Berner Fachhochschule
Technik und Informatik

Projekt 2 Modul 7302

Fachbereich Technik und Infromatik
Frühlingssemester 2012

Ants - AI Challenge

Studierende: Lukas Kuster
Stefan Käser

Professoren: Dr. Jürgen Eckerle

Datum: 14. Juni 2012

Management Summary

Ants AI Challenge ist ein Programmierwettbewerb, bei welchem ein Bot programmiert wird der ein Ameisenvolk steuert. Das Ameisenvolk soll auf einer Map Futtersuchen sowie gegnerische Völker angreifen und vernichten. Dabei müssen Problem wie die Pfadsuche, das Verteilen von Aufgaben sowie das Schwarmverhalten gelöst werden. In unserer Arbeit wollten wir herausfinden was es alles braucht um einen solchen intelligenten Bot zu schreiben und gegen andere Mitspieler anzutreten. Wir konzetrierten uns auf die Aufgabenverteilung sowie die Pfadsuche. Diese Erfahrungen wollen wir für die Bachelorarbeit mitnehmen, wo wir an einer aktiven Challenge teilnehmen möchten oder uns in der für dieses Projekt verwendete Challenge vertiefen.

Datum	14. Juni 2012
Name Vorname	Lukas Kuster
Unterschrift	.....
Name Vorname	Stefan Käser
Unterschrift	.....

Inhaltsverzeichnis

1. Einleitung **1**

2. Spielbeschrieb **3**
2.1. Der Wettbewerb . . . 3
2.2. Spielregeln . . . 3
2.3. Schnittstelle . . . 3

3. Implementation **5**
3.1. Tasks . . . 5
3.2. Missionen . . . 6
3.3. Pfadsuche . . . 6
3.4. JavaScript Addon für HMTL-Gameviewer . . . 8

4. Rückblick **11**
4.1. Resultate . . . 11
4.2. Herausforderungen . . . 11
4.3. Ziele für Bachelorarbeit . . . 11

A. Beliebiger Anhang **13**

B. Weiterer Anhang **15**
B.1. Test 1 . . . 15

Glossar **17**

Literaturverzeichnis **19**

Stichwortverzeichnis **21**

Abbildungsverzeichnis

3.1. Simple-Path Algorithmus 7
3.2. Heuristische Kosten (blau), Effektive Kosten (grau) 7
3.3. Clustereinteilung auf der Landkarte. Clustergrösse 4x4, Landkarte 16x16 8
3.4. Die Kanten jedes Clusters wurden berechnet 8
3.5. Die Kanten jedes Clusters wurden berechnet 9

Tabellenverzeichnis

1. Einleitung

TODO

Dieses Dokument dient einerseites zur Illustration der LaTeX Vorlage anhand des Corporate Designs der Berner Fachhochschule und andererseits als Anleitung für deren Verwendung. Dabei wird vorausgesetzt, dass der Benutzer bereits Erfahrungen mit LaTeX besitzt oder gewillt ist, sich während der Benutzung in das Thema einzuarbeiten. Im Quellenverzeichnis sind einige nützliche Einträge zu diversen Büchern und Dokumenten im Internet über LaTeX zu finden.

2. Spielbeschrieb

2.1. Der Wettbewerb

Die AI Challenge[1] ist ein internationaler Wettbwerb des University of Waterloo Computer Science Club der im Zeitraum Herbst 2011 bis Januar 2012 stattgefunden hat. Das Spiel ist ein zugbasiertes Multiplayerspiel in welchem sich Ameisenvölker gegenseitig bekämpfen. Ziel einer AI-Challenge ist es, einen Bot zu schreiben, der die gegebenen Aufgaben mit möglichst intelligent Algrithmen löst. Die zu lösenden Aufgaben der Ants AI Challenge sind die Futtersuche, das Explorieren der Karten, das Angreifen von gegnerischen Völkern und deren Ameisenhaufen sowie dem Schützen des eigenen Ameisenhaufen.

2.2. Spielregeln

Nachfolgend sind die wichtigsten Regeln, die während dem Spiel berücksichtigt werden müssen, aufgelistet.

- Pro Zug können alle Ameisen um ein Feld (vertikal oder horizontal) verschoben werden.
- Pro Zug steht insgesamt eine Rechenzeit von einer Sekunde zur Verfügung. Es dürfen keine Threads erstellt werden.
- Bewegt sich eine Ameise in die 4er Nachbarschaft eines Futterpixel, wird dieses eingesammelt. Beim nächsten Zug entsteht bei dem Ameisenhügel eine neu Ameise.
- Die Landkarte besteht aus passierbaren Landpixel sowie unpassierbaren Wasserstellen.
- Ein Gegener wird geschlagen, wenn im Kampfradius der eigenen Ameise mehr eigene Ameise stehen als gegnerische Ameisen im Kampfradius der Ameise die angegriffen wird.
- Ein Gegner ist ausgeschieden wenn alle seine eigenen Ameisenhügel vom Gegner vernichtet wurden. Pro verlorener Hügel gib es einen Punkteabzug. Pro feindlicher Hügel der zerstört wird gibt es zwei Bonuspunkte.
- Steht nach einer definierbaren Zeit (Anzahl Züge) kein Sieger fest, wird der Sieger anhang der Punkte ermittelt.

