auch als Zweiter zu starten | F |
| - | während des gegnerischen Spielzugs warten | F |
| - | Scannen, nur bis ein neuer gegnerischer Stein erkannt wurde | W |
| - | volle Spalten überspringen | W |
| - | wenn ein leerer Platz erkannt wurde, zur nächsten Spalte | W |
| - | | F |

**Legende: F** = Forderung, **W** = Wunsch,

## 3.3 Konzept

| | Variante 1 | Variante 2 | Variante 3 |
|---|---|---|---|
| Berechnung | auf dem Computer | auf dem Controller | |
| Rückmeldung | Computer | LED-Matrix | Akustisches Signal |
| Programmiersprache | MicroPython | Scratch | |
| Algorithmus | Zufall | Minimax/Alpha-Beta | Vorprogrammierte Züge |
| Verbindung zum PC | Bluetooth | USB Kabel | |
| Art der Start-betätigung | vom Computer | Kraftsensor | Taste am Hub |
| Spielfeld-Erkennung | Farbsensor | Manuelle Eingabe | |
| Steinplatzierung | Greifarm | Fallmechanismus/ Rutsche | Förderband |

**Tabelle 3.2:** Morphologischer Kasten

# 4 Umsetzung und Ergebnisse

## 4.1 Mechanischer Aufbau des LEGO-Spike-4-Gewinnt-Roboters

Für die Umsetzung des 4-Gewinnt-Roboters wurde eine mechanische Konstruktion gewählt, die es erlaubt, das Spielfeld zu scannen sowie Chips gezielt in eine Spalte einzuwerfen. Der Aufbau umfasst drei LEGO Spike-Motoren, einen Farbsensor und einen Drucktaster. Im Folgenden werden die einzelnen Komponenten detailliert beschrieben.

- **Horizontalantrieb – Motor D**
  Der horizontale Antrieb des Farbsensors erfolgt über Motor D. Dieser ist dafür zuständig, die Spielfeldspalten nacheinander anzufahren. Der Motor ist mit einer Achse verbunden, welche zwei Räder antreiben. Die Bewegung erfolgt in gleichmäßigen Schritten: Eine Drehung um exakt 72 Grad bewegt den Schlitten um eine Spalte weiter. Diese Schrittweite wurde so gewählt, dass sie der Breite einer Spalte im Spielfeld entspricht. Dadurch ist eine exakte Positionierung des Sensors über jeder Spalte möglich, ohne dass zusätzliche Sensoren zur Positionsbestimmung notwendig sind.

- **Vertikalantrieb – Motor E, Kette und Farbsensor**
  Um das Spielfeld auch in vertikaler Richtung abfahren zu können, ist der Farbsensor an einer Kette montiert. Diese Kette wird durch Motor E angetrieben. Der Sensor ist an einem mittleren Segment der Kette befestigt und fährt beim

Drehen der Kette entsprechend auf und ab. Ein Schritt des Motors um etwa 95 Grad bewegt den Sensor um genau eine Spielfeldhöhe weiter. Auf diese Weise können sämtliche sechs Reihen der aktuellen Spalte nacheinander abgescannt werden. Der Sensor wurde dabei so montiert, dass er exakt über der Mitte jedes Feldes positioniert ist, um eine zuverlässige Farberkennung zu ermöglichen. Die Rückwärtsbewegung der Kette erlaubt es, den Sensor wieder nach unten zu fahren.

- **Chipauswerfer – Motor A**
  Das Einwerfen des eigenen Spielsteins erfolgt über Motor A. An diesem Motor ist eine Stange montiert, die bei einer vollständigen Umdrehung einen Spielchip aus dem Vorratsmagazin in die gewünschte Spalte stößt. Nach der Auslösung kann ein neuer Chip in die Abschussposition nachrutschen. In der Software ist eine Wartezeit nach dem Auslösen eingebaut, damit der Chip sicher im Spielfeld ankommt, bevor die nächste Aktion beginnt.

- **Startsignal – Drucksensor (Force Sensor)**
  Um dem Roboter mitzuteilen, dass der menschliche Spieler seinen Zug abgeschlossen hat, wurde ein Drucksensor verwendet. Dieser befindet sich an der Vorderseite des Roboters. Sobald der Spieler den Sensor leicht berührt, wird ein Signal ausgelöst, das Prozess startet.

