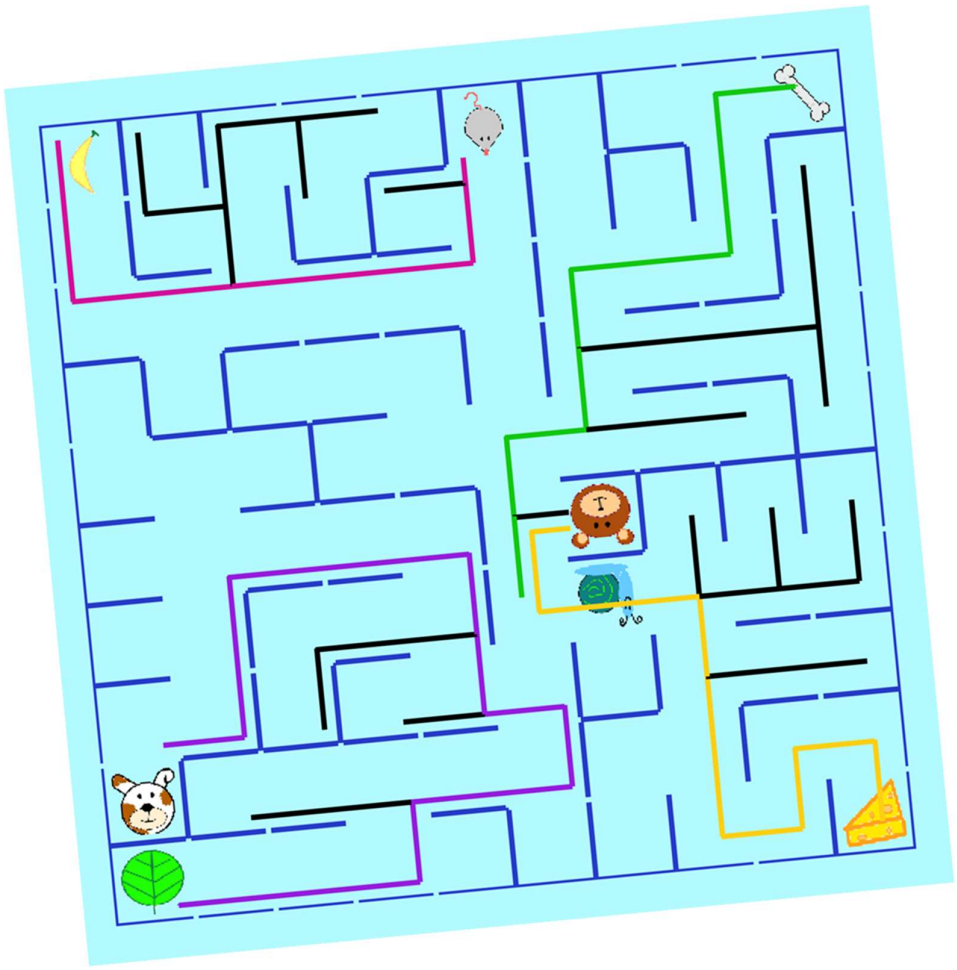
A_maze with a Robot Swarm



Ein Projekt von Aarav und Viyona Singh – Gymnasium Gröbenzell

Kurzfassung

Irrgärten enthalten viele herausfordernde Konzepte, sowohl aus dem mathematischen als auch aus dem computerwissenschaftlichen Bereich. Durch ihren komplexen Bau sind Irrgärten schwer zu lösen. Bei der Umsetzung von Irrgärtengenerier- und Lösungsalgorithmen spielt Schwarmintelligenz eine entscheidende Rolle. Roboter sind darauf angewiesen, kooperativ zu agieren, um gemeinsam komplexe Aufgabenstellungen zu bewältigen. Diese kollektive Herangehensweise ermöglicht es den Robotern nicht nur, Zeit zu sparen, sondern auch ihre Batteriekapazität zu schonen, die sie bei spezifischen Aktionen verbrauchen. Sie können ihre Energievorräte wieder aufladen, indem sie eine beliebig platzierte Ladestation im Irrgarten aufspüren.

Die erstaunlichen Resultate zahlreicher durchgeführter Experimente wurden mathematisch nachgewiesen und systematisch dokumentiert. Darüber hinaus wurde ein Modell auf Basis von Reinforcement-Learning entwickelt, das auf die Optimierung von Zeit und Batterieverbrauch abzielt.

Inhaltsverzeichnis

1 Einleitung

Was versteht man unter einem Irrgarten? Ein Irrgarten ist ein Wegesystem, welches durch vielen Richtungsänderungen, Verzweigungen, Kreuzungen, Schleifen und Sackgassen das Gelangen von einem zum anderen Punkt zu einem Rätsel macht. Dieser kann verschiedene Formen enthalten. Ein Irrgarten ohne Schleifen heißt Standard-Irrgarten oder auch perfekter Irrgarten und hat nur einen möglichen Lösungsweg.

Die Graphentheorie spielt in der Mathematik eine wichtige Rolle. Auch Irrgärten können in in Graphen umgewandelt werden (Abb. 1). Dabei entsprechen die Diagrammknoten den Kästchen des Irrgartens und die Diagrammkanten ihren Verbindungen. Möchte man perfekte Irrgärten in einen Graphen umwandeln, erhält man einen Baumdiagramm. Das liegt daran, dass es keine Schleifen im Irrgarten und somit auch keine im Graphen gibt.

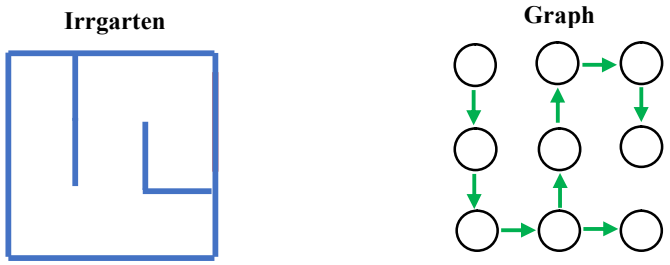


Abbildung 1: Ein Irrgarten in einen Graphen umgewandelt

Beim Lösen von Irrgärten folgt man in der Regel vom Ausgangspunkt aus verschiedenen Wegen, bis man das Ziel erreicht. Das funktioniert jedoch nur, wenn man den Überblick über den gesamten Irrgarten hat. Wenn man sich jedoch *im* Irrgarten befindet und nur begrenzte Energie hat, um den Endpunkt zu erreichen, wird das Lösen des Irrgartens schwieriger fallen. So entstand die Idee, verschiedene Methoden zu erforschen und zu programmieren, um eine Lösung für dieses Problem zu finden. Es gibt verschiedene Algorithmen zum Lösen eines Irrgartens, die für unterschiedliche Situationen dienen. Beispielsweise gibt es welche für das Lösen perfekter Irrgärten und welche für das Lösen nicht perfekter Irrgärten.