Die ausführlichen Regeln können auf der Webseite nachgelesen werden: http://aichallenge.org/specification.php

2.3. Schnittstelle

Die Spielschnittstelle ist simpel gehalten. Nach jeder Spielrunde erhält der Bot das neue Spielfeld mittels String-InputStream, die Spielzüge gibt der Bot dem Spielcontroller mittels String-OutputStream bekannt. Unser MyBot leitet vom Interface Bot[2] ab. Ein Spielzug wird im folgendem Format in den Output-Stream gelegt:

o <Zeile> <Spalte> <Richtung>

Beispiel:

```
o 4 7 W
```

[1] http://www.aichallenge.org
[2] Das Interface ist im Code unter ants.bot.Bot.Java auffindbar

Die Ameise wird von der Position Zeile 4 und Spalte 7 nach Westen bewegt.
Der Spielcontroller ist in Python realisiert, der Bot kann aber in allen gänglichen Programmiersprachen wie Java, Python, C#, C++ etc. geschrieben werden.

3. Implementation

3.1. Modell

Für die Modellierung habe wir uns auf die nötigsten Klassen beschränkt, um das Modell einfach zu halten. Ein wichtiger Aspekt der Modellierung war dabei die Abbildung des Spiel-Zustands auf State-Klassen, die uns jederzeit Zugriff auf alle bekannten Variablen des Spiels bieten. Einige Informationen können dabei direkt von der Spiel-Engine übernommen werden, die meisten Zustands-Informationen werden aber berechnet.

Die Modellierung der Klassen, die Spiel-Elemente repräsentieren, war etwas einfacher; hier konnten wir auch einzelne Enumerationen u.ä. aus dem Beispiel-Bot der AI-Challenge übernehmen.

3.1.1. State-Klassen

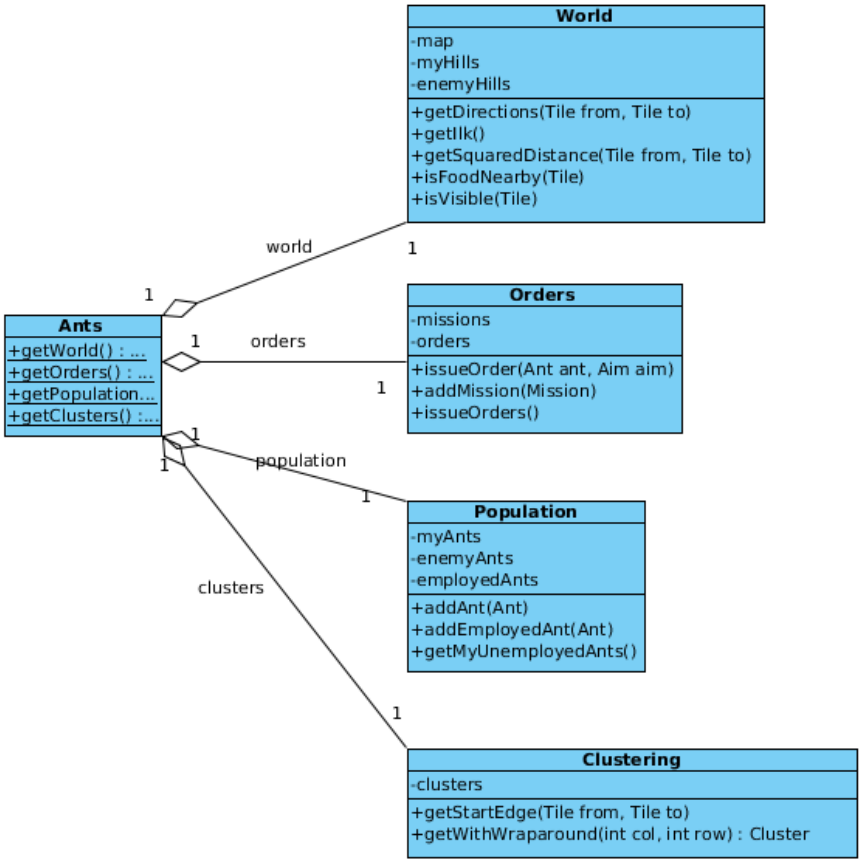

Abbildung 3.1.: State-Klassen (vereinfacht)

Abbildung **??** zeigt eine Übersicht über die Zustands-Klassen. Für das Diagramm wurden lediglich die wichtigsten Methoden und Attribute berücksichtigt. Die State-Klassen implementieren alle das Singleton-Pattern.

Ants

Die Ants Klasse ist die zentrale State-Klasse. Sie bietet auch einfachen Zugriff auf die anderen State-Klassen. Ursprünglich hatten wir alle Methoden, die mit dem Zugriff auf den Spielzustand zu tun hatten, direkt in der Ants Klasse implementiert, haben aber schnell gemerkt, dass das unhandlich wird. Die Ants Klasse dient jetzt vor allem als Container für die anderen State-Klassen und implementiert nur noch einige Methoden, die Zustandsänderungen in verschiedenen Bereichen vornehmen.

World

Die World Klasse enthält Informationen zur Spielwelt. Hier wird die Karte abgespeichert, in der für jede Zelle die aktuell bekannten Informationen festgehalten werden. Das beinhaltet die Sichtbarkeit der Zelle und was die Zelle aktuell enthält (Ameise, Nahrung, Wasser, ...). Ausserdem werden Listen geführt, wo sich die eigenen und die bekannten gegnerischen Hügel befinden. Die Klasse bietet Methoden zur Distanzberechnung, gibt Auskunft über einzelne Zellen und darüber, ob sich Nahrung in der Umgebung einer bestimmten Zelle befindet.

Orders

In der Orders Klasse wird über Befehle und Missionen der einzelnen Ameisen Buch geführt. Die Liste der Befehle wird dabei in jedem Zug geleert und neu befüllt, während die Liste der Missionen Zug-übergreifend geführt wird. Das zentrale Verwalten der Befehle dient vor allem dazu, sicherzustellen, dass keine widersprüchlichen Befehle ausgegeben werden (mehrere Befehle für eine Ameise, gleiche Ziel-Koordinaten für mehrere Ameisen, ...)