- **Spielfeldscan – Farbsensor an Kette**
  Für die Farberkennung des Spielfeldes wurde ein LEGO-Farbsensor verwendet, der über die oben beschriebene Kettenkonstruktion vertikal verfahrbar ist. Die Farbmessung erfolgt jeweils in der Mitte eines Spielfeldes. Der Sensor erkennt RGB-Werte, die per Software verschiedenen Spielsteinfarben (in diesem Projekt benutzt: Rot, Gelb oder Leer) zugeordnet werden. Während des Spiels vergleicht der Algorithmus die gemessenen Werte mit diesen Referenzwerten, um die tatsächliche Belegung jedes Feldes möglichst robust zu bestimmen. Der Abstand zwischen Sensor und Spielfeld beträgt etwa 7 mm – dieser Wert hat sich als optimal für zuverlässige Farbmessung erwiesen.

**Zusammenfassung**

Die mechanische Konstruktion basiert auf einem kartesischen Koordinatensystem, bei dem der Sensor durch die Kombination aus horizontaler (Motor D) und vertikaler Bewegung (Motor E + Kette) jede Spielfeldposition präzise anfahren kann. Das System erlaubt eine vollautomatische Spielweise: Der Roboter erkennt die aktuelle Spielsituation, berechnet den optimalen Zug und setzt diesen physisch um.

