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 Karte Futter suchen 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 konzentrierten uns auf die Aufgabenverteilung sowie die Pfadsuche. Diese Erfahrungen wollen wir für die Bachelorarbeit mitnehmen, wo wir an der nächsten AI-Challenge, die voraussichtlich im August beginnt, aktiv teilnehmen möchten, oder unsere Implementierung für diese Challenge verbessern.

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. Modell . . . 5
3.2. Bot . . . 11
3.3. Pfadsuche . . . 13
3.4. JavaScript Addon für HMTL-Gameviewer . . . 15

4. Rückblick **17**
4.1. Resultate . . . 17
4.2. Herausforderungen . . . 17
4.3. Ziele für Bachelorarbeit . . . 17

A. Spielanleitung **19**

Abbildungsverzeichnis

3.1. State-Klassen (vereinfacht) . . . 5
3.2. Spiel-Elemente der Spielwelt (vereinfacht) . . . 7
3.3. Spiel-Elemente für die Suche (vereinfacht) . . . 8
3.4. Tasks . . . 9
3.5. Missionen . . . 11
3.6. Ablauf des ersten Zugs des Spiels . . . 12
3.7. Ablauf der weiteren Züge des Spiels . . . 13
3.8. Simple-Path Algorithmus . . . 14
3.9. Heuristische Kosten (blau), Effektive Kosten (grau) . . . 14
3.10. Clustereinteilung auf der Landkarte. Clustergrösse 4x4, Landkarte 16x16 . . . 15
3.11. Die Kanten jedes Clusters wurden berechnet . . . 15
3.12. Das Popup zeigt dei Aufgabe und den Pfad (blau), welcher die Ameise ablaufen wird. . . . 16

1. Einleitung

Im Rahmen des Moduls "Projekt 2" (7302) haben wir uns mit der Implementierung eines Bots für den Online-Wettbewerb AI-Challenge (Ants) beschäftigt. Die AI-Challenge ist ein Wettbewerb, der im Herbst 2011 zum 3. Mal stattfand und jedes Jahr mit einem anderen Spiel durchgeführt wird. Ziel ist es jeweils, einen Bot zu programmieren, der durch geschickten Einsatz von KI-Technologien das Spiel möglichst erfolgreich bestreiten kann. In dieser Durchführung ging es darum, ein Ameisenvolk durch Sammeln von Ressourcen und Erobern von gegnerischen Hügeln zum Sieg über die gegnerischen Ameisen zu führen.

Wir hatten uns zum Ziel gesetzt, einen Bot zu implementieren, der möglichst alle Bereiche des Spiels beherrscht, also Nahrung sammeln, die Gegend entdecken, Hügel erobern und gegen feindliche Ameisen kämpfen. Im Gegenzug legten wir kein besonderes Gewicht darauf, dass der Bot eines dieser Verhalten besonders gut beherrschen muss. Das primäre Ziel war es, Erfahrungen zu sammeln im Hinblick auf die Bachelor-Arbeit.

Den grössten Aufwand bei der Implementierung steckten wir in die Pfadsuche, da diese eine Voraussetzung für nahezu alle Teil-Aufgaben des Bots ist. Nachdem wir mit dem bekannten A*-Algorithmus zwar kleine Erfolge erzielten, aber auch schnell Performance-Probleme bekamen, entschlossen wir uns, auf Basis eines Clustering des Spielfeldes den HPA*-Algorithmus umzusetzen, was zu erheblichen Performance-Verbesserungen führte.

Ein weiterer Punkt, auf den wir viel Wert legten, war die Programmstruktur. Unser Bot ist Objektorientiert aufgebaut; die zentrale Einheit sind die verschiedenen Tasks, die jeweils für eine Aufgabe der Ameisen zuständig sind.

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 Zustandsinformationen werden aber berechnet.

Die Modellierung der Klassen, die Spielelemente 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