Population

Die Population Klasse dient der Verwaltung der eigenen und der gegnerischen Ameisen-Völker. Hier werden die Ameisen mit ihren aktuellen Aufenthaltsorten festgehalten. Wenn für eine Ameise ein Befehl ausgegeben wird, wird diese als beschäftigt markiert; über die Methode `getMyUnemployedAnts()` kann jederzeit eine Liste der Ameisen abgefragt werden, die für den aktuellen Zug noch keine Befehle erhalten haben.

Clustering

Die Clustering Klasse dient dem Aufteilen des Spielfeldes in Clusters für die HPA*-Suche (s. Abschnitt 3.3.3). Hier werden die berechneten Clusters abgelegt, und der Zugriff auf sie erfolgt ebenfalls über die Clustering Klasse.

3.1.2. Spiel-Elemente (Welt)

Abbildung **??** zeigt die wichtigsten Klassen, die die Elemente des Spiels repräsentieren. Der Übersichtlichkeit wegen wurden nur die wichtigsten Attribute und Operationen in das Diagramm aufgenommen.

3.1.3. Spiel-Elemente (Suche)

Abbildung **??** zeigt die wichtigsten Klassen, die für die Pfadsuche verwendet werden.

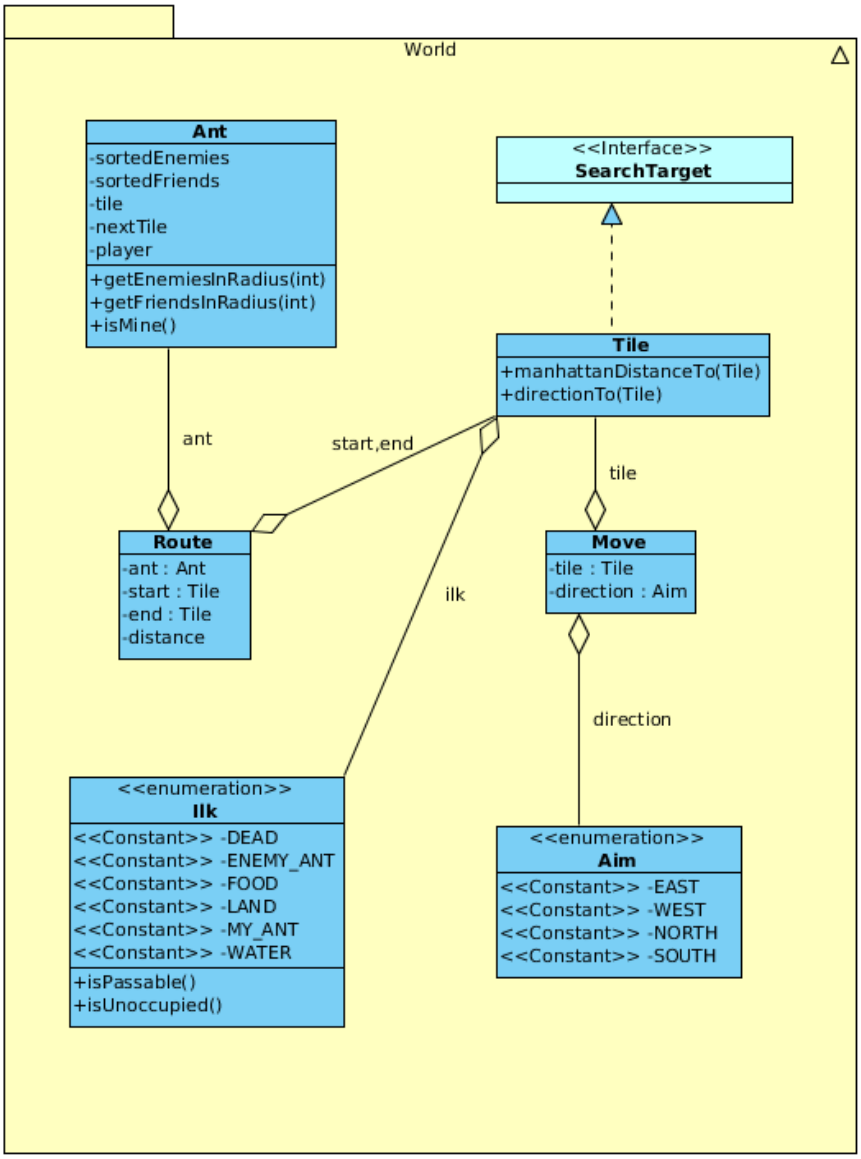

Abbildung 3.2.: Spiel-Elemente der Spielwelt (vereinfacht)

3.1.4. Tasks

Die Tasks bzw. Aufgaben des Bots wurden in eigenen Klassen implementiert. Das Interface Task [1] definiert eine setup()-Methode welche den Task initiert, sowie eine perform()-Methode welche den Task ausführt. Im Program werden die Tasks nach deren Wichtigkeit ausgeführt, was auch der nachfolgenden Reihenfolge entspricht. Jeder Task kann nur auf die unbeschäftigten Ameisen zur Verfügung, d.h. jene welchen noch keine Aufgabe zugeteilt wurde.

MissionTasks

Dieser Task prüft alle akutellen Missionen auf deren Gültigkeit wie zum Beispiel, ob die Ameise der Mission noch am Leben ist. Falls gültig, wird der nächste Schritt der Mission ausgeführt.