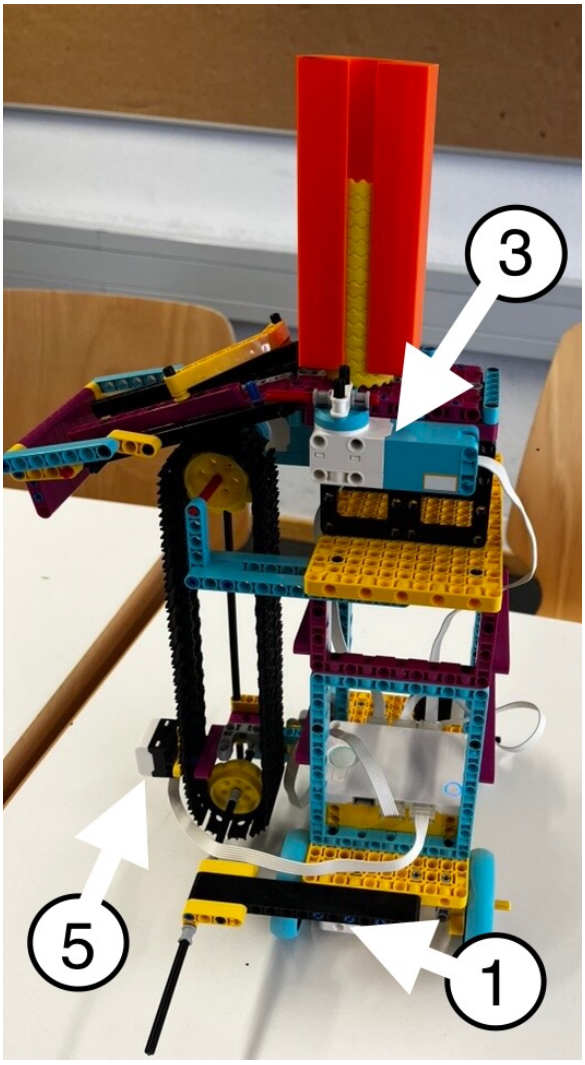


**Abbildung 4.1:** Seitenansicht links

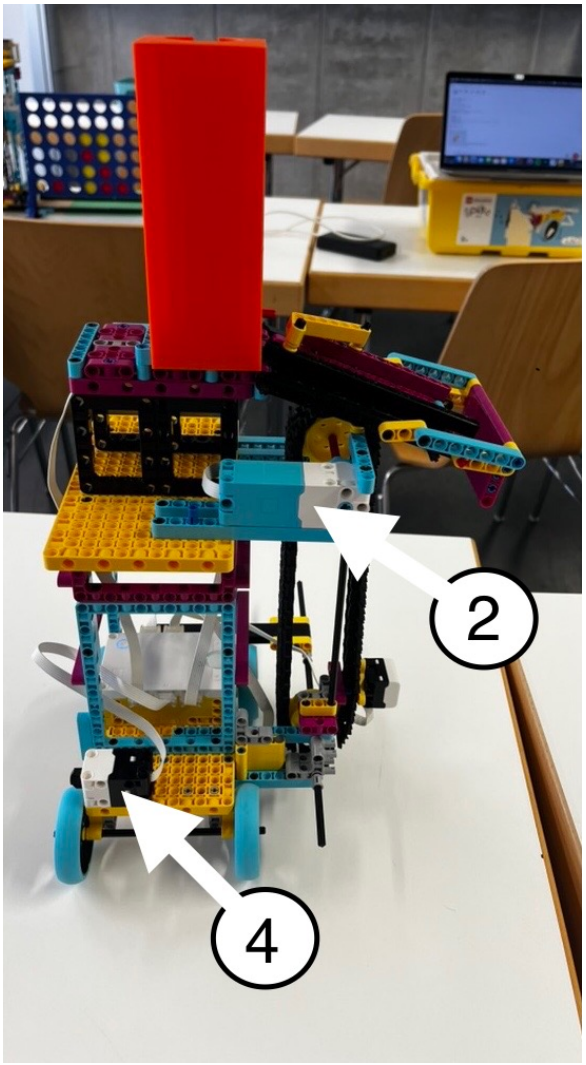


**Abbildung 4.2:** Seitenansicht rechts

## 4.2 Software

Im Kapitel Software wird genauer beschrieben, wie die Programmierung des Vier-Gewinnt-Roboters umgesetzt ist. Die Software bildet das Kernstück des Roboters und ist entscheidend dafür, dass dieser eigenständig am Spiel teilnehmen kann.