Neben den zahlreichen Lösungsalgorithmen sind auch Schwarmintelligenz und Reinforcement Learning Teil der Problemlösung, da diese es den Lösern des Irrgartens ermöglicht, sich gegenseitig zu helfen, um den Zielpunkt zu erreichen und sich dabei mit Hilfe der künstlichen Intelligenz regelmäßig zu verbessern. Mit Irrgärten lohnt es sich, viele Experimente durchzuführen, da man dabei erstaunliche Beobachtungen machen kann. Diese kann man schließlich mit einem mathematischen Ansatz nachweisen.

Das Ziel des Projektes liegt darin, die Mathematik genauer zu verstehen und Methoden anzuwenden, um bestimmte Situationen in unbekannten Irrgärten zu erleichtern.

Die Langfassung dieses Projekts ist wie folgt aufgebaut: Das zweite Kapitel gibt einen ersten Einblick in die verwendeten Algorithmen und Konzepte. Kapitel drei geht näher auf die Programmierung ein. Im vierten Kapitel werden Experimente untersucht, worauf in Kapitel fünf eine Strategie mit Reinforcement Learning folgt.

2 Hintergrund

2.1 Irrgärten beliebig generieren

Um Irrgärten beliebig generieren zu können, kann man verschiedene Algorithmen wie zum Beispiel von Prim [1] und Kruskal [2] verwenden. Diese beiden Algorithmen dienen nur für das Generieren perfekter Irrgärten. Um Irrgärten mit Schleifen zu erstellen, kann man von einem perfekten Irrgarten Versperrungen löschen, wodurch sich Schleifen bilden.

2.2 Irrgärten lösen

Um einen beliebig generierten Irrgarten zu lösen [3] bzw. von einem zum anderen Kästchen zu gelangen, ist es wichtig zu unterscheiden, ob man den ganzen Irrgarten im Überblick hat oder darin gefangen ist ohne diesen zu kennen. Algorithmen wie der Kürzeste-Weg-Algorithmus setzen voraus, dass der Irrgarten als Ganzes überblickt wird. In diesem Fall sollte man dazu fähig sein, den kleinstmöglichen Weg vom Startpunkt bis zum Zielpunkt zu erkennen. Dagegen ist die Linke- bzw. Rechte-Hand-Methode und der Floodfill Algorithmus dafür geeignet, wenn man den Irrgarten nicht kennt.

2.3 Schwarmintelligenz

Die Entwicklung der Technologie nimmt stark zu. Zukünftig werden Roboter im Haushalt helfen, Fragen beantworten, sich mit uns unterhalten und vieles mehr. Dafür müssen sie miteinander komunizieren können. Roboter müssen sich gegenseitig helfen können, um die Lösungen komplexer Aufgaben zu ermitteln. Dies wird als *Schwarmintelligenz* bezeichnet.

Dahinter steckt folgendes Prinzip: Die Masse weiß es besser, spürt die Wahrheit schneller und findet die klügeren Lösungen.

Ein Beispiel für die Schwarmintelligenz ist folgendes:

Am 21. Mai 1968 verlor die US Navy den Kontakt zu einem ihrer Atom-U-Boote im Nordatlantik, der USS Scorpion und hatte einfach zu wenige Informationen über die Unglücksursache. Sie bildeten ein Team aus Technikern, Offizieren, Navigatoren, die allesamt überlegten, was passiert sein könnte: Wie schnell, in welche Richtung und wie tief ist das Boot gefahren? All diese Einzelmeinungen fasste man anschließend zu einem Mittelwert zusammen.

2.4 Reinforcement Learning

Reinforcement Learning (abgekürzt RL) steht für eine Methode des maschinellen Lernens und dient für das Besserwerden in einer Aktion (Abb. 2). Hier wird die Vorgehensweise beim Reinforcement Learning beschrieben:

Ein Agent führt eine Aktion durch, welche seine Umwelt beeinflusst. Dadurch bildet sich ein Folgezustand, der wiederum an den Agenten zurückgemeldet wird. Während des Übergangs in den Folgezustand erhält

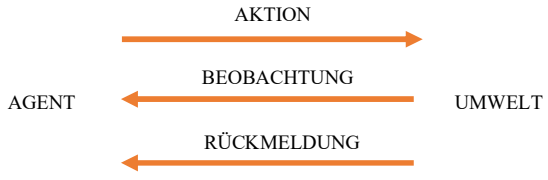


Abbildung 2: Reinforcement Learning

der Agent eine Belohnung bzw. eine Bestrafung. Das Ziel besteht darin, die optimalen Aktionen zu erlernen und die Fehler in Zukunft zu vermeiden.

Zum Reinforcement Learning gehört auch der berühmte Computerprogramm AlphaZero [4], dessen Algorithmus in komplex aufgebauten Brettspiele wie Schach und Go gewinnen kann. Dabei benutzt der Agent den Monte-Carlo-Tree-Search (abgekürzt MCTS), in dem er mithilfe von großen Baumdiagrammen, die aber nicht alle Zustände des Spiels enthalten, den besten Zug berechnen kann.

3 Implementierung

Der gesamte Code steht online verfürgbar [5]. Wie ist der gesammte Code aufgebaut? Für die Übersichtlichkeit wurde dieser in verschiedenen Dateien unterteilt, die in den nächsten Seiten genauer betrachtet werden. Diese werden hier angegeben und beschrieben:

Name	maze_generator.py	utils.py	grid.py	robot.py	ui.py
Beschreibung	Algorithmen für das beliebige Generieren von Irrgärten	häufig gebrauchbare Funktionen	Informationen zum Irrgarten	für die Roboter	die gesamte Benutzeroberfläche

3.1 Implementierung des Prim's Algorithmus

Zunächst wird die Datei „maze_generator.py" betrachtet. Hierbei wurde eine randomisierte Version von dem Prim's Algorithmus in eine Funktion umgesetzt. Da diese aber nur perfekte Irrgärten generiert, wurde eine weitere Funktion geschrieben, welche eine bestimmte Anzahl an Versperrungen eines perfekten Irrgartens löscht. Somit entstehen Schleifen, wodurch der Irrgarten nicht mehr perfekt bleibt. In Abb. 3 ist ein Beispiel für das Generieren eines 3x3 großen Irrgartens zu sehen, das die Vorgehensweise der Funktion verdeutlicht:

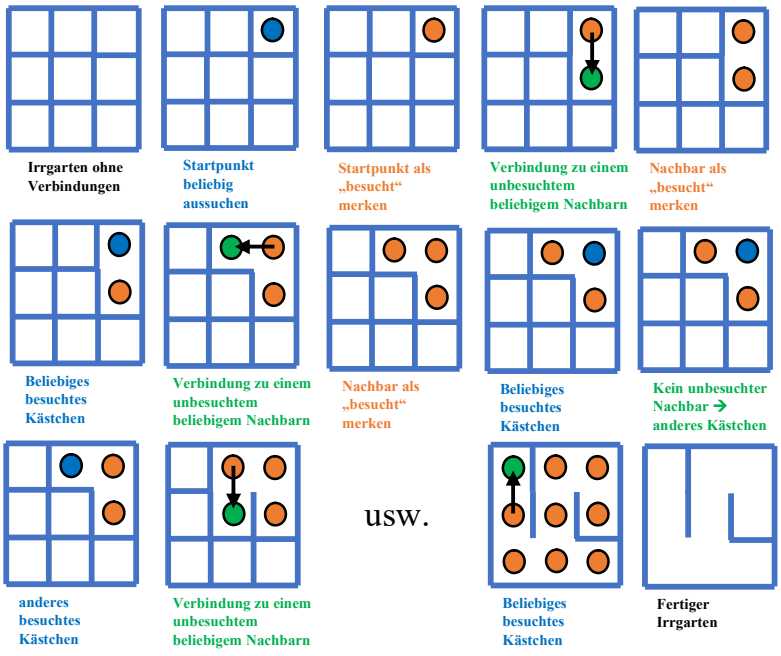


Abbildung 3: Prim's Algorithmus

Hier wird die Funktion in einem Pseudocode beschrieben:

1. Starte mit einem Gitter ohne verbundene Kästchen
2. Wähle ein beliebiges Kästchen als aktuelles Kästchen und markiere es als besucht
3. Wiederhole, solange es unbesuchte Kästchen gibt:
    1. Wähle einen beliebigen Nachbar
    2. Falls dieser noch nicht besucht worden ist:
        1. Markiere den gewählten Nachbar als „besucht“
        2. Lösche die Wand zwischen aktuellen Kästchen und den gewählten Nachbar
    3. Mache gewählten Nachbar zum aktuellen Kästchen

Die Irrgärten werden in Listen gespeichert. Dabei werden die Kästchen des Irrgartens mit Zahlen makiert. Das erste Kästchen ist 0, das zweite (der Reihe nach) 1, usw. Der in Abb. 3 generierter Irrgarten würde als Liste so ausschauen:

[[3], [2, 4], [1, 5], [0, 6], [1, 7], [2], [3, 7], [4, 6, 8], [7]]

Nachbarn von Kästchen 0 — Nachbarn von Kästchen 1 — ...

3.2 Ladestationen

Außer den Robotern und deren Zielpunkten werden im Irrgarten auch Ladestationen anzeigt. Doch wofür sind diese da? Jeder Roboter hat anfangs ein gewisses Batterievolumen, das er verbraucht, sobald er seine Position im Irrgarten ändert. Richtungswechsel kostet ihn keine Batterie. Um seine Batterie wieder voll aufzuladen, muss der Roboter zu einer Ladestation, die sich im Irrgarten befindet. Die Positionen der Ladestationen sind den Robotern jedoch nicht bekannt.

Trifft ein Roboter auf eine Ladestation, merkt er ihre Position, um sie bei einer zukünftigen Batterieschwäche aufsuchen zu können. Falls dies der Fall ist , verwendet der Roboter den Kürzesten-Weg-Algorithmus, um die nächste Ladestation aufzuspüren und hin und wieder zurück zu seinem Ursprung zu fahren.

3.3 Implementierung der Lösungsalgorithmen

Nun wird die Umsetzung der Lösalgorithmen beschrieben, die ein Teil von „robot.py“ ist. Zu denen gehören die Linke-Hand-Methode, die Rechte-Hand-Methode, der Kürzeste-Weg-Algorithmus und der Floodfill-Algorithmus.

Bei der Linken-Hand-Methode legt der Roboter die linke Hand (bzw. bei der Rechten-Hand-Methode die rechte) auf eine Wand und hält beim Durchlaufen ständigen Kontakt mit ihr, bis er ans Ziel gelangt (siehe Abb. 5). Wenn die Irrgärten Inseln (= autarke Zentralstrukturen, die nicht mit der Außenwand verbunden sind) besitzen, kann diese Methode jedoch versagen, da der Roboter in einer Schleife steckenbleiben kann (siehe Abb. 4).

Abbildung 5: Linke-Hand-Methode funktioniert

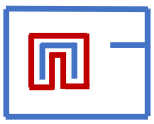


Abbildung 4: Rechte-Hand-Methode versagt

Der Kürzeste-Weg-Algorithmus funktioniert nicht nur bei perfekten Irrgärten, sondern auch bei Irrgärten mit Schleifen. Der Roboter ist dazu fähig, den kürzesten Weg vorauszusehen, da er den Irrgarten kennt. Bei diesem Algorithmus verwendet der Roboter die Breitensuche, welches zu den uniformen Suchalgorithmen dazugehört, um den kürzesten Weg zu berechnen. Dabei beschreitet er zunächst alle Kästchen, die von seiner Ausgangsposition direkt erreichbar sind. Dies wiederholtder Roboter solange, bis einer seiner Nachbarn dem Zielkästchen entspricht. Ist er am Ziel angelangt, verfolgt er den Weg zum Startpunkt zurück. Dieser gilt als den kleinstmöglichen Weg (siehe Abb. 6).

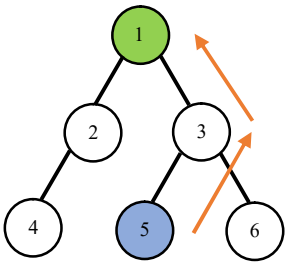


Abbildung 6: Kürzeste-Weg-Algorithmus

Im Floodfill-Algorithmus ignoriert der Roboter zunächst alle Versperrungen und markiert die einzelnen Kästchen mit dem Abstand zum Zielpunkt. Im nebenstehenden Irrgarten (Abb. 7) ist das rechts unten liegende Kästchen der Zielpunkt. So hat dieser den Abstand 0 von sich selbst. Der Startpunkt links oben hat vom Zielpunkt den Abstand 5. Nun bewegt sich der Roboter vom Startpunkt zu dem Nachbar, der den kleinsten Abstand zum Zielpunkt hat. Da in diesem Fall beide Nachbarn den selben Abstand besitzen, ist es unwichtig, wohin der Roboter hinfährt. Hier wird angenommen, dass er nach rechts fährt. Wieder muss der Roboter zu dem Nachbar mit der kleinsten Zahl. Das wiederholt er so lange, bis er das rechts oben liegende Kästchen erreicht. Hier ist der Nachbar mit dem gerringsten Abstand zum Zielpunkt die direkt untere. Der Roboter bemerkt aber, dass er dort nicht hinfahren kann, da sich dazwischen eine Versperrung befindet. So muss er die Abstandzahlen aktualisieren (Abb. 8). Dazu ändert er die Kästchenzahl zu der minimalen Zahl seiner Nachbarn um eins vergrößert. Der Roboter weiß, dass das untere Kästchen nicht sein Nachbar ist. So entspricht die minimale Nachbarzahl eine 3. Nachdem er die aktuelle Kästchenzahl zu einer 3 + 1 = 4 umändert, wiederholt er den oben beschriebenen Schritt, bis er ans Ziel angelangt.

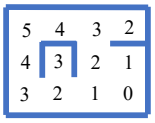


Abbildung 7: Makierung der Kästchen

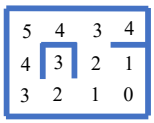


Abbildung 8: nach der Aktualisierung

Wann wird welcher Algorithmus verwendet? Der Roboter ist mit der Linken-, Rechten-Hand-Methode oder dem Floodfill Algorithmus im Irrgarten unterwegs. Der Kürzeste-Weg-Algorithmus wird benutzt, um bei Batterieschwächen zu der nächsten Ladestation fahren zu können, damit der Roboter nicht batterielos wird.