[1] das Interface ist im Code unter ants.tasks.Bot.Java auffindbar

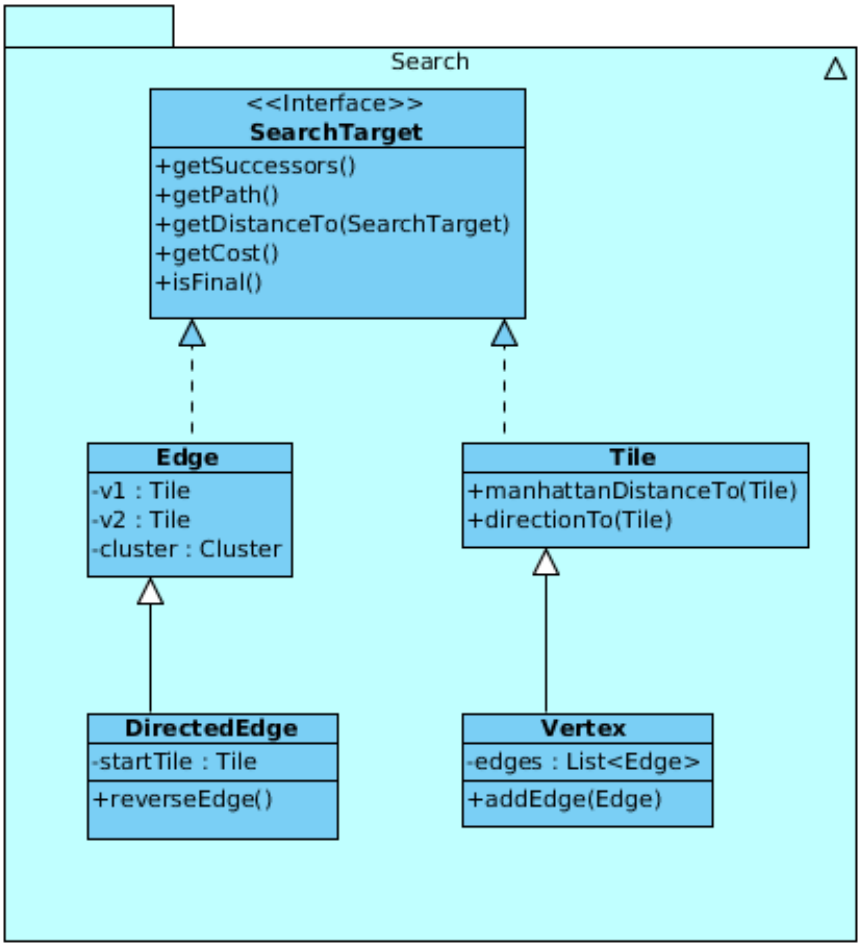

Abbildung 3.3.: Spiel-Elemente für die Suche (vereinfacht)

GatherFoodTask

Für jedes Food-Tile wird in einem definierbaren Radius r die nächsten Ameisen bestimmt. Danach wird aufsteigend der Luftliniendistanz versucht mit dem Pfadsuchalgorithmus SiMPLE oder falls dieser kein Pfad gefunden hat mit A* eine passierbare Route gesucht. Falls diese existiert wird mit der Ameise und dem Food-Tile eine GatherFoodMission erstellt, welche die Ameise zum Food-Tile führt. Zu jedem Food-Tile wir immer nur eine Ameise geschickt.

AttackHillsTask

Sobald gegnerische Ameisenhaufen sichtbar sind, sollen diese angegriffen werden, da dies +2 Punkte gibt. Die Kriterien, dass eine Pfad zum gegnerischen Haufen gesucht wird, sind die selben wie beim GatherFoodTask, ausser dass mehrere Ameisen das Ziel angreifen können. Es wird ein AttackHillMission erstellt.

CombatTask

Beim Angriffstask wird berechnet ob wir in einem Kampfgebiet (viewRadius2) die Überhand, d.h mehr Ameisen platziert haben. Falls ja wird die gegnerische Ameise angegriffen.

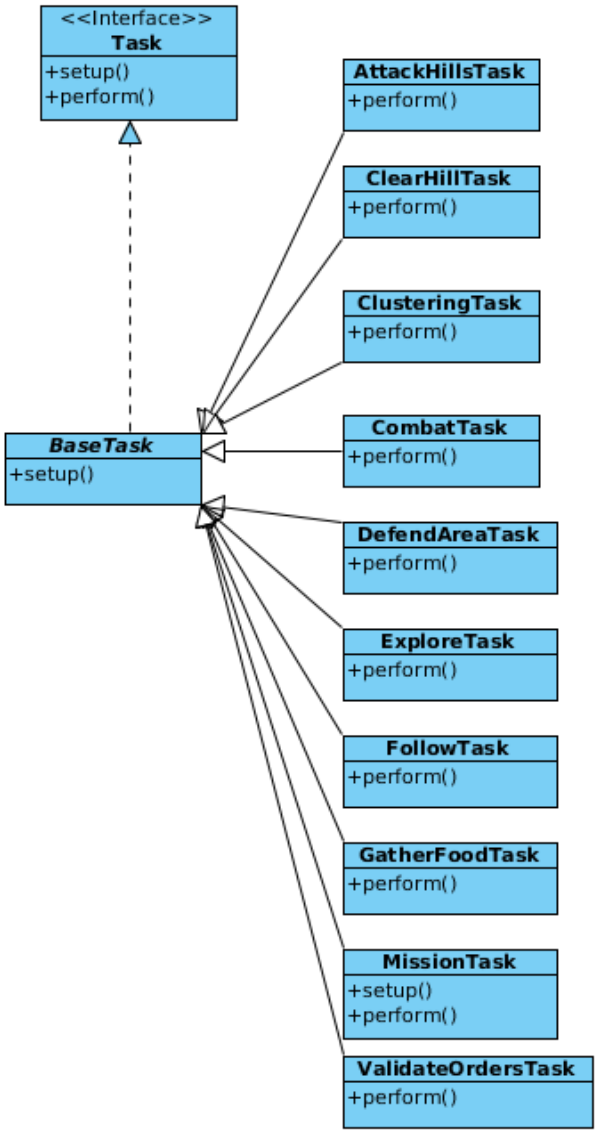

Abbildung 3.4.: Tasks

DefendAreaTask

Dieser Task wäre vogesehen um eine Region wie zum Beispiel der eingene Ameisenhügel zu schützen. Dieser Task ist aber noch nicht implementiert.