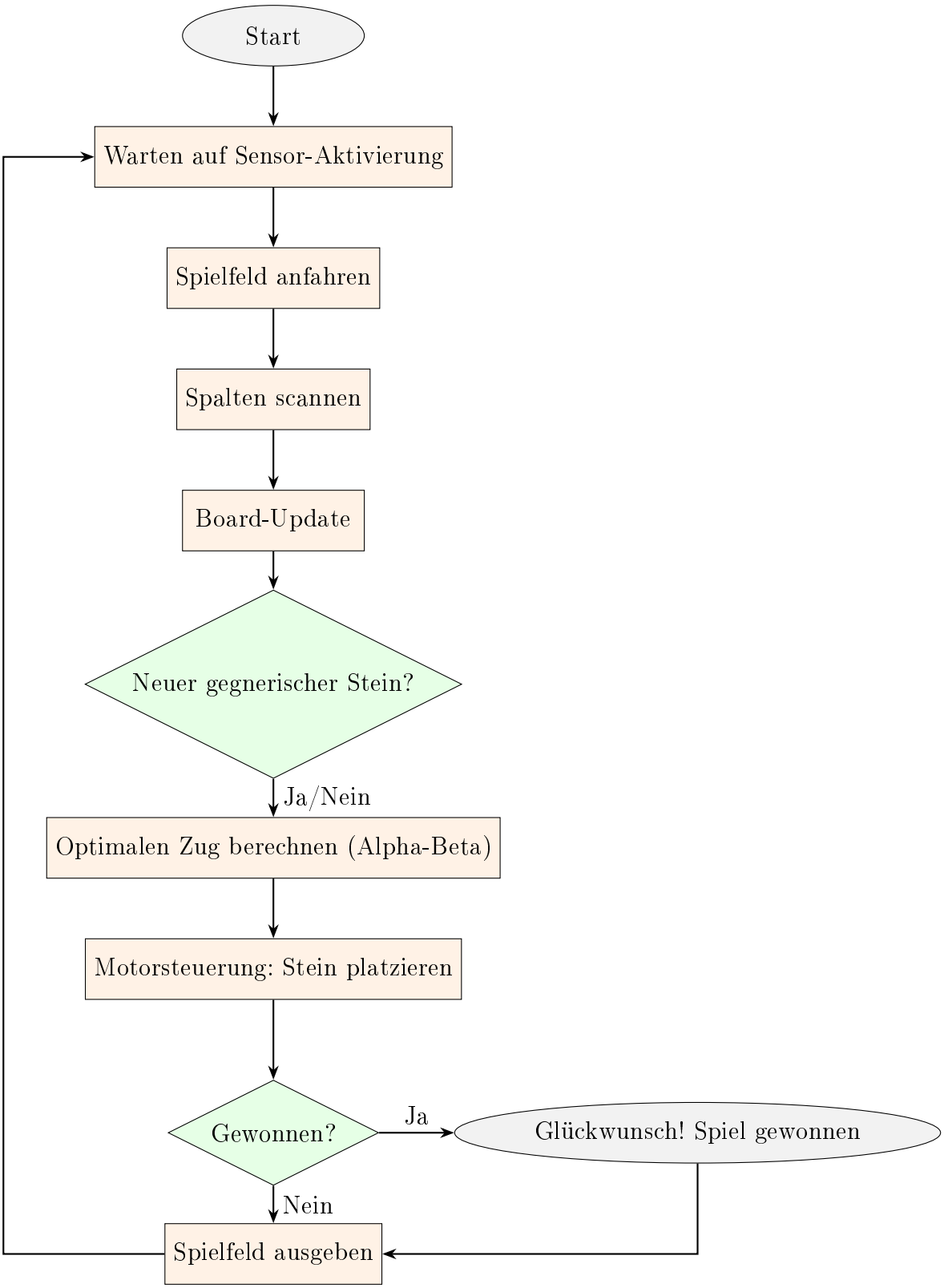


**Abbildung 4.3:** Flussdiagramm der Software

## 4.3 Algorithmus zur Entscheidungsfindung

Ein zentraler Bestandteil des 4-Gewinnt-Roboters ist die Entscheidungsfindung durch einen algorithmischen Spielbaum. Dieser wird mit dem bekannten
Minimax-Algorithmus unter Verwendung von Alpha-Beta-Pruning realisiert. Ziel ist es, basierend auf dem aktuellen Spielfeldzustand den optimalen Zug für die KI zu berechnen. Der Algorithmus bewertet mögliche Züge bis zu einer bestimmten Tiefe im Spielbaum und trifft Entscheidungen, die langfristig zum Sieg führen können oder gegnerische Gewinnzüge verhindern.

### 4.3.1 Spielfeld und Spielerdefinition

Im Vorfeld des Algorithmus ist festgelegt, welches Symbol der Algorithmus (die KI) spielt:

```
1     my_piece = 1  # 1 = YELLOW (AI player), -1 = RED
2     opponent_piece = -my_piece
```

Dabei entspricht 1 dem gelben Spielstein (der KI), -1 dem roten Spielstein (dem Gegner). Diese numerische Darstellung vereinfacht die Bewertung und das Vergleichen der Felder im Spielfeld.

#### Bewertungsfunktion (evaluate)

Der Algorithmus benötigt eine Bewertungsfunktion, die die Qualität eines Spielzustands abschätzt. Dies geschieht durch eine Heuristik, die mögliche Gewinnlinien zählt und bewertet.

Die Bewertungsfunktion basiert auf der Idee, sogenannte „Fenster“ (Ausschnitte aus

4 Feldern) im Spiel zu analysieren und zu beurteilen, wie viele Steine der KI bzw. des Gegners in diesen Fenstern enthalten sind:

```
    def evaluate_window(window, player):
    opp_player = opponent_piece if player == my_piece else
   my_piece
    score = 0
    if window.count(player) == 4:
    score += 100
    elif window.count(player) == 3 and window.count(0) == 1:
    score += 5
    elif window.count(player) == 2 and window.count(0) == 2:
    score += 2
    if window.count(opp_player) == 3 and window.count(0) == 1:
    score -= 4
    return score
```

Diese Funktion bewertet sowohl offensive als auch defensive Situationen. Ein Fenster mit drei eigenen Steinen und einem leeren Feld wird positiv bewertet, ein Fenster mit drei gegnerischen Steinen und einem leeren Feld hingegen negativ, um Bedrohungen abzuwehren.

Die Hauptfunktion zur Bewertung des gesamten Spielfeldes aggregiert alle horizontalen, vertikalen und diagonalen Fenster:

```
    def evaluate(board):
    score = 0
    center_array = [board[r][field_width//2] for r in range(
   field_height)]
    center_count = center_array.count(my_piece)
    score += center_count * 3
```

Zunächst werden die mittleren Spalten stärker gewichtet, da sie strategisch wichtiger

sind (von dort aus können mehr Gewinnlinien entstehen).

Anschließend werden alle Zeilen, Spalten und Diagonalen analysiert:

```
    for r in range(field_height):
    row_array = [board[r][c] for c in range(field_width)]
    for c in range(field_width - 3):
    window = row_array[c:c+4]
    score += evaluate_window(window, my_piece)
    score -= evaluate_window(window, opponent_piece)
```

Diese Schleifen bilden das heuristische Fundament für die spätere Entscheidungsfindung.

### 4.3.2 Minimax mit Alpha-Beta-Pruning

Die Hauptentscheidung trifft der Minimax-Algorithmus. Dabei wird rekursiv der Spielbaum aufgebaut, wobei sich der Algorithmus abwechselnd in die Rolle der KI („maximizing player“) und des Gegners („minimizing player“) versetzt. Um die Effizienz zu steigern, wird Alpha-Beta-Pruning genutzt. Dabei werden Äste im Spielbaum verworfen, wenn sie nachweislich zu schlechteren Ergebnissen führen.

Der Einstiegspunkt ist:

```
    def alpha_beta(board, depth, alpha, beta, maximizing_player)
   :
```

Zuerst wird geprüft, ob der aktuelle Zustand bereits im Transposition Table gespeichert ist – einem Cache zur Vermeidung redundanter Berechnungen:

```
    key = (board_hash(board, maximizing_player), depth)
    if key in transposition_table:
    return transposition_table[key]
```

Anschließend erfolgt eine Terminalprüfung: Ist der Zug eine Gewinnsituation, oder wurde die maximale Tiefe erreicht?

```
    valid_locations = [col for col in range(field_width) if
is_valid_location(board, col)]
    terminal = winning_move(board, my_piece) or winning_move(
board, opponent_piece) or len(valid_locations) == 0
    if depth == 0 or terminal:
    ...
```

Falls ja, gibt die Funktion eine Bewertung zurück. Andernfalls wird der Spielbaum weiter durchlaufen.