3.4 Benutzeroberfläche


Abbildung 9: Roboter und deren Zielpunkte als Tiere und Lieblingsessen

Die Benutzeroberfläche ist in der Datei „ui.py“ zu finden. Um diese übersichtlich zu gestalten, wurde anstatt Roboter Tiere im Irrgarten abgebildet und deren Zielpunkte als die jeweiligen Lieblingsnahrung gekennzeichnet. Diese werden in Abbildung 9 angezeigt. Ihre zurückgelegenen Strecken wurden mit verschiedenen Farben makiert und die übrigen Batteriemengen werden oberhalb des Irrgartens angezeigt. Zur Gestaltung wurde die Bibliothek „pygame.py“ benutzt, da sich diese für eine ordentliche Formatierung lohnt. Die Benutzeroberfläche kann man jederzeit deaktivieren, um beispielsweise während den Experimenten Zeit zu sparen.

4 Experimente und Beobachtungen

Nun geht es mit den Experimenten weiter. Für alle Experimente wird die Irrgartengröße mit m als Länge und mit n als Breite bezeichnet. Um bei den Experimenten genaue Resultate zu erhalten, wurden diese jeweils insgesamt 500 mal wiederholt.

4.1 Experiment 1: Minimaler, maximaler und durchschnittlicher Batterieverbrauch

Beschreibung: Beim ersten Experiment wurden die minimale, maximale und die durchschnittliche Menge an Batterie untersucht, welche die Roboter beim Gelangen von einer Ecke zum Zielpunkt in unterschiedlich großen Irrgärten mithilfe der Linken-Hand- bzw. der Rechten-Hand-Methode verbrauchen, wobei sich im Irrgarten keine Ladestationen befinden.

Beobachtung: Folgende Tabelle beschreibt die einzelnen Verbräuche an Batterien:

Zielpunkt	**Minimaler Batterieverbrauch**	**Maximaler Batterieverbrauch**	**Durchschnittlicher Batterieverbrauch**
Gegenüberliegende Ecke	m + n – 2	2mn – m – n	etwa mn – 1
Beliebig gewählt	0	2mn – 3	ungelöst

Die beiden Diagramme in Abbildung 10 und 11 zeigen an, wie viele Roboter eine bestimmte Menge an Batterie verbrauchen, wobei der Zielpunkt entweder der gegenüberliegenden Ecke entspricht, oder beliebig ausgewählt wird.

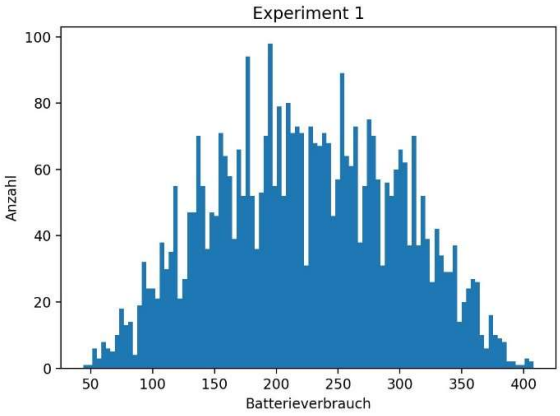


Abbildung 10: Histogramm für gegenüberliegende Ecke als Zielpunkt

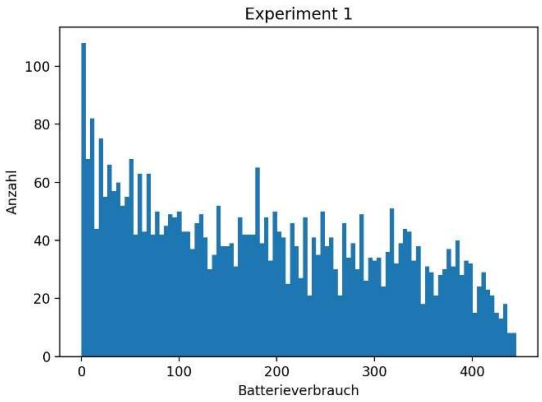


Abbildung 11: Histogramm für beliebiges Zielpunkt

Erklärung: Zuerst wurde der Fall betrachtet, wobei sich das Ziel in der gegenüberliegenden Ecke befindet.

Abbildung 12: Beispiel für minimalen Batterieverbrauch

Minimaler Batterieverbrauch. Bei dem minimalen Batterieverbrauch muss der Roboter bei jedem Schritt seinen Abstand zu dem Zielpunkt verringern. Da er keine diagonale Schritte ausführen darf, wird stets entweder seine horizontale oder seine vertikale Lage sich seinem Ziel annähern (Abb. 12). Eine Option hierfür ist, dass er ununterbrochen Kontakt mit der Außenwand des Irrgartens hält. Um die entsprechende horizontale Lage des Zielpunktes zu erreichen, muss er sich mindestens um n – 1 Schritte bewegen, da die

Breite des Irrgartens n Kästchen beträgt, der Startpunkt aber nicht mitgezählt werden soll. Vertikal muss er sich um m – 1 Schritte bewegen, aus einem ähnlichem Grund, nur ist da nicht die Breite, sondern die Höhe des Irrgartens, also m, entscheidend. Fasst man die Werte zusammen, ergibt dies:

(m – 1) + (n – 1) = m + n – 2 Schritte.

Abbildung 13: Beispiel für maximalen Batterieverbrauch

Maximaler Batterieverbrauch. Der abgebildete Weg im Irrgarten (Abb. 13) ist der längste Weg im Irrgarten, wenn der Roboter von oben-links nach unten-rechts mit der Linken-Hand-Methode fahren will. Dabei wurde der dunkel gefärbte Weg von dem Roboter doppelt gefahren und der hellgefärbte nur einmal. Das liegt daran, dass der dunkle Weg zu „Sackgassen" geführt hat und somit beim Rückkehr zum zweiten Mal besucht wurde. Man kann erkennen, dass jedes Kästchen mindestens einmal besucht wurde. Das ist selbstverständlich, da nach dem maximalen Batterieverbrauch gefragt ist. Der dunklere Weg war für den Roboter überflüssig und wurde doppelt besucht. Der Weg vom Start- zum Endpunkt ohne überflüssiges Abbiegen wurde nur einmal besucht. Damit der einmal besuchte Weg so kurz wie möglich ist, muss der Batterieverbrauch beim Fahren dieser Strecke den minimalen Batterieverbrauch entsprechen. Dieser beträgt m + n – 2. Nimmt man an, der Roboter fährt den gesamten Weg im Irrgarten doppelt, fährt er 2(mn – 1). Nun subtrahiert man den einmal gefahrenen Weg von 2(mn – 1) :

2(mn – 1) – (m + n – 2) = 2mn – 2 – m – n + 2 = 2mn – m – n

Durchschnittlicher Batterieverbrauch. Zunächst wird geschätzt, dass der durchschnittliche Batterieverbrauch das arithmetische Mittel des minimalen und des maximalen Batterieverbrauchs entspricht. Dieses beträgt gerundet etwa:

[m + n – 2 + 2mn – m – n] / 2 = mn – 1

Nun wird der Fall, in dem der Zielpunkt beliebig ausgewählt wird, betrachtet. Dabei entspricht der minimale Batterieverbrauch Null, und zwar dann, wenn der Zielpunkt dem Startkästchen entspricht. Der maximale Batterieverbrauch beträgt 2mn – 3. Dies ist der Fall, wenn der Zielpunkt direkt neben dem Startpunkt ist. Der minimale Weg beträgt dabei 1. So rechnet man: 2mn – 2 – 1 = 2mn – 3.

