1 Dokumentation

Der relevante Sourcecode ist in drei große Packages gegliedert:

- **ui**: Enthält die Klassen, um gegen die Agenten zu spielen, bzw. um die Agenten gegeneinander spielen zu lassen
- **game**: Enthält alle Klassen, welche für die Implementierung des Spiels Lost Cities relevant sind
- **ai**: Enthält alle Implementierungen der vorgestellten KIs

1.1 Wie man den Code ausführt

1.1.1 Selbst gegen eine KI antreten

Wenn man gegen eine KI spielen will, muss man die Main-Methode der Klasse *GUI* aufrufen, die sich im Package *ui* befindet. Die GUI sollte problemlos starten, wenn sich der Ordner *images* im gleichen Verzeichnis wie die Klasse befindet.

Die Auswahl der KI erfolgt noch im Code. Standardmäßig ist die HP-KI ausgewählt. Will man einen anderen Gegner haben, muss man die Zeile 80 verändern. Achtung: Nicht den ersten Parameter (Player-Index) verändern. Der zweite Parameter (Bedenkzeit der KI) kann verändert werden.

Bei der Ausführung hat man immer den ersten Zug. Unten sieht man die eigenen Handkarten. Durch einen Klick selektiert man eine. Anschließend muss man auf das Ziel klicken, worauf man die Karte spielen will. Die untersten fünf Plätze entsprechen den eigenen Expeditionen. Die mittleren Plätze stellen die Ablagestapel und den Nachziehstapel dar. Die obersten Plätze sind die Expeditionen der KI. Nachdem Ziehen einer Karte, beginnt die KI ihren Zug. Wenn sie fertig ist, wird der Zug in der Konsole ausgegeben. Zu Ende der Partie erfolgt eine Meldung mit den Punkteständen.

1.1.2 Zwei KIs gegeneinander spielen lassen

Wenn man zwei KIs gegeneinander antreten lassen will, muss man die Main-Methode der Klasse *RunningGame* aufrufen, die sich im Package *ui* befindet.

Erneut erfolgt die Auswahl der KIs im Code. In Zeile 55 und 56 kann man die KI ändern (erneut nicht den ersten Parameter ändern).

Wichtig für die Ausführung sind drei Variablen:

- **AMOUNT_GAMES**: Die Anzahl der Spiele, die zwischen den Agenten ausgeführt werden sollen.

- **textActive**: Wenn dieser Wert auf true gesetzt ist, werden die Züge beider Agenten ausgegeben. Außerdem wird vor jedem Zug des ersten Agenten der Spielzustand aus seiner Sicht ausgegeben.
- **analysisActive**: Wenn man diesen Wert auf true setzt, werden zu Ende aller Spiele ein paar Daten (Siegesverteilung, Punktedurchschnitt, etc.) ausgegeben. Außerdem werden die Punktestände aller Spiele in die Datei *results.csv* geschrieben.

1.2 Die Implementierungen

Alle Agenten sind Unterklasse der Klasse *ArtificialIntelligence*. Zur Initialisierung des jeweiligen Agenten werden folgende Klassen benötigt:

- **ZB-KI**: RandomAI
- **VZB-KI**: ImprovedRandomAI
- **LP-KI**: InformationSetAILight
- **HP-KI**: InformationSetAIHard
- **HPP-KI**: InformationSetAIHard2

Die Konstruktoren der zufallsbasierten KIs benötigen nur einen Index (1 oder 2) als Parameter. Die Konstruktoren der MCTS-Agenten benötigen einen zweiten Parameter, in dem die Bedenkzeit in Millisekunden angegeben ist.

Jeder Agent hat die Methoden *playCard* und *takeCard*, die zum Spielen der Züge genutzt werden. Außerdem gibt es noch die Methoden *receiveOpponentPlayMove* und *receiveOpponentTakeMove* um Informationen über die Züge der Gegenseite zu erhalten.

Der Aufbau eines SO-ISMCTS-Agenten besteht aus folgenden Unterklassen:

- **InformationSetAI**: Hier wird die Erstellung und Ausführung der anderen relevanten Klassen ausgeführt.
- **MCTSAgent**: Hier erfolgt die Ausführung der MCTS.
- **StateCopy**: Hier wird der Spielzustand kopiert, eine Determinisierung angefertigt und die Simulationen ausgeführt.

Relevant sind noch die Klassen *MCTSThread*, um die Ausführung des Agenten für die Parallelisierung zu gewährleisten, und *Node*, um die Knoten des Suchbaums zu erstellen.

Die Anzahl der Threads kann in der Klasse *InformationSetAI* geändert werden. Den Wert der Konstante c findet man in der Klasse *Node*. Der Wert N für die Technik Best-Of-N ist in der Klasse *StateCopyHard2* anzufinden.

1.3 Die Domäne

Die Implementierung des Spiels *Lost Cities* erfolgt hauptsächlich in der Klasse *Game*. Für die Spielobjekte sind die Klassen *Card* und *MoveSet* relevant, wobei der Zug in die Klassen *PlayMove* und *TakeMove* aufgeteilt ist. Außerdem kann der Spielzustand in der Klasse *GameState* gespeichert werden, wobei man bei der Erstellung aus dem *Game*-Objekt angeben muss, aus welcher Sicht der Spielzustand erstellt werden soll.