ExploreTask

Für alle noch unbeschäftigten Ameisen wird mittels ManhattenDistance der nächste Ort gesucht, der noch nicht sichtbar, also unerforscht ist. Falls ein Pfad mittels Pfadsuchalgorithmus gefunden wird, wird eine ExplorerMission (s. Abschnitt 3.2) erstellt, das heisst die Ameise wird den gefundenen Pfad in den nächsten Spielzügen ablaufen.

FollowTask

Der FollowTask ist für Ameisen angedacht welche aktuell keine Aufgabe haben. Diese Ameisen sollen einfach einer beschäftigten Ameise folgen, damit diese nicht alleine unterwegs ist.

ClearHillTask

Dieser Task bewegt alle Ameisen, welche neu aus unserem Hügel ßchlüpfenünd noch keinen Befehl haben, davon weg. So werden nachfolgende Ameisen nicht durch diese blockiert.

ClusteringTask

Der ClusteringTask wird als Vorbereitung für den HPA* Algorithmus verwendet. Hier wird alle sichtbaren Kartenregionen ein Clustering vorgenommen. Das Clustering wird im Kapitel 3.3.3 im Detail beschreiben.

3.1.5. Missionen

Eine Mission dauert über meherer Spielzüge. Die meisten Missionen (GatherFoodMission,ExploreMission,AttackHillMission,AttackAntM sind Pfadmissionen[2] bei welchen die Ameise einem vorgegebenen Pfad, der bereits beim Erstellen der Mission berechnet wurde, folgt. Je nach spezifischer Mission sind aber die Abbruchbedingungen anders. Zum Beispiel die GatherFoodMission ist nur solange gültig wie das Futter noch nicht von einer anderen Ameise eingesammelt wurde.

3.2. Bot

3.3. Pfadsuche

Wir haben drei mögliche Pfadalgorithmen in unserem Code eingebaut. Via Klasse PathFinder kann für die Pfadsuche der Alogrithmus ausgewählt werden.

3.3.1. Simple Algorthmus

Der Simple Algorithmus versucht das Ziel zu erreichen indem er zuerst die eine, dann die andere Achse abläuft. Sobald ein Hindernis in den Weg kommt bricht der Algorithmus ab. Im folgenden Beispiel sucht der Alogrithmus den Vertikal-Horizontal Pfad. Da dieser Pfad wegen dem Wasserhindernis (blau) nicht ans Ziel führt, wird via Horizontal-Vertikal Pfad gesucht. Hier wird der Pfad gefunden. Dieser Algorithmus ist, wie der Name bereits aussagt, sehr einfach aufgebaut und kostet wenig Rechenzeit. Dafür kann er keinen Hindernissen ausweichen.

3.3.2. A* Algorthmus

Beim A* Algorithmus wird für jeden expandierten Knoten einen heuristischen Wert f(x) für gesamte Pfadlänge berechnet. Dass heisst f(x) besteht aus einem Teil g(x) welches die effektiven Kosten vom Startknoten zum aktuellen Knoten berechnet. Der andere Teil ist ein heuristischer Wert der Pfadkosten welche bis zum Zielknoten noch anfallen werden. Dieser Wert muss die effektiven Kosten zum Ziel immer unterschätzen. Dies ist in unserem Spiel dadurch gegeben, dass sich die Ameisen nicht diagonal bewegen können, wir aber für den heuristischen Wert die Luftlinie zum Ziel nehmen. Die Pfadsuche wird immer bei dem Knoten fortgesetzt welcher die kleinsten Kosten f(x) hat.

Das Bild zeigt, dass der effektive Pfad (grau) vom expandierenden roten Knoten minimal 10 Pixel lang sein kann. Die Luftlinie (blau) als heurstischer Pfad hat aber nur die Länge 7.6 Pixel. Damit erfüllt unsere Implementation die Anforderungen des Algorithmus.

Dieser Algorithmus wird in unserem Code für eine Pfadsuche über alle Pixel (jedes Pixel ist ein Node) verwendet aber auch für die berechneten Kanten welche im HPA* verwendet werden.