Zusätzlicher Versuch:

Um den Wert des durchschnittlichen Batterieverbrauchs zu überprüfen, wurde ein zusätzlicher Versuch durchgeführt. Dabei wurde die Batterie der Roboter genau auf mn – 1 gestellt und nach mehreren Durchläufen des Irrgartens geschaut, wie viele Roboter es geschafft haben, ihr Zielpunkt zu erreichen. Diese müsste 50% betragen, da mn – 1 der Durchschnitt ist. Genau dies war der Fall. So wurde bestätigt das der durchschnittliche Batterieverbrauch der Roboter dem arithmetischen Mittel des minimalen und des maximalen Batterieverbrauchs entspricht.

4.2 Experiment 2: Ladestation im Zentrum des Irrgartens mit einer gewissen Batteriemenge und unterschiedlichen Irrgartengrößen

Beschreibung: Als nächstes wird die Hilfe einer Ladestation im Zentrum des Irrgartens betrachtet. Dabei wurde der zusätzliche Versuch des ersten Experiments wiederholt, nur diesmals mit einer Ladestation genau im Zentrum des Irrgartens. Die Roboter müssen zu den gegenüberliegenden Ecken deren Startpunkte mit einer Batteriemenge von mn – 1 gelangen, wobei sich in unterschiedlichgroßen Irrgärten eine Ladestation im Zentrum befindet. Es wurde für n gleich m und für verschiedene Werte von n gleich 10 bis 40 experimentiert. Das Experiment wurde 500 mal mit 4 Robotern wiederholt, also insgesamt mit 2000 Robotern.

Dabei benutzen sie im ersten Durchgang nur die Linke- bzw. Rechte-Hand-Methode für das Erreichen des Zielpunktes, während sie im zweiten Durchgang den Kürzesten-Weg-Algorithmus anwenden, um regelmäßig ihre Batterie aufzuladen. Nun wird der prozentuale Anteil an Roboter, die ihr Ziel erreichen, im ersten und im zweiten Durchgang verglichen.

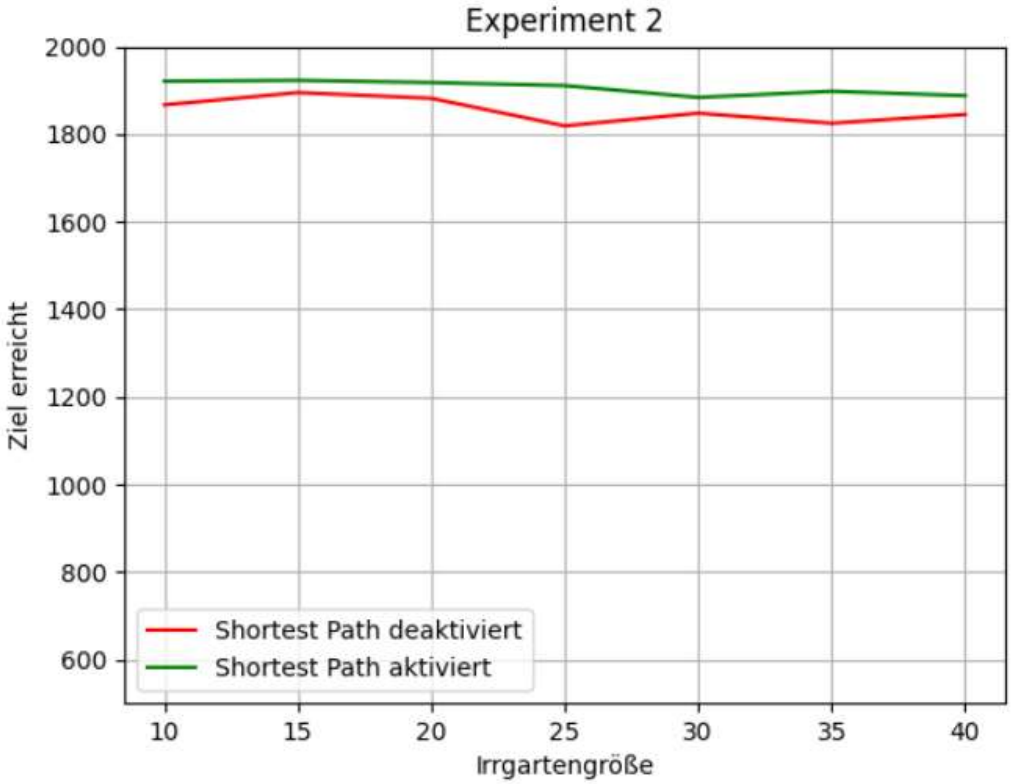


Abbildung 14. Diagramm für Experiment 2

Beobachtung/Erklärung: Als Ergebnis erhält man Abb. 14. Man erkennt, dass im zweiten Durchgang die in Prozent angegebenen Werte größer sind als im ersten Durchgang. Außerdem sind diese bei unteschiedlichen Irrgartengrößen nicht deutlich zu unterscheiden, da auch die Batteriemenge steigert und somit die Anzahl an Robotern, die ihr Ziel erreichen, ausgleicht. Hierbei sind stets etwa 90% der Roboter, also fast alle, am Ziel angelangt, da sie auf ihrem Weg vollaufgeladen wurden.

Bemerkung: Es ist interessant, dass eine Ladestation im Zentrum des Irrgartens nicht allen Robotern hilft. Es schaut so aus, dass der Weg zur Ladestation ¼ des maximalen Batterieverbrauches 2mn – m – n beträgt, also etwa ½ mn und somit alle Roboter den Zielpunkt erreichen. Jedoch stimmt dies nicht, da die Roboter manchmal die Ladestation nicht finden oder sich im übrigen Irrgarten verlieren.

4.3 Experiment 3: Ladestation im Zentrum des Irrgartens mit einer gewissen Irrgartengröße und unterschiedlichen Batteriemengen

Beschreibung: Das zweite Experiment wird wiederholt, nur mit einer steigenden Batteriemenge von m + n – 2 bis 2mn – m – n und einer bestimmten Irrgartengröße von m = 15 und n = 15. Einerseits wird beobachtet, wie sich die Anzahl an Roboter, die ihr Ziel erreichen, sich bei den verschiedenen Batteriemengen verändert. Andererseits werden wieder der erste und der zweite Durchgang miteinander verglichen.