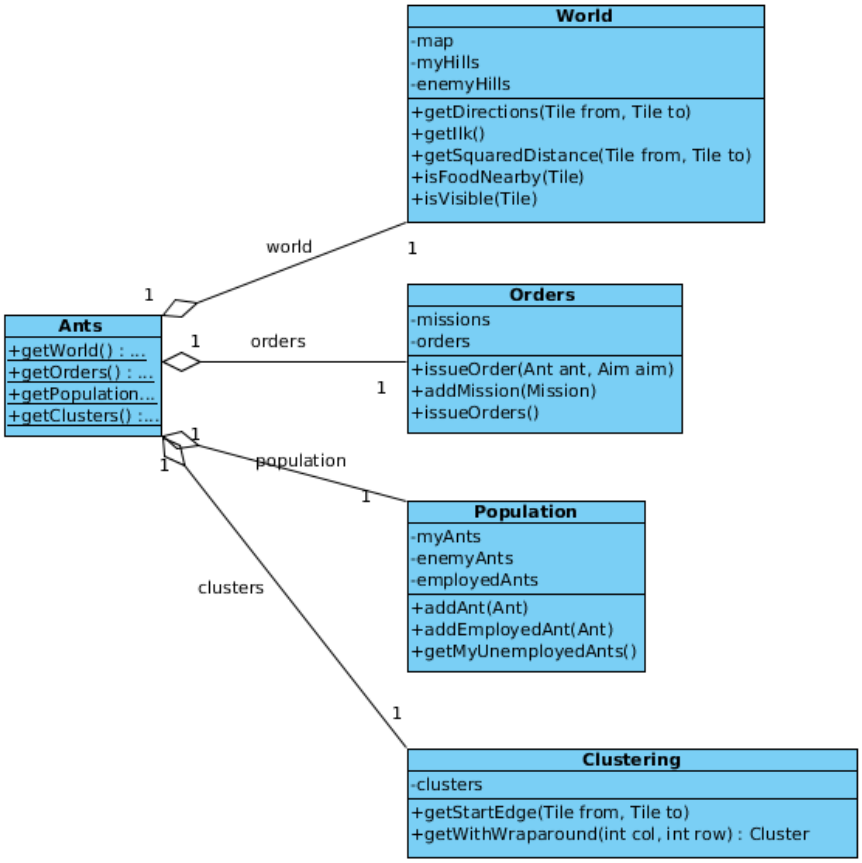

Abbildung 3.1.: State-Klassen (vereinfacht)

Abbildung 3.1 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 zugübergreifend 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 die Ameise 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, der Zugriff auf sie erfolgt ebenfalls über die Clustering Klasse.

3.1.2. Spiel-Elemente (Welt)

Abbildung 3.2 zeigt die wichtigsten Klassen, die die Elemente des Spiels repräsentieren. Der Übersichtlichkeit wegen wurden nur die wichtigsten Attribute und Operationen in das Diagramm aufgenommen.

Ant

Eine Ant gehört immer zu einem Spieler; über die Methode isMine() können unsere eigenen Ameisen identifiziert werden. Eine Ameise weiss jeweils, in welcher Zelle sie steht. Das Feld nextTile dient der Verfolgung einer Ameise über mehrere Züge – das Feld wird jeweils aktualisiert, wenn der Ameise ein Befehl ausgegeben wird; im nächsten Zug können wir dann prüfen, ob die Ameise den Befehl korrekt ausführen konnte. Eine Ameise kennt auch die anderen Ameisen in ihrer Umgebung: Über die Methoden getEnemies/FriendsInRadius() können alle bekannten Freunde und Feinde in einem bestimmten Radius ermittelt werden.

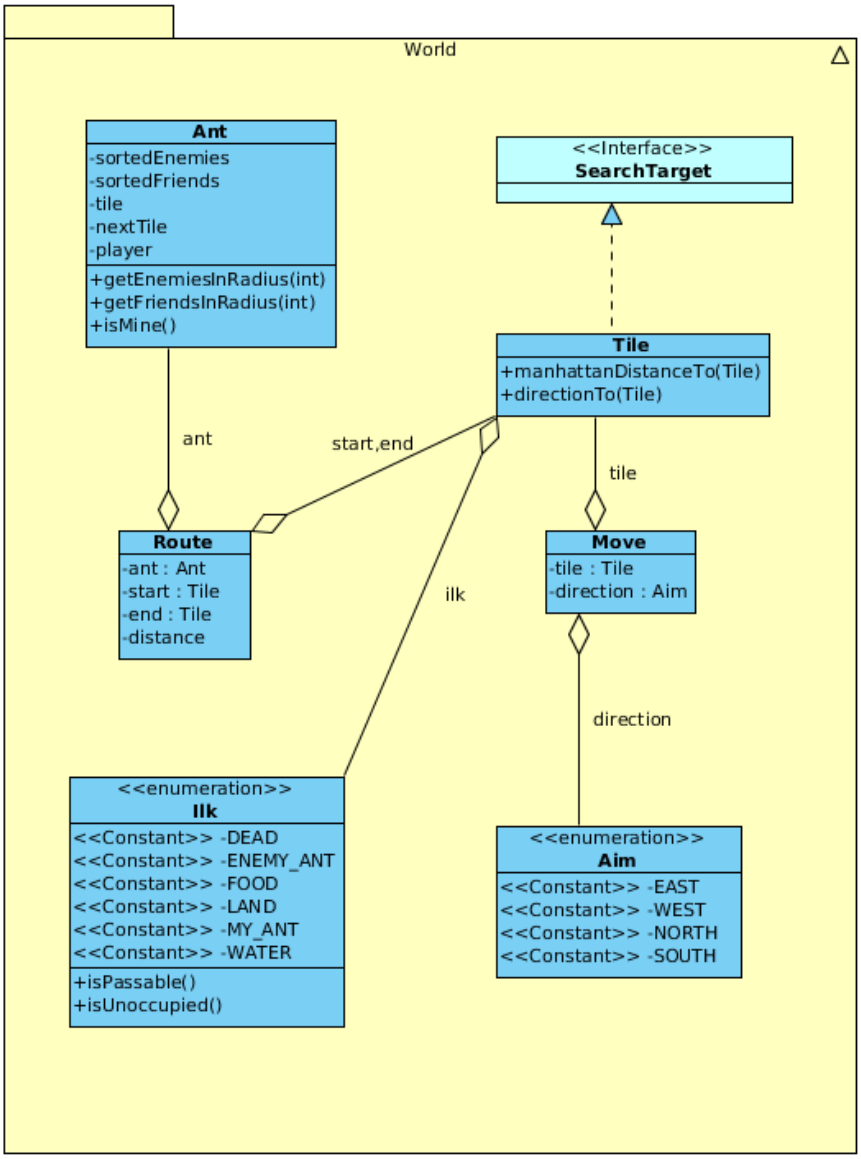

Abbildung 3.2.: Spiel-Elemente der Spielwelt (vereinfacht)

Tile

Das Tile repräsentiert eine Zelle des Spielfelds. Es implementiert das SearchTarget Interface (s. 3.1.3). Es bietet zudem Methoden für die einfache Distanzberechnung, sowie für das Bestimmen der Richtungen, in die ein anderes Tile liegt.