### 4.3.3 Maximierender Spieler (KI):

```
    if maximizing_player:
    value = -float('inf')
    for col in valid_locations:
    ...
    new_score = alpha_beta(..., False)[1]
    if new_score > value:
    value = new_score
    best_col = col
    alpha = max(alpha, value)
    if alpha >= beta:
    break
```

Hier versucht der Algorithmus, die maximal erreichbare Bewertung zu finden und prüft regelmäßig, ob das aktuelle Ergebnis besser ist als die bisherige beste Option. Wenn `alpha >= beta`, wird der restliche Baum abgeschnitten (Pruning).

**Minimierender Spieler (Gegner):** Analog erfolgt das Vorgehen für den Gegner:

```
    else:
    value = float('inf')
    for col in valid_locations:
    ...
    new_score = alpha_beta(..., True)[1]
    if new_score < value:
    value = new_score
    best_col = col
    beta = min(beta, value)
    if beta <= alpha:
    break
```

Am Ende wird das Ergebnis in der Transpositionstabelle gespeichert und zurückgegeben:

```
    transposition_table[key] = result
    return result
```

### 4.3.4 Dynamische Suchtiefe

Je nach Spielphase kann es sinnvoll sein, tiefer oder flacher zu suchen. Zu Beginn reicht eine niedrige Tiefe, da viele Züge möglich sind. In späteren Phasen erhöht sich die Tiefe:

```
    def get_dynamic_depth(board):
    empty = sum(row.count(0) for row in board)
    if empty > 30:
    return 3
    else:
    return 4
```

Diese dynamische Anpassung balanciert Spielstärke und Rechenzeit optimal.

### 4.3.5 Fazit

Der eingesetzte Minimax-Algorithmus mit Alpha-Beta-Pruning stellt das strategische Herzstück des 4-Gewinnt-Roboters dar. Durch gezielte Bewertung von Spielpositionen, Berücksichtigung gegnerischer Drohungen und dynamische Tiefenanpassung kann der Roboter selbstständig Züge planen, Gefahren abwehren und letztlich siegreich agieren. Die Verwendung eines Transpositionstables beschleunigt dabei die Entscheidungsfindung, indem bereits analysierte Spielsituationen nicht erneut bewertet werden müssen. Das Ergebnis ist ein hochgradig effektives Entscheidungsverfahren für ein strategisches Spiel wie 4-Gewinnt.

### 4.3.6 Begrenzung

Der LEGO Spike Hub besitzt mit seinem 100MHz ARM Cortex-M4 Prozessor, 320 KB RAM und 1 MB Flash-Speicher eine stark begrenzte Hardwareausstattung. Diese Ressourcen reichen für einfache Steuerungsaufgaben, setzen aber dem Einsatz komplexer Algorithmen wie Minimax enge Grenzen.

Insbesondere der geringe Arbeitsspeicher ist entscheidend: Bereits bei einer Suchtiefe von 4 erreicht der Algorithmus die Speichergrenze, da jeder Spielzug rekursiv bewertet

und gespeichert wird. Eine tiefere Suche führt zu Speicherüberläufen oder langen Berechnungszeiten.

Durch Alpha-Beta-Pruning und eine dynamisch begrenzte Suchtiefe lässt sich der Algorithmus dennoch effizient auf dem Hub einsetzen – mit akzeptabler Reaktionszeit und stabiler Ausführung.

# 5 Zusammenfassung

# Literaturverzeichnis


[Ado09] Julius Adorf. *Adversariale Suche für optimales Spiel: Der Minimax-Algorithmus und die Alpha-Beta-Suche*. Proseminararbeit. Betreuer: Lars Kunze, Dominik Jain. Abgabetermin: 2. Dezember 2009. München: Technische Universität München, Fakultät für Informatik, Forschungs- und Lehreinheit Informatik IX, 2009. URL: https://www.juliusadorf.com/pub/alphabeta-seminar-paper.pdf.

[Bel24] Charles A. Bell. *MicroPython for the Internet of Things: A Beginner's Guide to Programming with Python on Microcontrollers*. English. 2nd. Berkeley, CA: Apress, 2024. ISBN: 9781484298619.