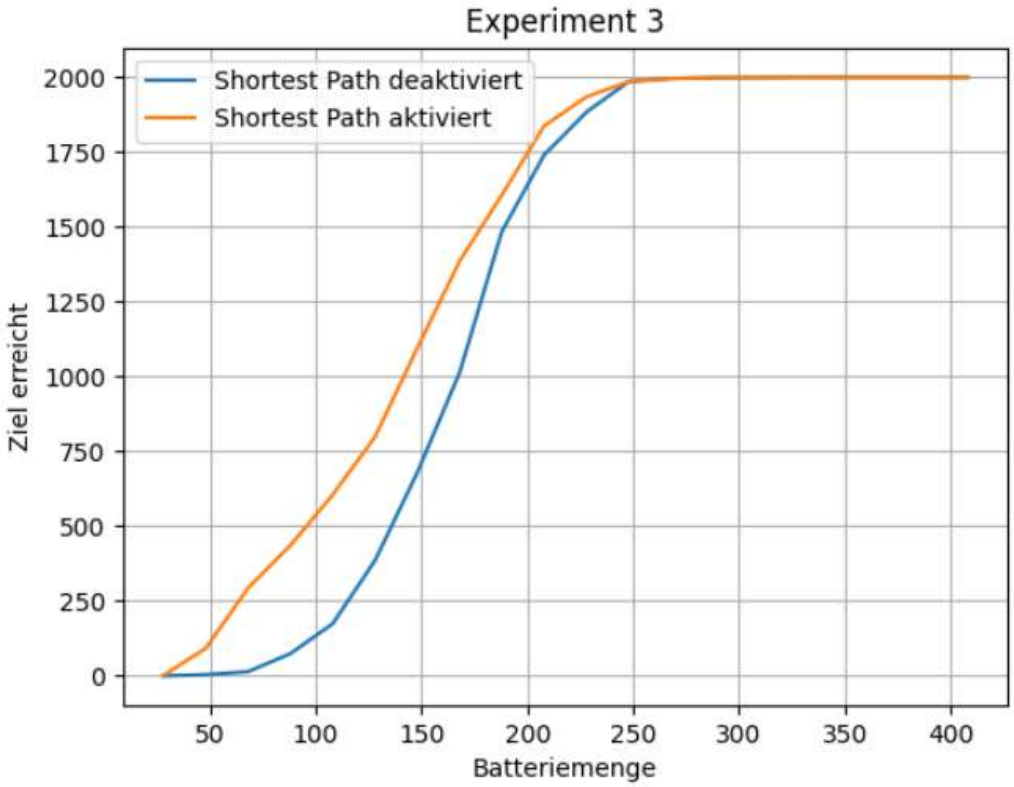


Abbildung 15: Diagramm für Experiment 3

Beobachtung/Erklärung: Abb. 15 zeigt das Ergebnis. Hier steigen die Werte bei beiden Durchgängen ziemlich steil, je größer die Batteriemenge ist. Die Batteriemenge spielt eine große Rolle für den Robotern, da ohne sie das Ziel nicht erreicht werden kann. Der erste Durchgang erzielt beim Versuch höhere Werte als der zweite Durchgang, da dort der Kürzeste-Weg-Algorithmus aktiviert ist.

4.4 Experiment 4: Irrgärten mit der Schwarmintelligenz lösen

Beschreibung: Bis jetzt haben die Roboter keine Informationn geteilt. Genau dies passiert in diesem Experiment. Der Vorteil der Schwarmintelligenz wird betrachtet, indem wieder Experiment 2 durchgeführt wird, jedoch alle Roboter ihre Informationen mit den anderen Roboter teilen. Zu den Information gehören die Ladestationpositionen, sowie alle Kästchenverbindungen der Roboter. Außerdem ist diesmals der Kürzeste-Weg-Algorithmus immer aktiviert.

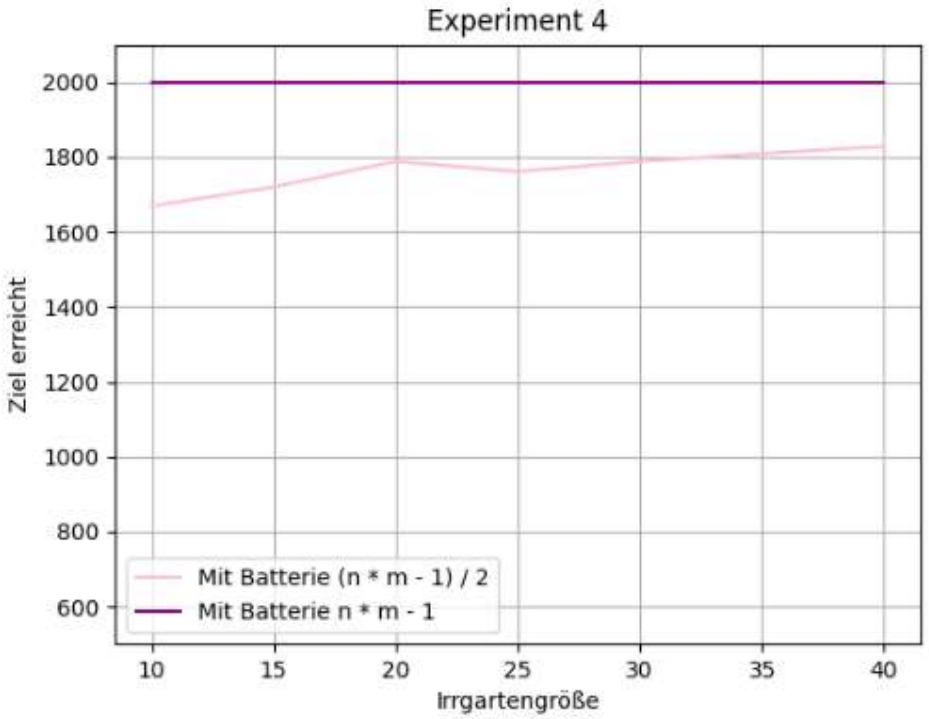


Abbildung 16: Diagramm für Experiment 4

Beobachtung/Erklärung:

Erstaunlicherweise ist es ein großer Vorteil, wenn die Roboter ihre Informationen teilen. Das liegt daran, da nun alle Roboter nicht nur den eigenen gefahrten Weg, sondern auch die der anderen kennen. So können sie ab einem gewissen Punkt Abkürzungen mithilfe des Kürzesten-Weg-Algorithmus vorhersehen und ihr Ziel somit schneller erreichen. Wie man im Diagramm (Abb. 16) sehen kann, erreichen bei durchschnittlicher Batteriemenge (mn – 1) stets alle Robotern ihr Ziel. Um die Steigung zu erkennen, wurde die Batteriemenge in einem weiterem Versuch in Hälfte geteilt ((mn – 1) / 2). Dabei erhielt man die in hellrosa eingezeichnete Linie. Außerdem erreichen die Roboter auch alle das Ziel, wenn es keine Ladestation im Irrgarten gibt.