Route

Eine Route repräsentiert eine einfache Start-Ziel Verbindung. Sie hält für eine Ameise die Luftliniendistanz zu einem bestimmten Ziel-Feld fest.

Move

Ein Move entspricht einem Zug einer Ameise. Für ein bestimmtes Tile wird angegeben, in welche Richtung sich die Ameise bewegen soll.

Ilk

Ilk ist der Typ einer Zelle. Der Ilk einer Tile-Instanz gibt an, was sich gerade in der Zelle befindet. Dies kann eine Gelände-Typ sein, wenn die Zelle ansonsten leer ist, oder es kann eine Ameise, Nahrung, oder eine Hügel sein. Die Ilk-Enumeration bietet Hilfsmethoden, um festzustellen, ob eine Zelle passierbar oder besetzt ist.

Aim

Aim ist einfach eine Repräsentation einer Himmelsrichtung

3.1.3. Spiel-Elemente (Suche)

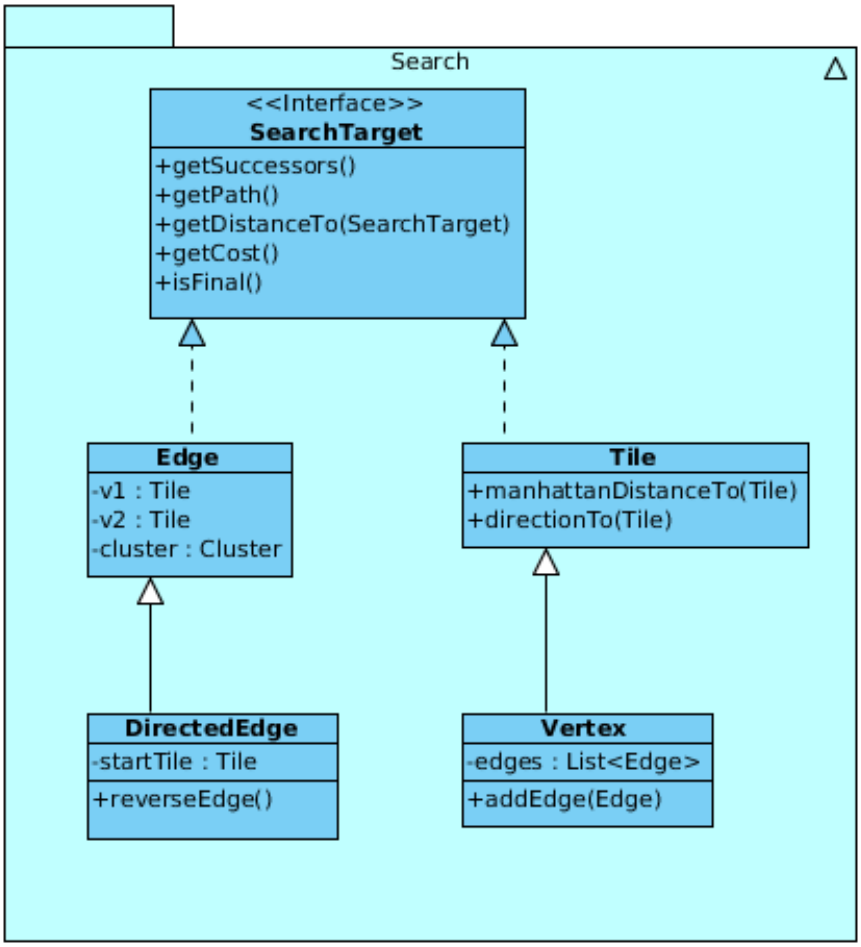

Abbildung 3.3.: Spiel-Elemente für die Suche (vereinfacht)

Abbildung 3.3 zeigt die wichtigsten Klassen, die für die Pfadsuche verwendet werden. Der Übersichtlichkeit wegen wurden nur die wichtigsten Attribute und Operationen in das Diagramm aufgenommen.

SearchTarget

Das SearchTarget ist ein Interface für Strukturen, die als Suchknoten in der Pfadsuche verwendet werden können. Es definiert die für die Suche nötigen Methoden, wie getSuccessors(), getCost(), oder getPath(). Implementierende Klassen sind Edge (repäsentiert eine Kante in einem Cluster) und Tile (repräsentiert eine Zelle des Spielfelds,

s. 3.1.2). Erweiterungen dieser Klassen sind DirectedEdge (eine gerichtete Kante) und Vertex (eine Zelle mit zugehörigen Kanten).