[2] Die abstrakte Klasse PathMission ist im Code unter ants.missions.PathMission.java auffindbar

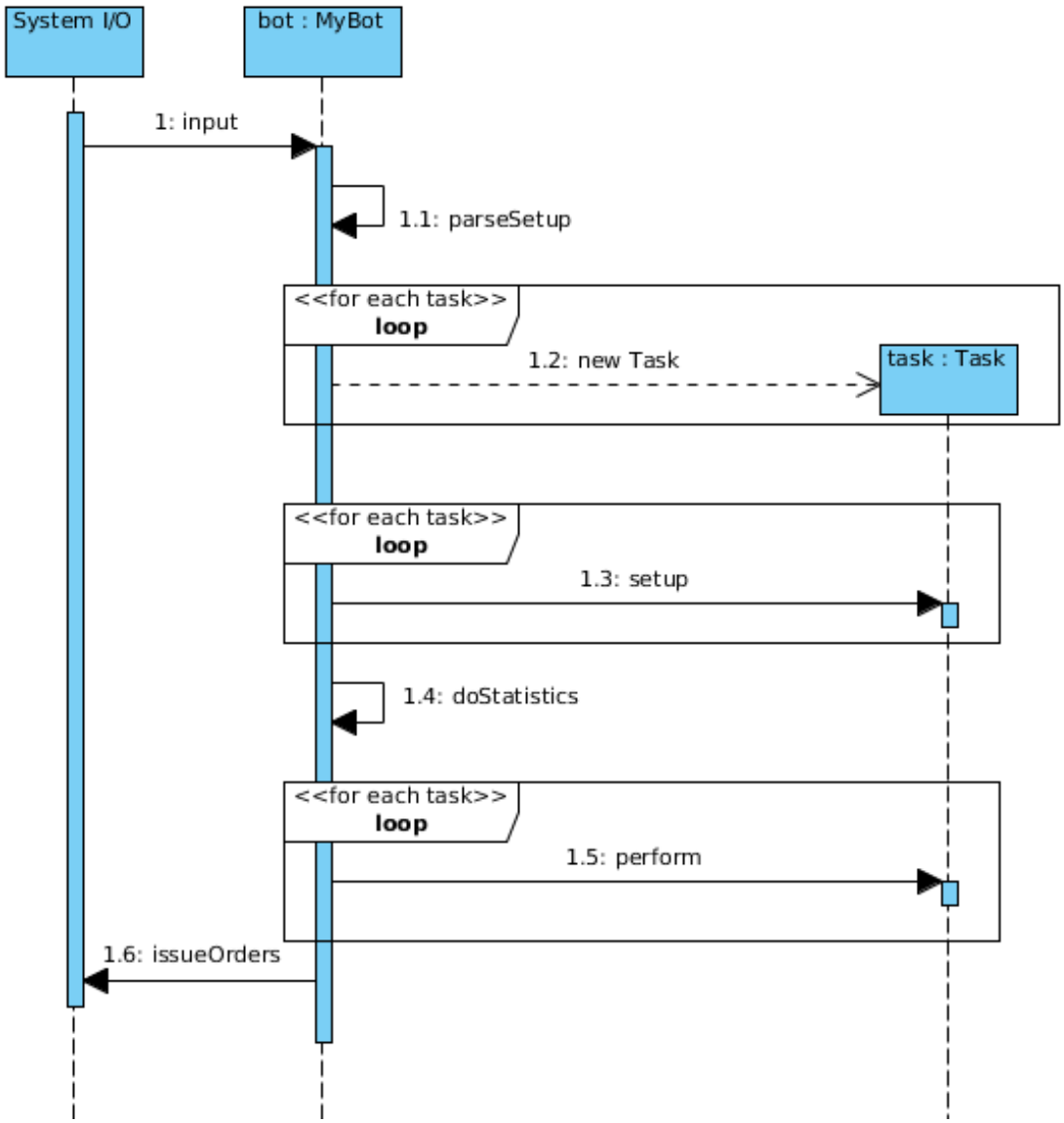

Abbildung 3.5.: Ablauf des ersten Zugs des Spiels

3.3.3. HPA* Algorthmus

Eine Pfadsuche A* überalle Pixel ist sehr teuer, da es viel Pfade gibt, die zum Teil nur ein Pixel nebeneinander liegen. Es werden bis zum Schluss verschiedenen Pfaden nachgegangen. Abhilde zu dieser sehr feinmaschigen Pfadsuche bietet der Hierarcical Pathfinding A* bei welchem im sogenanten Clustering über mehrere Pixel verlaufende Kanten und Knoten berechnet werden.

Clustering Das Clustering wird während dem ClusteringTask ausgeführt, Dabei wird die Landkarte in sogenannte Clusters unterteilt. Auf dem Bild 3.3 wurde die Karte in 16 Clusters aufgeteilt.

Danach wird für jeden Cluster und ein Nachbar aus der vierer Nachbarschaft die Verbindungskanten berechnet. Dies kann natürlich nur für Clusters gemacht werden bei welchen die Landkarte bereits komplett sichtbar ist, was zu Begin des Spiel nicht gegeben wird. Deshalb wird der ClusteringTask in jedem Spielzug aufgerufen, in der Hoffnung ein Clusterkomplett verbinden zu können. Sobald eine beliebige Seite eines Clusters berechnet ist wird diese Aussenkante beim Cluster und dem anleigenden Nachbar gespeichert und nicht mehr berechnet.

Sobald ein Cluster zwei oder mehrere Aussenkanten kennt berechnet er die Innenkanten mit A* welche die Knoten der Aussenkanten verbindet. Dies ergibt nun ein Pfadnetz über die Gesamtekarte. Nun wird ein Pfad vom Pixel