[Bet25] Betzold GmbH. *LEGO Education SPIKE Technic Farbsensor*. Zugriff am 04.07.2025. 2025. URL: https://www.betzold.de/prod/E_761144/.

[Has20] SA Hasbro. *Das Originale 4Gewinnt Anleitung*. Hasbro Gaming, 2020.

[LEG20a] LEGO Education. *Technical Specifications: Technic Color Sensor*. https://assets.education.lego.com/v3/assets/blt293eea581807678a/blt62a78c227edef070/5f8801b9a302dc0d859a732b/techspecs_technic colorsensor.pdf?locale=en-us. Accessed: 2025-07-04. 2020.

[LEG20b] LEGO Education. *Technical Specifications: Technic Large Hub*. https://assets.education.lego.com/v3/assets/blt293eea581807678a/bltf512a371e82f6420/5f8801baf4f4cf0fa39d2feb/techspecs_technic largehub.pdf?locale=en-us. Zugriff am 03.07.2025. 2020.

[LEG20c] LEGO Education Community. *Exploring SPIKE™ Prime Sensors*. Accessed: 2025-07-04. 2020. URL: https://community.legoeducation.com/blogs/31/220.

[PLJ23] Ignas Plauska, Agnius Liutkevičius und Audronė Janavičiūtė. „Performance Evaluation of C/C++, MicroPython, Rust and TinyGo Programming Languages on ESP32 Microcontroller“. In: *Electronics* 12.1 (2023). Open Access Article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license, S. 143. DOI: 10.3390/electronics12010143. URL: https://www.mdpi.com/2079-9292/12/1/143.

[SS23] Albin Samefors und Felix Sundman. *Investiating Energy Consumption and Responsiveness of low power modes in MicroPython for STM32WB55*. Bachelor's thesis. 15 hp (first-cycle education). Jönköping, Sweden, Juni 2023. URL: https://www.diva-portal.org/smash/get/diva2:1778426/FULLTEXT01.pdf.