3.1.4. Tasks

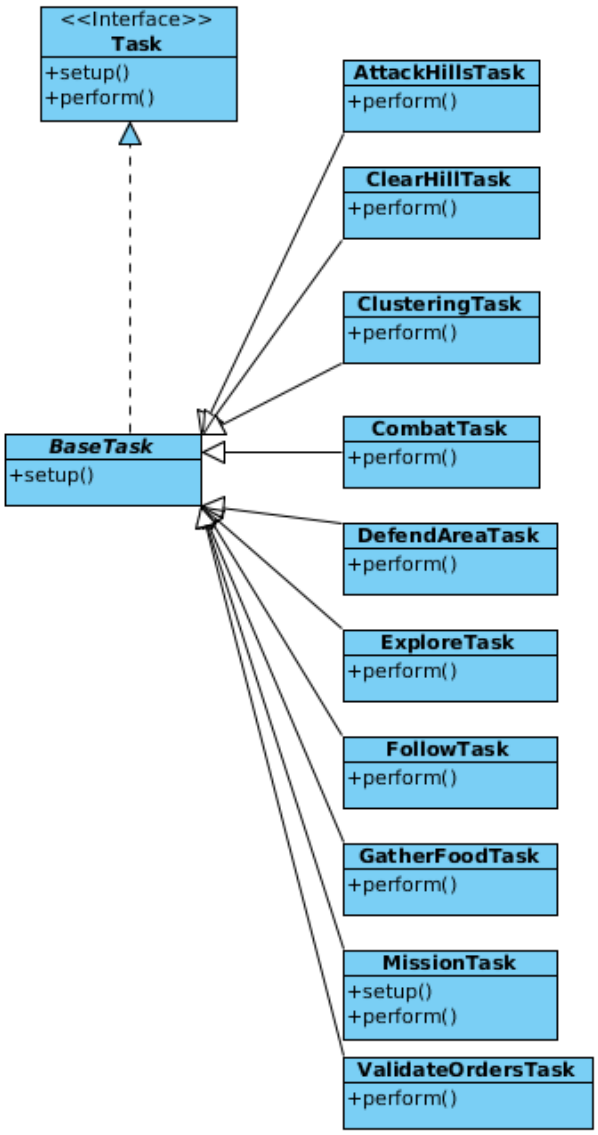

Abbildung 3.4.: Tasks

Zu Beginn des Projekts haben wir die wichtigsten Aufgaben einer Ameise identifiziert. Diese Aufgaben wurden als Tasks in eigenen Klassen implementiert. Das Interface Task[1] definiert eine setup()-Methode welche den Task initiiert, sowie eine perform()-Methode welche den Task ausführt. Im Programm werden die Tasks nach deren Wichtigkeit ausgeführt, was auch der nachfolgenden Reihenfolge entspricht. Jedem Task stehen nur die unbeschäftigten Ameisen zur Verfügung, d.h. jene welchen noch keine Aufgabe zugeteilt wurde.

MissionTasks

Dieser Task prüft alle akutellen Missionen auf deren Gültigkeit, beispielsweise ob die Ameise der Mission den letzten Zug überlebt hat und die Mission weiterführen kann. Falls gültig, wird der nächste Schritt der Mission ausgeführt.

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

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 (s. Abschnitt 3.3.1) oder falls dieser kein Pfad gefunden hat mit A* eine passierbare Route gesucht. Wenn ein Pfad existiert, kann mit der Ameise und dem Food-Tile eine GatherFoodMission erstellt werden, welche die Ameise zum Food-Tile führt. Zu jedem Food-Tile wird immer nur eine Ameise geschickt.

AttackHillsTask

Sobald gegnerische Ameisenhaufen sichtbar sind, sollen diese angegriffen werden. Falls dieser, wie bereits erwäht, zerstört wird, werden zwei Bonuspunkten gutgeschrieben. 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.

DefendAreaTask

Dieser Task wäre vogesehen um eine Region wie zum Beispiel der eigene Ameisenhügel zu schützen. Dieser Task wurde mit im Zuge dieser Arbeit nicht implementiert.

ExploreTask

Für alle noch unbeschäftigten Ameisen wird mittels ManhattanDistance der nächste Ort gesucht, der noch nicht sichtbar, also unerforscht ist. Falls ein Pfad mittels Pfadsuchalgorithmus gefunden wird, wird eine ExplorerMission (s. Abschnitt 3.1.5) erstellt. 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 einer nahe gelegenen, beschäftigten Ameise folgen, damit diese nicht alleine unterwegs ist.

ClearHillTask

Dieser Task bewegt alle Ameisen, welche neu aus unserem Hügel s̈chlüpfenv̈om Hügel weg. So werden nachfolgende Ameisen nicht durch diese blockiert.