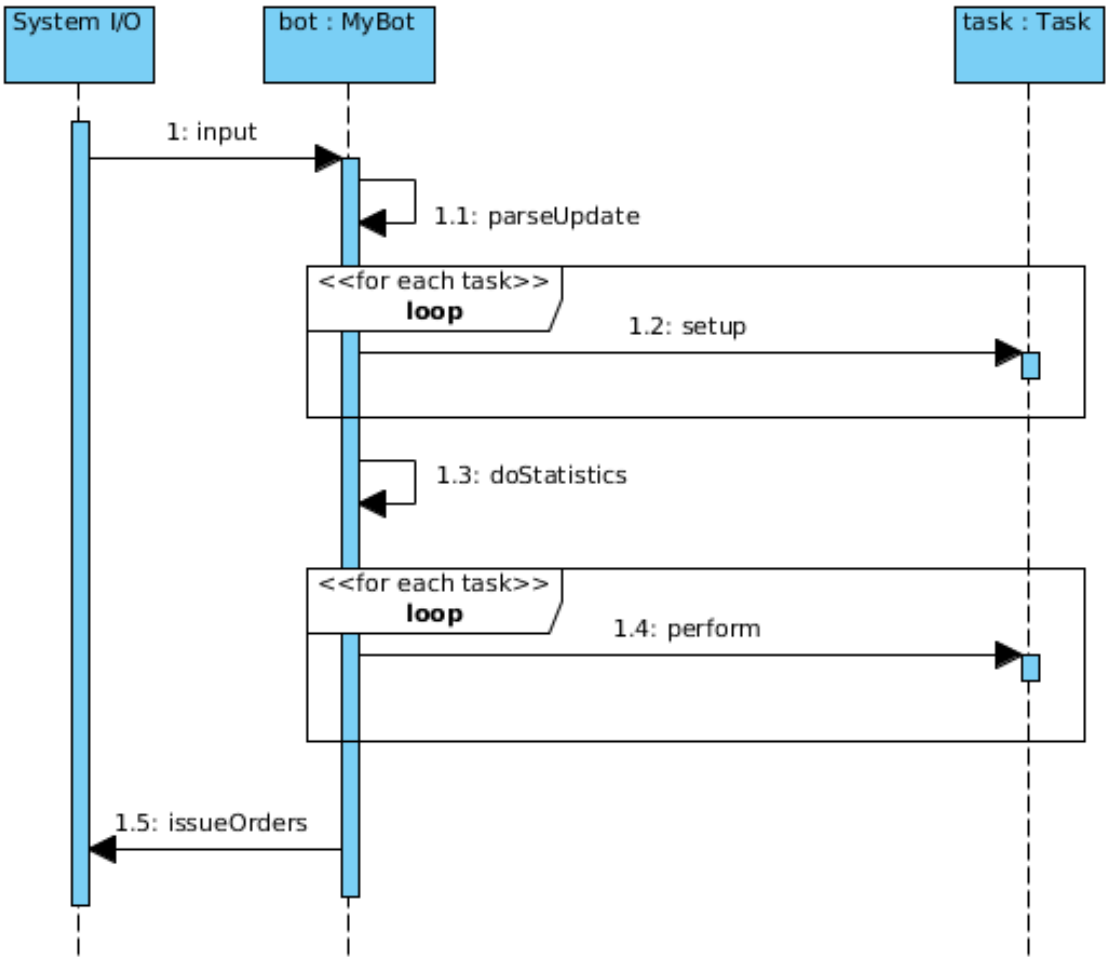

Abbildung 3.6.: Ablauf der weiteren Züge des Spiels

(0,9) nach (14,9) gesucht. Zuerst wird eruiert in welchem Cluster sich das Start- bzw Zielpixel befindet. Danach wird in dem gefunden Cluster ein Weg zu einem beliebigen Knoten auf der Clusterseite gesucht. Sind diese Knoten erreicht kann nun das vorberechnete Pfadnetz mittels bereits beschrieben A* Algroithmus verwendet werden um die beiden Knoten auf dem kürzesten möglichen Pfad zu verbinden.[3]

TODO Beispielbild

3.4. JavaScript Addon für HMTL-Gameviewer

Das Codepaket welches von den Challengeersteller mitgeliefert wird bietet bereits eine hilfreiche 2D-Visualisierung des Spiels mit welchem das Spielgeschehen mitverfolgt werden kann. Die Visualisierung wurde mit HMTL und Javascript implementiert. Leider ist es nicht möglich zusätzliche Informationen auf die Seite zu projizieren. Deshalb haben wir den Viewer mit einer solchen Funktion erweitert. Mit der Codezeile Logger.liveInfo(...) kann eine Zusatzinformation geschrieben werden. Es muss definiert werden mit welchem Zug und wo auf dem Spielfeld die Infomation angezeigt werden soll. Im Beispiel wird an der Position der Ameise ausgegeben welchen Task die Ameise hat.

```
Logger.liveInfo(Ants.getAnts().getTurn(), ant.getTile(), "Task: %s ant: %s", issuer, ant.getTile());
```

[3]Der gefundene Pfad könnte mittels Pathsmoothing verkürzt werden. Dies wurde aber in unserer Arbeit nicht implementiert.

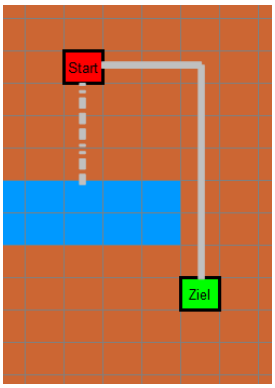

Abbildung 3.7.: Simple-Path Algorithmus

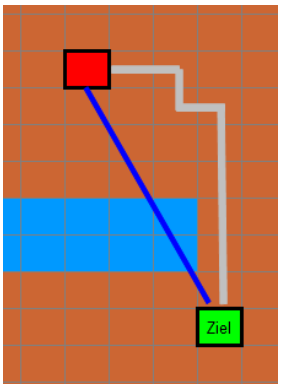

Abbildung 3.8.: Heuristische Kosten (blau), Effektive Kosten (grau)