Bemerkung: In diesem Experiment hat man den Vorteil für das gemeinsame Erforschen des Irrgartens betrachtet. Die Roboter helfen einander, damit sie alle ihr Ziel ohne Ladestationen erreichen können.

4.5 Experiment 5: Schleifen im Irrgarten

Beschreibung: In diesem Experiment werden Irrgärten mit Schleifen untersucht. Es wird gezeigt, dass die Roboter mit der Linken- bzw. der Rechten-Hand-Methode manchmal stecken bleiben. Gleichzeitig wird untersucht, wann dies der Fall ist.

Beobachtung/Erklärung:

Roboter bleiben in Irrgärten mit Schleifen mit der Linken- bzw. der Rechten-Hand-Methode dann stecken, wenn sie eine Insel ständig umkreisen. In Abb. 17 bleiben die Maus und die Schnecke um kleinen Inseln stecken und drehen sich somit in kurzen Schleifen. Halten die Roboter jedoch mit der Außenwand Kontakt, kann dies auch der Fall sein. Dies ist für den Affen in Abb. 18 der Fall. Da er mit der Hand zur äußeren Seite des Irrgartens ständigen Kontakt hält, ist seine Schleife deutlich größer als im ersten Beispiel.

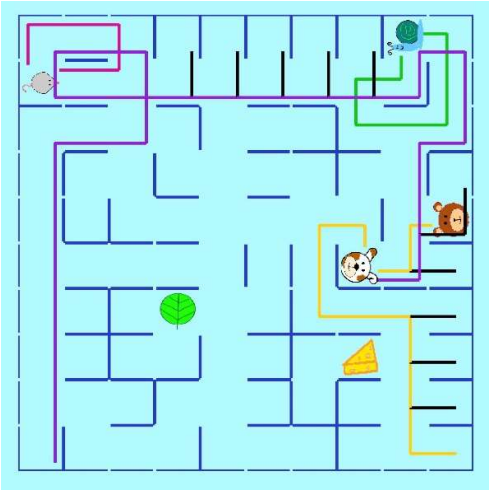


Abbildung 17: Schleifen um Inseln

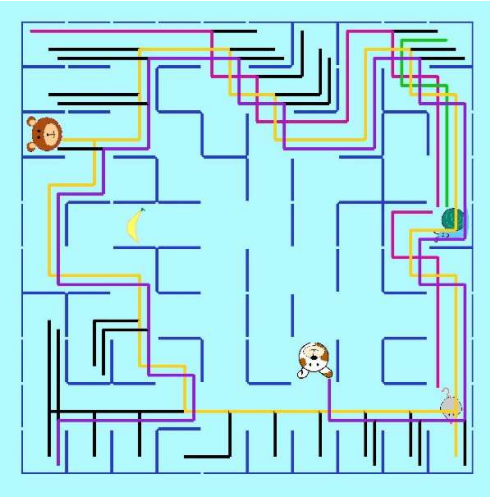


Abbildung 18: Schleifen entlang der Außenwand

4.6 Experiment 6: Floodfill Algorithmus

Beschreibung: In diesem Experiment wird als Lösung zu Experiment 5 der Floodfill-Algorithmus erforscht. Dabei wurde das erste Experiment wiederholt, wobei die Roboter statt der Linken- bzw. Rechten-Hand-Methode den Floodfill-Algorithmus anwenden, um von einer Ecke zu einem beliebigen Zielpunkt zu gelangen. Außerdem haben die Irrgärten diesmals insgesamt mn / 5 Lücken. Wieder wurden der minimale, maximale und durchschnittliche Batterieverbrauch gemessen.

Beobachtung/Erklärung:

Wie in der Tabelle (Abb. 19) zu sehen ist, steigt der durchschnittliche Batterieverbrauch, je größer der Irrgarten ist. Das ist verständlich, da der Weg von einer zur gegenüberliegenden Irrgartenecke länger wird. Der minimale Batterieverbrauch entspricht meistens null, da der Zielpunkt beliebig ausgesucht wird. Bei kleinen Irrgärten ist es wahrscheinlicher, dass der Zielpunkt dem Startpunkt entspricht, da es dort eine geringere Anzahl an Kästchen gibt. Der maximale Batterieverbrauch beträgt keine so hohen Werte, da es Lücken im Irrgarten gibt. Der durchschnittliche Batterieverbrauch ist deutlich weniger als im ersten Experiment mit der

Linken- bzw. Rechten-Hand-Methode. Das liegt daran, dass der Floodfill Algorithmus stets dem Zielpunkt folgt und es im Irrgarten Lücken gibt.

Irrgartengröße	Minimaler Batterieverbrauch	Maximaler Batterieverbrauch	Durchschnittlicher Batterieverbrauch
10 * 10	0	37	13.50
15 * 15	0	61	22.80
20 * 20	0	80	30.64
25 * 25	1	102	40.74
30 * 30	0	125	48.80
35 * 35	1	141	57.66
40 * 40	0	159	65.58

Abbildung 19: Resultate beim Floodfill Algorithmus

5 Entwicklung einer Strategie mit RL

Schließlich wird eine weitere Verbesserung für das strategische Irrgartenlösen mithilfe von RL eingebaut. Die Frage besteht darin, ob die Roboter sich stets zur selben Zeit bewegen müssen. Wird es sich als Roboter lohnen, zwischendurch zu warten, während die anderen Roboter den Irrgarten genauer untersuchen können? Es wird untersucht, ob RL zu einer richtigen Strategie führen kann.

5.1 AlphaZero

Die Verwendung von Baumdiagrammen ist eine übliche Methode, um einer Maschine Strategien für Spiele, insbesondere Brettspiele, beizubringen. Dabei enthält das Baumdiagramm alle möglichen Zustände des Spiels. Diese Methode wird als Zustandsraumanalyse bezeichnet und kann beispielsweise für das Brettspiel Tic-Tac-Toe verwendet werden. Allerdings ist zu beachten, dass die Anzahl an Zuständen bei Spielen wie Schach oder Go enorm groß sein kann. Bei solch komplexen Brettspielen kann man den Monte-Carlo-Tree-Search (abgekürzt MCTS) anwenden.

In MCTS zieht der Agent, anstatt alle Zustände einer Strategie zu berechnen, stets den meistvertrauten Zug. Dies tut er durch seine Erfahrung von voherigen Versuchen. Jedoch erforscht dieser auch neue Züge, um keine guten Züge zu übersehen. Um sein Vertrauen zu berechnen, kann er den UCB Code verwenden, der unten angezeigt wird. UCB steht für "Upper Confidence Bound" (Oberer Vertrauensbereich).

$$UCB_i = \bar{X}_i + c\sqrt{\frac{\ln(t)}{N_i}}$$

Hier ist i der Zug, von dem der Agent den Vertrauenswert berechnen will. Dabei steht $\bar{X}_i$ für seine Erfahrungen aus vorherigen Versuchen. Hat er beispielsweise in der Vergangenheit schon einen guten Zug gezogen, erzielt dieser Zug einen höheren Vertrauenswert. Das liegt daran, dass $\bar{X}_i$ einen hohen Wert beträgt, da schon bewusst ist, das der Zug gut ist.