ClusteringTask

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

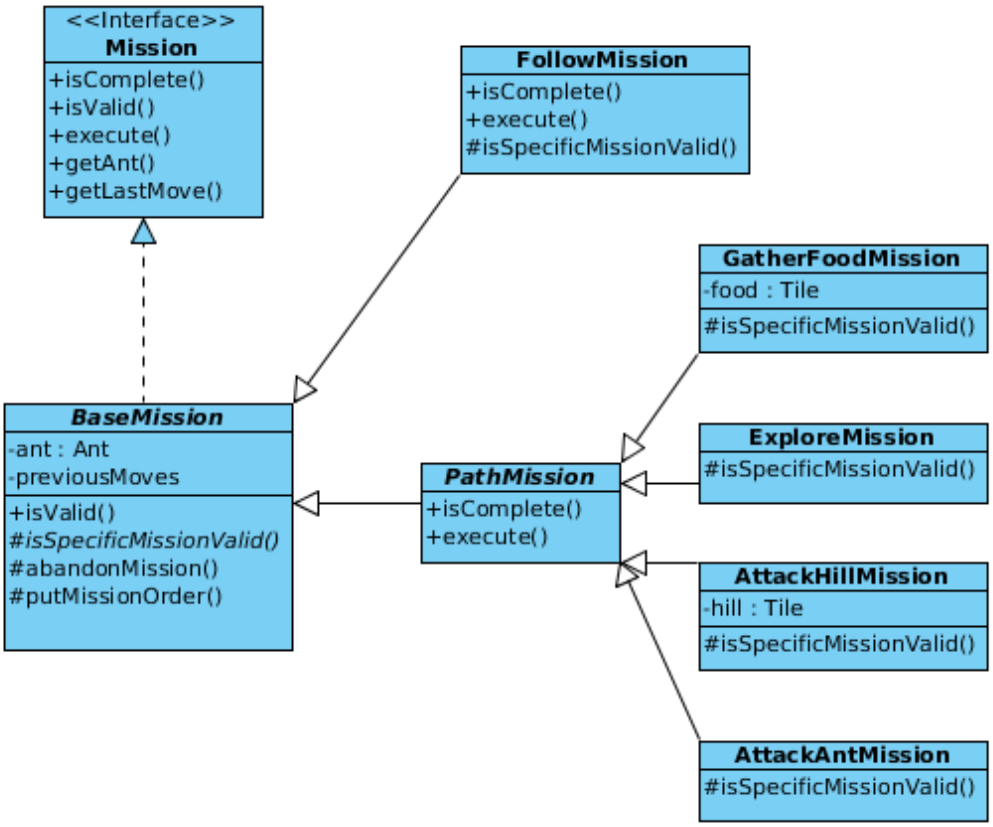

Abbildung 3.5.: Missionen

3.1.5. Missionen

Eine Mission dauert über meherer Spielzüge. Die meisten Missionen (GatherFoodMission, ExploreMission, AttackHillMission, AttackAntMission) sind Pfadmissionen[2] bei welchen die Ameise einem vorgegebenen Pfad, der bereits beim Erstellen der Mission berechnet wurde, folgt. Die FollowMission ist eine spezielle Mission, mit der eine Ameise einfach einer anderen Ameise hinterherläuft.

Eine Mission kann auch abgebrochen werden, wenn es keinen Sinn mehr macht, sie weiter zu verfolgen. 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. Abbildung 3.5 zeigt einen Überblick über die wichtigsten Missionen und ihre Hierarchie.

3.2. Bot

Als Basis für unsere Bot Implementation haben wir den Beispiel-Bot verwendet, der im Java-Starter-Package enthalten ist, das von der AI-Challenge-Website heruntergeladen werden kann. Dieser erbt von den Klassen AbstractSystemInputReader und AbstractSystemInputParser, die die Interaktion mit der Spiele-Engine über die System-Input/Output Streams kapseln. Für eine optimierte Lösung könnte der Bot auch angepasst werden, dass er selber auf die Streams zugreift; im Rahmen dieser Arbeit erschien uns das aber noch nicht nötig.