# Abbildungsverzeichnis

# Tabellenverzeichnis

# A Komplettes Python-Programm

```
1  import motor
2  import color_sensor
3  import color
4  import time
5  import force_sensor
6  from hub import port, sound
7
8  # ── CONFIGURATION ──
9  my_piece = 1 # -1 = RED, 1 = YELLOW (AI player)
10 opponent_piece = -my_piece
11
12 speed_D = 230
13 speed_A = 500
14 speed_E = 600
15 move_distance_e = 97 # Step motor E per row, calibrate if necessary!
16 move_distance_d = 73
17 break_motor_e = 1
18 break_motor_e_back = 2
19 break_motor_d = 1
20 field_width = 7
21 field_height = 6
22 waiting_line = 1 # Waiting time after reaching the line (in seconds)
23
24 # ── HELPER FUNCTIONS ──
25 def sensor_activated():
26     try:
27         return force_sensor.force(port.C) > 30
28     except:
29         return False
30
```

```
31  def update_board(row, col, detected_color):
32    if detected_color in (color.RED, color.PURPLE, color.MAGENTA):
33      board[row][col] = -1
34    elif detected_color in (color.YELLOW, color.WHITE, color.GREEN):
35      board[row][col] = 1
36    else:
37      board[row][col] = 0
38
39  def print_board(board):
40    symbol_map = {-1: "R", 1: "Y", 0: "B", None: "B"}
41    print("\nCurrent board (bottom right = [0][0]):")
42    for row_idx in reversed(range(field_height)):
43      row = board[row_idx]
44      print(" ".join(symbol_map.get(cell, "?") for cell in row))
45
46  def is_valid_location(board, col):
47    return board[field_height-1][col] == 0
48
49  def get_next_open_row(board, col):
50    for r in range(field_height):
51      if board[r][col] == 0:
52      return r
53    return None
54
55  def winning_move(board, piece):
56    for c in range(field_width-3):
57      for r in range(field_height):
58        if board[r][c] == piece and board[r][c+1] == piece and board[r][
    c+2] == piece and board[r][c+3] == piece:
59          return True
60    for c in range(field_width):
61      for r in range(field_height-3):
62        if board[r][c] == piece and board[r+1][c] == piece and board[r
    +2][c] == piece and board[r+3][c] == piece:
63          return True
64    for c in range(field_width-3):
65      for r in range(field_height-3):
66        if board[r][c] == piece and board[r+1][c+1] == piece and board[r
    +2][c+2] == piece and board[r+3][c+3] == piece:
67          return True
```

```
68      for r in range(3, field_height):
69          if board[r][c] == piece and board[r-1][c+1] == piece and board[r
        -2][c+2] == piece and board[r-3][c+3] == piece:
70              return True
71      return False
72
73  def evaluate_window(window, player):
74      opp_player = opponent_piece if player == my_piece else my_piece
75      score = 0
76      if window.count(player) == 4:
77          score += 100
78      elif window.count(player) == 3 and window.count(0) == 1:
79          score += 5
80      elif window.count(player) == 2 and window.count(0) == 2:
81          score += 2
82      if window.count(opp_player) == 3 and window.count(0) == 1:
83          score -= 4
84      return score
85
86  def evaluate(board):
87      score = 0
88      center_array = [board[r][field_width//2] for r in range(field_height
        )]
89      center_count = center_array.count(my_piece)
90      score += center_count * 3
91      for r in range(field_height):
92          row_array = [board[r][c] for c in range(field_width)]
93          for c in range(field_width - 3):
94              window = row_array[c:c+4]
95              score += evaluate_window(window, my_piece)
96              score -= evaluate_window(window, opponent_piece)
97      for c in range(field_width):
98          col_array = [board[r][c] for r in range(field_height)]
99          for r in range(field_height - 3):
100             window = col_array[r:r+4]
101             score += evaluate_window(window, my_piece)
102             score -= evaluate_window(window, opponent_piece)
103     for r in range(field_height - 3):
104         for c in range(field_width - 3):
105             window = [board[r+i][c+i] for i in range(4)]
```

```
106         score += evaluate_window(window, my_piece)
107         score -= evaluate_window(window, opponent_piece)
108     for r in range(3, field_height):
109       for c in range(field_width - 3):
110         window = [board[r-i][c+i] for i in range(4)]
111         score += evaluate_window(window, my_piece)
112         score -= evaluate_window(window, opponent_piece)
113     return score
114
115   transposition_table = {}
116
117   def board_hash(board, maximizing_player):
118     return (tuple([item for row in board for item in row]),
      maximizing_player)
119
120   def get_dynamic_depth(board):
121     # Count empty fields
122     empty = sum(row.count(0) for row in board)
123     if empty > 30:
124       return 3 # Beginning: fast, low depth
125     else:
126       return 4 # End: high depth for best moves
127
128   def alpha_beta(board, depth, alpha, beta, maximizing_player):
129     key = (board_hash(board, maximizing_player), depth)
130     if key in transposition_table:
131       return transposition_table[key]
132     valid_locations = [col for col in range(field_width) if
      is_valid_location(board, col)]
133     valid_locations.sort(key=lambda c: abs(c - field_width // 2))
134     terminal = winning_move(board, my_piece) or winning_move(board,
      opponent_piece) or len(valid_locations) == 0
135     if depth == 0 or terminal:
136       if terminal:
137         if winning_move(board, my_piece): return (None, 1000000)
138         elif winning_move(board, opponent_piece): return (None,
      -1000000)
139         else: return (None, 0)
140       else:
141         return (None, evaluate(board))
```

```
142     if maximizing_player:
143       value = -float('inf')
144       best_col = valid_locations[0]
145       for col in valid_locations:
146         row = get_next_open_row(board, col)
147         if row is None:
148           continue
149         board_copy = [r[:] for r in board]