Der zweite Summand $c\sqrt{\frac{\ln(t)}{N_i}}$ bezeichnet die Bevorzugung der noch nicht gezogenen Zügen. Dabei erhöht sich im Bruch $\frac{\ln(t)}{N_i}$ der Zähler mit der Zeit und der Nenner entspricht die Anzahl an Zügen, wo der Zug i gezogen wurde.

In MCTS gibt es vier Schritte:

1. Selection (Übergangsgleichung ermitteln)
2. Expansion (Eingabeparameter definieren)
3. Simulation (Simulation einrichten)
4. Backpropagation (Prozessausgaben analysieren)

Zunächst wird der Startszustand als einzigen Knoten eines Diagramms gesehen. Dieser wird im Laufe des Trainierens des Modells immer weiter ergänzt. In Selection, den ersten Schritt, wird ein noch nicht ergänzter Zug ausgewählt. In diesem Fall ist der Startzustandsknoten der einzige im Diagramm. Nun wird im nächsten Schritt, dem Expansion, dieser Knoten durch einen beliebigen Zug ergänzt. In Simulation werden beliebige Schritte bis zum Spielende durchdacht. In dem letzten Schritt, dem Backpropagation, wird der im Selection ausgewählter Zug und die Züge davor mithlife des UCB Codes bewertet.

5.2 Implementierung und Beobachtungen

Viele Implementierungen von AlphaZero kann man online finden. Für dieses Projekt wurde eine schon exestierende Implementierung erweitert [6] und zum Irrgartenspiel umgeschrieben. Der Aufbau von AlphaZero ist gleich geblieben. Umgeschrieben wurde die Spielbeschreibung und die Belohnungen. Im Folgenden wird die Implementierung des RL Models beschrieben.

Spiel: Wie in den Experimenten beginnen die Roboter in den einzelnen Ecken in einem Irrgarten mit m = n = 10. Dabei haben zwei Roboter immer 25 Batterie, also eine ziemlich geringe Batteriemenge, und die anderen beiden 100 Batterie, was ziemlich viel sein kann. Bei jedem Schritt haben die Roboter mit geringen Batteriemenge zwei Möglichkeiten: Entweder warten sie, oder sie bewegen sich mithilfe des Floodfill Algorithmus.

Belohnung: Es wird mit zwei verschiedenen Belohnungsmodellen experimentiert. Dazu gehört einerseits die Belohnung dafür, ob man das Ziel erreicht hat und andererseits, wie schnell man es erreicht hat. Hier wird das Belohnungsschema verdeutlicht:

Belohnungsmodell 1: 10 Punkte, wenn das Ziel zu erreicht wird, ansonsten -25 Punkte.

Belohnungsmodell 2: 10 Punkte, wenn das Ziel zu erreicht wird, zusätzlich noch (100 – T) Punkte, wobei das Ziel in T Schritten erreicht wurde, ansonsten -25 Punkte.

	Nur Schwarmintelligenz	**Schwarmintelligenz mit AlphaZero**
Belohnungsmodell 1	-25	**-17**
Belohnungsmodell 2	-12	**-5**

Abbildung 20: Vergleich zwischen mit und ohne AlphaZero

Man bemerkt, dass meistens die Roboter mit schwacher Batterie das Ziel nicht erreichen können. In solchen Fällen beträgt die Belohnung 10 + 10 – 25 – 25 = -30. Selten kann einer der schwachen Roboter es schaffen, das Ziel zu erreichen. Dabei beträgt die Belohnung 10 + 10 + 10 – 25 = 5.

Die von dem Reinforcement Learning Model gefundene Strategie ist, dass die Roboter mit geringer Batteriemenge solange warten müssen, bis nicht die Gefahr besteht, dass die Zeit ausläuft, während die mit hoher Batteriemenge den Irrgarten erforschen können. Nun können die Roboter mit schwacher Batterie mithilfe der ihnen zugeteilten Information von dem Irrgarten ihr Ziel erreichen, ohne dass sie batterielos werden.

Wie man in der Tabelle (Abb. 20) sieht, ist es besser mit AlphaZero als ohne, da die Werte bei *nur Schwarmintelligenz* weniger sind als bei *Schwarmintelligenz mit AlphaZero*.

6 Zusammenfassung

6.1 Projektarbeit

Nachdem mithilfe von Algorithmen ein beliebig ausschauendes Irrgarten programmiert wird, nutzen vier Roboter verschiedene Methoden an, um im Irrgarten gemeinsam eine Aufgabe mit einer beschränkten Batteriemenge erreichen zu können.

6.2 Ziel

Ziel dieses Projekts ist, Teile der Mathematik genauer zu verstehen und Methoden wie Algorithmen, Schwarmintelligenz und Reinforcement Learning zu erforschen und zu programmieren, um schwierige Situationen in unbekannten Irrgärten zu erleichtern. Die Resultate dieses Projektes sind mit autonomen Robotern zu verbinden, die sich einander helfen, um ein Ziel zu erreichen und dabei wenigere Ressourcen verbrauchen. In der Realität kann dieses Schema beispielsweise im Verkehr angewendet werden.

6.3 Probleme

- Gibt es Schleifen im Irrgarten, bleiben die Roboter mit der Linken- bzw. Rechten-Hand-Methode manchmal stecken
  → als Lösung wurde ein weiterer Algorithmus, der Floodfill Algorithmus programmiert, der für Irrgärten mit Schleifen eignet
- Oft trauen sich die Roboter nicht, von einem Kästchen aus weiter zu fahren und bewegen sich deshalb stets zu einer Ladestation und zurück
  → so wurden sie gezwungen weiterzufahren, anstatt die Batterie an einer Ladestation aufzufüllen
- Der Floodfill-Algorithmus ist nicht dermaßen schnell, da die Aktualisierung aller Zahlen der Kästchen im Irrgarten zu lange dauert
  → ungelöst
- Wird der Zielpunkt beliebig gewählt, ist der durchschnittliche Batterieverbrauch nicht in der Abhängigkeit von m und n bekannt
  → ungelöst

6.4 Weitere Schritte

Weitere Schritte sind beispielsweise das Programmieren eines Fangenspiels im Irrgarten oder auch das weitere Trainieren und Verbessern des Reinforcement Learning Models.

7 Quellenangaben

[1] https://de.wikipedia.org/wiki/Algorithmus_von_Prim

[2] https://de.wikipedia.org/wiki/Algorithmus_von_Kruskal

[3] https://de.wikipedia.org/wiki/L%C3%B6sungsalgorithmen_f%C3%BCr_Irrg%C3%A4rten

[4] https://de.wikipedia.org/wiki/AlphaZero

[5] https://github.com/vv-ss/avpython/tree/main/jugend_forscht_2023/maze_swarm

[6] https://medium.com/@_michelangelo_/alphazero-for-dummies-5bcc713fc9c6

Alle Abbildungen von Abb. 1 bis Abb. 20 wurden selber gestaltet

8 Unterstützungsleistungen

Unser Projekt wurde durch Christoph Bürgis und Jonas Röhrl unterstützt.