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

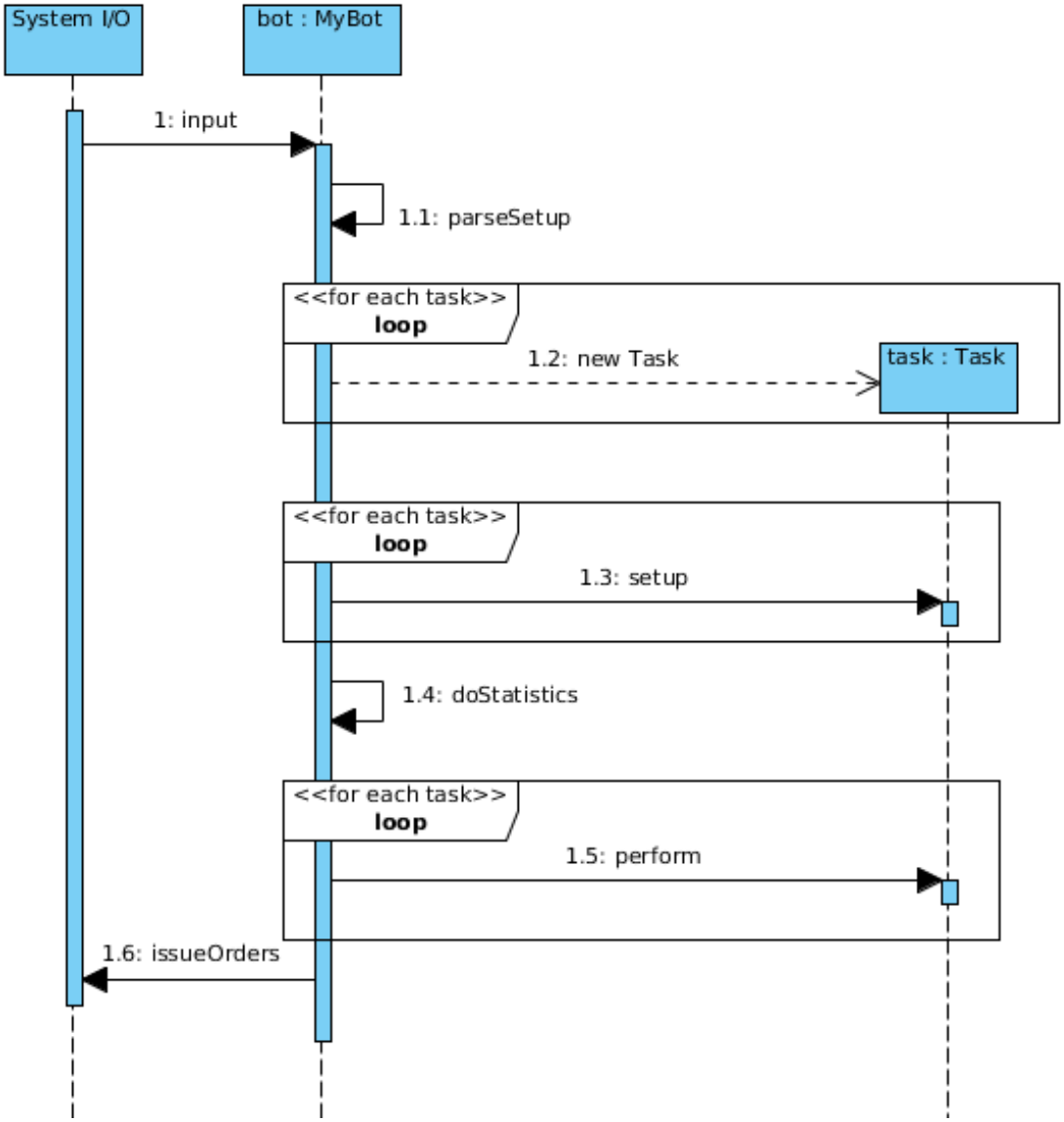

Abbildung 3.6.: Ablauf des ersten Zugs des Spiels

3.2.1. Ablauf eines Zugs

Abbildung 3.6 zeigt den Ablauf des ersten Zugs, während Abbildung 3.7 den Ablauf aller weiteren Züge zeigt.

Jeder Zug beginnt mit dem Einlesen des Inputs vom SystemInputStream. Wenn der Bot das Signal "READY"(1. Zug) oder "GO"(alle weiteren Züge) erhält, kann er den gesammelten Input verarbeiten (Methode parseSetup() resp. parseUpdate()). Danach wird die eigentliche Logik des Bots ausgeführt.

Im 1. Zug werden dabei Instanzen der Tasks erstellt. Abgesehen davon unterscheidet sich der 1. Zug von diesem Punkt an nicht mehr von allen nachfolgenden Zügen. Die Tasks werden vorbereitet (Aufruf der jeweiligen setup() Methode; danach werden einige statistische Werte aktualisiert und in jedem 10. Zug auch geloggt. Dann werden die Tasks in der definierten Reihenfolge aufgerufen. Hier wird der Löwenanteil der Zeit verbracht, denn die Tasks enthalten die eigentliche Logik unserer Ameisen.