150         board_copy[row][col] = my_piece
151         new_score = alpha_beta(board_copy, depth-1, alpha, beta, False)
    [1]
152         if new_score > value:
153           value = new_score
154           best_col = col
155         alpha = max(alpha, value)
156         if alpha >= beta:
157           break
158       result = (best_col, value)
159     else:
160       value = float('inf')
161       best_col = valid_locations[0]
162       for col in valid_locations:
163         row = get_next_open_row(board, col)
164         if row is None:
165           continue
166         board_copy = [r[:] for r in board]
167         board_copy[row][col] = opponent_piece
168         new_score = alpha_beta(board_copy, depth-1, alpha, beta, True)
    [1]
169         if new_score < value:
170           value = new_score
171           best_col = col
172         beta = min(beta, value)
173         if beta <= alpha:
174           break
175       result = (best_col, value)
176     transposition_table[key] = result
177     return result
178
179
```

```
180 # ---- MAIN PROGRAM ----
181 board = [[0 for _ in range(field_width)] for _ in range(field_height)]
182 last_board = [row[:] for row in board]
183
184 def move_motor_e_to_zero():
185   pass
186
187 while True:
188   print("Waiting for sensor at port C...")
189   while not sensor_activated():
190     time.sleep(0.1)
191   print("Sensor detected! Starting move...")
192   time.sleep(1)
193
194   # Move to field
195   motor.run_for_degrees(port.D, 198, 170)
196   print("Field reached...")
197   time.sleep(1.5)
198
199   transposition_table.clear()
200
201   opponent_piece_found = False
202   opponent_col = None
203   move_motor_e_to_zero()
204
205   for col in range(field_width - 1, -1, -1):
206     matrix_col = field_width - 1 - col
207     if last_board[field_height-1][matrix_col] != 0:
208       if col > 0:
209         motor.run_for_degrees(port.D, move_distance_d, 170)
210         time.sleep(break_motor_d)
211       continue
212
213     free_row = None
214     for row in range(field_height):
215       if last_board[row][matrix_col] == 0:
216         free_row = row
217         break
218     if free_row is None:
219       continue
```

```
220
221     motor.run_for_degrees(port.E, move_distance_e * (free_row), speed_E)
222     time.sleep(break_motor_e)
223     time.sleep(waiting_line)
224
225     detected_color = color_sensor.color(port.B)
226     update_board(free_row, matrix_col, detected_color)
227     print("Matrix entry: Row {}, Col {}: {}".format(
228       free_row, matrix_col,
229       "RED" if board[free_row][matrix_col] == -1
230       else "YELLOW" if board[free_row][matrix_col] == 1
231       else "NONE"))
232
233     if (last_board[free_row][matrix_col] == 0 and
234       board[free_row][matrix_col] == opponent_piece):
235       print("New opponent piece in column {}, row {}".format(matrix_col,
      free_row))
236       opponent_piece_found = True
237       opponent_col = col
238       time.sleep(1)
239
240     motor.run_for_degrees(port.E, -move_distance_e * free_row, speed_E)
241     time.sleep(break_motor_e_back)
242
243     if opponent_piece_found:
244       break
245
246     if col > 0:
247       motor.run_for_degrees(port.D, move_distance_d, speed_D)
248       time.sleep(break_motor_d)
249
250     if opponent_piece_found and opponent_col is not None and
    opponent_col != 0:
251       steps_right = opponent_col
252       motor.run_for_degrees(port.D, move_distance_d * steps_right,
    speed_D)
253       time.sleep(4)
254
255     board_numeric = [[0 if x is None else x for x in row] for row in
    board]
```

```
256     dynamic_depth = get_dynamic_depth(board_numeric)
257     best_col, _ = alpha_beta(
258       board_numeric,
259       depth=dynamic_depth,
260       alpha=-float('inf'),
261       beta=float('inf'),
262       maximizing_player=(my_piece == 1)
263     )
264
265     best_row = get_next_open_row(board_numeric, best_col)
266     color_str = "RED" if my_piece == -1 else "YELLOW"
267     print("\nOptimal position for {}:".format(color_str))
268     print("Column: {}, Row: {}".format(best_col, best_row))
269
270     if best_col is not None and best_row is not None:
271       physical_target_col = field_width - 1 - best_col
272       motor.run_for_degrees(port.D, -move_distance_d *
    physical_target_col, speed_D)
273       time.sleep(break_motor_d * 4)
274       motor.run_for_degrees(port.A, -360, speed_A)
275       time.sleep(3)
276       board[best_row][best_col] = my_piece
277       last_board = [row[:] for row in board]
278
279       if winning_move(board, my_piece):
280         print(" Congratulations! The robot has WON the game!")
281         print_board(board)
282         sound.beep(440, 1000000, 100)
283         motor.run_for_degrees(port.D, -move_distance_d * best_col,
    speed_D)
284         time.sleep_ms(2500)
285         motor.run_for_degrees(port.D, -move_distance_d * 3, speed_D)
286         sound.beep(0, 1000000, 100)
287         break
288
289       motor.run_for_degrees(port.D, -move_distance_d * best_col, speed_D
    )
290
291     print_board(board)
292     time.sleep_ms(2500)
```

```
293     motor.run_for_degrees(port.D, -199, speed_D)
294
295     time.sleep(break_motor_d)
296
297     while sensor_activated():
298       time.sleep(0.1)
299     print("Move completed. Waiting for next activation...")
```

# B Nutzung von Künstliche Intelligenz basierten Werkzeugen

Im Rahmen dieser Arbeit wurden Künstliche Intelligenz (KI) basierte Werkzeuge benutzt. Tabelle 2.1 gibt eine Übersicht über die verwendeten Werkzeuge und den jeweiligen Einsatzzweck.

**Tabelle 2.1:** Liste der verwendeten KI basierten Werkzeuge

| **Werkzeug** | **Beschreibung der Nutzung** |
|---|---|
| ChatGPT | • Grundlagenrecherche zu Spieltheorie |
| Perplexity | • Grundlagenrecherche zu Spieltheorie<br>• Formulierungshilfe |
| DeepL | • Unterstützung beim Übersetzung von Abstract |
| Languagetool | • Formulierungshilfe<br>• Rechtschreibkorrektur |