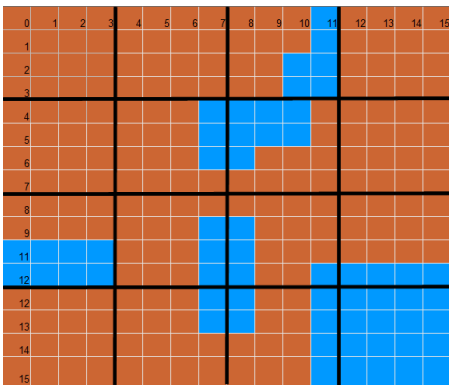

Abbildung 3.9.: Clustereinteilung auf der Landkarte. Clustergrösse 4x4, Landkarte 16x16

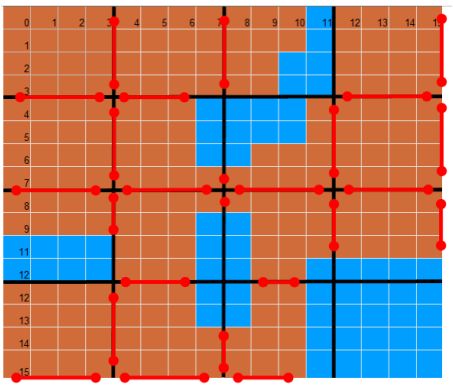

Abbildung 3.10.: Die Kanten jedes Clusters wurden berechnet

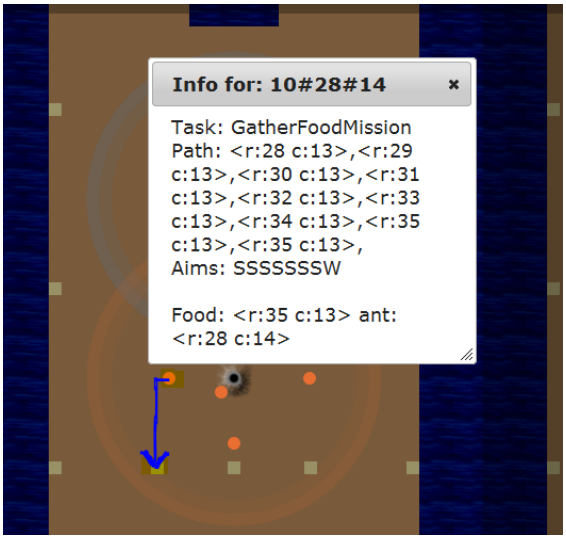

Abbildung 3.11.: Das Popup zeigt dei Aufgabe und den Pfad (blau), welcher die Ameise ablaufen wird.

Auf der Karte wird ein einfaches aber praktisches Popup angezeigt. Dank solchen Zusatzinfomrationen muss nicht mühsam im Log nach geschaut werden, welcher Ameise wann und wo welcher Task zugeordnet ist.

Das angezeigte Popup zeigt welchen Task (GatherFoodTask) die Ameise hat, wo sie sich befindet <r:28 c:14>, welches Futterpixel angesteuert wird <r:35 c:13> und welchen Pfad dazu berechnet wurde.

4. Rückblick

4.1. Resultate

TODO

4.2. Herausforderungen

TODO

4.3. Ziele für Bachelorarbeit

TODO

A. Beliebiger Anhang

Phasellus eget velit massa, sed faucibus nisi. Etiam tincidunt libero viverra lorem bibendum ut rutrum nisi volutpat. Donec non quam vitae lacus egestas suscipit at eu nisi. Maecenas non orci risus, at egestas tellus. Vivamus quis est pretium mauris fermentum consectetur. Cras non dolor vitae nulla molestie facilisis. Aliquam euismod nisl eget risus pretium non suscipit nulla feugiat. Nam in tortor sapien. Nam lectus nibh, laoreet eu ultrices nec, consequat nec sem. Nulla leo turpis, suscipit in vulputate a, dapibus molestie quam. Vestibulum pretium, purus sed suscipit tempus, turpis purus fermentum diam, id cursus enim mi a tortor. Proin imperdiet varius pellentesque. Nam congue, enim sit amet iaculis venenatis, dui neque ornare purus, laoreet porttitor nunc justo vel velit. Suspendisse potenti. Nulla facilisi.

B. Weiterer Anhang

B.1. Test 1

Phasellus eget velit massa, sed faucibus nisi. Etiam tincidunt libero viverra lorem bibendum ut rutrum nisi volutpat. Donec non quam vitae lacus egestas suscipit at eu nisi. Maecenas non orci risus, at egestas tellus. Vivamus quis est pretium mauris fermentum consectetur. Cras non dolor vitae nulla molestie facilisis. Aliquam euismod nisl eget risus pretium non suscipit nulla feugiat. Nam in tortor sapien.

B.1.1. Umfeld

Nam lectus nibh, laoreet eu ultrices nec, consequat nec sem. Nulla leo turpis, suscipit in vulputate a, dapibus molestie quam. Vestibulum pretium, purus sed suscipit tempus, turpis purus fermentum diam, id cursus enim mi a tortor. Proin imperdiet varius pellentesque. Nam congue, enim sit amet iaculis venenatis, dui neque ornare purus, laoreet porttitor nunc justo vel velit. Suspendisse potenti. Nulla facilisi.

Literaturverzeichnis

Stichwortverzeichnis

Berner Fachhochschule, 1
Bibliographie, 5
booktabs, 3

cmbright, 3
Corporate Design, 1

fancyhdr, 3

geometry, 3
Glossar, 4
glossaries, 3
graphicx, 3

hyperref, 3

makeidx, 3
makeindex, 3

Paket, 3

Stichwortverzeichnis, 3

Textcodierungen, 3
textpos, 3
Thesis, 1

Verbesserungsvorschläge, 1