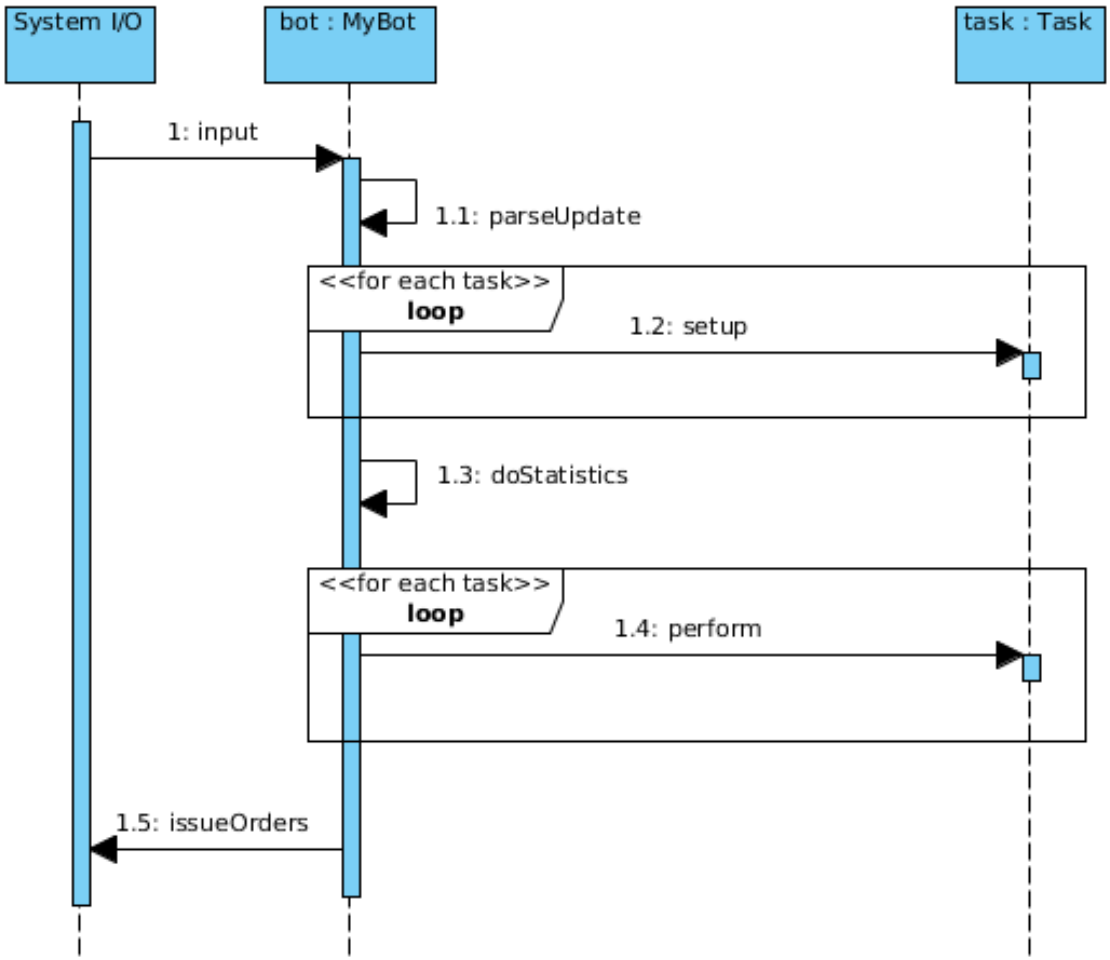

Abbildung 3.7.: Ablauf der weiteren Züge des Spiels

3.3. Pfadsuche

Wir haben drei mögliche Pfadalgorithmen in unserem Code eingebaut. Via PathFinder-Klasse 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 ein 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. f(x) besteht aus einem Teil g(x) welches die effektiven Kosten vom Startknoten zum aktuellen Knoten berechnet. Der andere Teil h(x) ist ein heuristischer Wert der die Pfadkosten welche bis zum Zielknoten approximiert. Dieser Wert muss die effektiven Kosten zum Ziel immer unterschätzen. Dies ist in unserem Spiel dadurch gegeben,

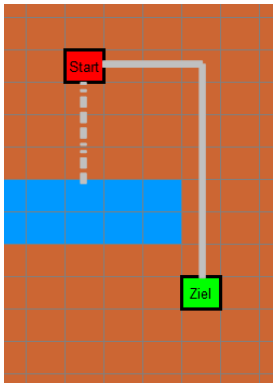

Abbildung 3.8.: Simple-Path Algorithmus

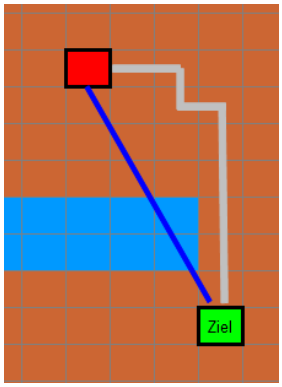

Abbildung 3.9.: Heuristische Kosten (blau), Effektive Kosten (grau)

dass sich die Ameisen nicht diagonal bewegen können, wir aber für den heuristischen Wert die Luftlinie zum Ziel verwenden. Die Pfadsuche wird immer bei dem Knoten fortgesetzt welcher die kleinsten Kosten f(x) hat.

Das Bild zeigt den effektiven Pfad (grau) vom expandierenden roten Knoten mit den minimalen Kosten von 10 Pixel. Die Luftlinie (blau) als heurstischer Wert hat aber nur eine Länge von 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. Der gleiche Code wir aber auch für die Pfadsuche mit dem Pfadnetz des HPA* verwendet.

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. Abhilfe 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 3.10 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 die auf einem sichtbaren Teil der Landkarte liegen, was zu Begin des Spiel nicht gegeben ist. Deshalb wird der ClusteringTask in jedem Spielzug aufgerufen, in der Hoffnung ein Cluster komplett verbinden zu können. Sobald eine beliebige Seite eines Clusters berechnet ist, wird diese Aussenkante im Cluster und dem anleigenden Nachbar gespeichert und nicht mehr neu berechnet.

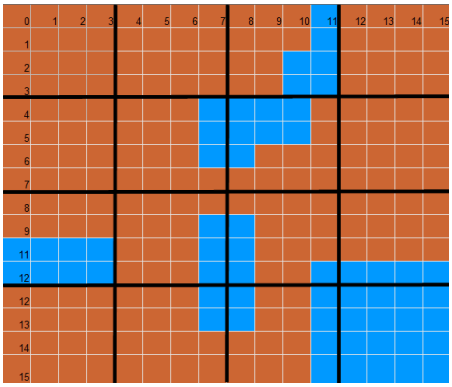

Abbildung 3.10.: Clustereinteilung auf der Landkarte. Clustergrösse 4x4, Landkarte 16x16

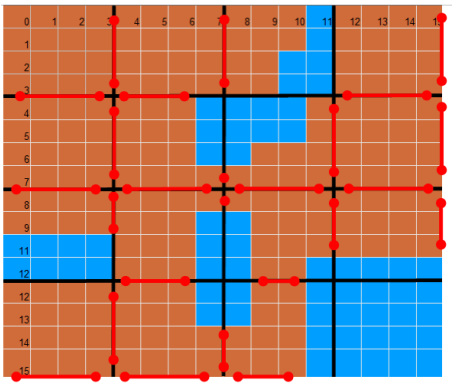

Abbildung 3.11.: Die Kanten jedes Clusters wurden berechnet

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 Gesamtkarte.
Nun wird ein Pfad vom Pixel (0,9) nach (14,9) gesucht. Zuerst wird eruiert in welchem Cluster sich das Start- bzw Zielpixel befindet. Danach wird in dem gefundenen 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())
```

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

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

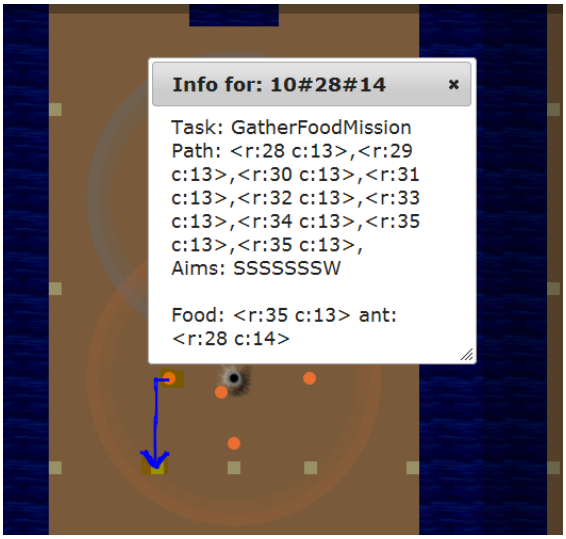

Abbildung 3.12.: Das Popup zeigt dei Aufgabe und den Pfad (blau), welcher die Ameise ablaufen wird.

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

Wir haben mit dieser Arbeit einen guten Einblick in die Programmierung von Bots erhalten. Uns ist bewusst geworden, dass viele Aspekte zusammenspielen müssen, damit sich die einzelen Ameisen intelligent verhalten. Da die Spielschnittstelle einfach gehalten ist und im Web gut dokumentiert ist kamen wir relativ schnell zu einem sichtbaren Resultat indem bereits erste Spiele verfolgt werden konnten. Mit einem sauberen Aufbau des Programmcodes konnten wir die Aufgaben einfach gliedern und haben einen guten Grundstein gelegt, falls wir dieses Projekt als Bachelorarbeit weiterverfolgen. Dank den eingesetzten Pfadsuchalgorithmen konnten wir die Rechenzeit verringern und haben so während einem Zug genug Zeit um weitere Berechnungen zu machen.

4.2. Herausforderungen

Beim Suchen eines Programmierfehlers musste das Logfile durchforscht werden, was sich als eine aufwendige und zeitraubende Arbeit herausstellte. Dadurch sind wir auch auf die Idee des Javascript Addon gekommen. Zu Beginn der Arbeit wurde die zur Verfügung stehende Zeit eines Zuges rasch aufgebraucht, da wir keinen schlauen Algorithmus für die Pfadsuche verwendeten, dies konnte mit A* und HPA* behoben werden.

4.3. Ziele für Bachelorarbeit

Im Projekt 2 haben wir uns vorallem auf die Pfadsuche konzentriert. Die Spieleentwicklung beinhaltet aber auch Strategie und Taktik. Dies konnte noch nicht angeschaut werden und wird sicher ein Teil der Bachelorarbeit sein. Die Ameisen sollten sich in einem Schwarm fortbewegen damit sie im Kampf stärker sind. Dies zu implementieren klingt interessant und trifft auch aufs Thema Künstliche Intelligenz gut zu. Weiter kann geprüft werden, ob ein supervised oder ein unsupervised Learning eingebaut werden kann.

A. Spielanleitung

Dieser Anhang beschreibt kurz, wie ein Spiel mit unserem Bot ausgeführt werden kann.

Das File Ants.zip enthält das Eclipse-Projekt mit dem gesamten Source-Code unserer Implementation, und der offiziellen Spiel-Engine. Zum einfachen Ausführen eines Spiels haben wir ein ANT-Buildfile (build.xml) erstellt. Dieses definiert 3 Targets, mit denen ein Spiel mit jeweils unterschiedlichen Parametern gestartet werden kann.

1. Das Target `testBot` ist lediglich zum einfachen Testen eines Bots sinnvoll und entspricht dem Spiel, das verwendet wird, um Bots, die auf der Website hochgeladen werden, zu testen.
2. Das Target `runTutorial` führt ein Spiel mit den Parametern aus, die im Tutorial auf der Website zur Erklärung der Spielmechanik verwendet werden.
3. Das Target `maze` führt ein Spiel auf einer komplexeren und grösseren, labyrinthartigen Karte aus und ist das interessanteste von den 3.

Im Unterordner tools befindet sich die in Python implementierte Spiel-Engine. Unter tools/maps liegen noch weitere vordefinierte Umgebungen, und unter tools/mapgen liegen verschieden Map-Generatoren, die zur Erzeugung beliebiger weiterer Karten verwendet werden können.

Im Unterordner tools/sample_bots befinden sich einige einfache Beispiel-Bots, gegen die man spielen kann. Viele der Teilnehmer haben zudem ihren Quellcode auf dem Internet publiziert, an möglichen Gegner besteht also auch kein Mangel.