Universität Rostock
Ludwig Krause: 209200612
Willem Schlüter: 220201009
Sadegh Hajimirzamohammadi: 221202356

Sommersemester 2024
Künstliche Intelligenz (Smart Computing)
07.07.2024

Programmier-Projekt Battlesnake

Aufgabe 1: Bot als UML Diagramm

Aufgabe 2: Welche Art von Agent?

Welche Art von Agent stellt der Bot dar ?

In der Vorlesung haben wir eine Reihe an Agenten kennengelernt. Für die Ausarbeitung des Snake Bots wählten wir das Konzept des „zielbasierten Agenten“, das bedeutet der Agent verfolgt ein klar definiertes Ziel, was er unter gewissen Beschränkungen versucht zu erreichen. Dabei greift er auf Techniken des Suchens und Planens zurück.

Das Ziel unseres Agenten unterscheidet sich je nach Situation, in der er sich befindet. Dabei ist das entscheidende Kriterium für die Situation, das Verhältnis der Länge von unserer Schlange und der Schlange der Gegner. Ist unsere Schlange kürzer bzw. genau so lang wie die anderen Schlangen, strebt die Snake nach dem „Futter“und sucht dieses gezielt. Sobald eine Länge erreicht ist mit der man länger ist als die anderen, wechselt die Schlange in einen anderen „Modus“und zielt nicht mehr auf das Futter ab, sondern sucht jetzt die Köpfe der anderen Schlangen um diese gezielt auszuschalten.

Benutzte Konzepte aus der Vorlesung

Die Schlange sucht über den A*-Stern Algorithmus, dieser wertet Knoten aus indem er g(n), die Kosten zum erreichen des Zieles und h(n), die Kosten um von dem Knoten zum Ziel zu gelangen addiert. Somit ergibt sich dann f(n), was die Kosten für die billigste Lösung durch n entspricht.

$$f(n) = g(n) + h(n) \tag{1}$$

Für unsere Snake bedeutet sie sucht immer den kürzesten Weg um Hindernisse herum zum gewählten Zielpunkt, also je nach „Modus“den Weg zum Futter, oder zu dem Kopf eines Gegenspielers. Deshalb fiel unsere Wahl auf den A*-Stern er sucht gezielt den kürzesten weg und ist somit optimal für die Anwendung in Battlesnake.

In unserer Implementierung des A*-Algorithmus werden Begrenzungen eingebaut, die die Schlange davon abhält, außerhalb des Spielfeldes zu laufen, in die Körper der anderen Schlangen oder der eigenen Schlange. Das selbe gilt für die Köpfe der anderen Schlangen, wenn die feindliche Schlange noch länger und damit eine Gefahr ist, dann wird anhand der Kopf Position der anderen Schlange, gefährliche Felder erkannt und vermieden. Im Futtermodus werden die Köpfe anderer Schlangen nur als Gefahr definiert, wenn sie gleich lang oder länger sind als unsere Schlange. Ist die gegnerische Schlange kleiner, wird der Kopf nicht als Gefahr identifiziert und es kann zum zufälligen Fressen der anderen Schlange kommen. Im Jagdmodus (wenn unsere Schlange 2 Stücken länger ist als alle anderen), werden die Köpfe als Ziel gesucht.

Darum diese Konzepte

Wir haben uns für einen Zielbasieren Agenten entschieden, da dieser es erlaubt immer das aktuell relevante Ziel zu erkennen, wodurch die Schlange optimiert darauf hinsteuern kann. Ein weiter Grund ist die Flexibilität, die dadurch gegeben ist. Durch das wechselen „Spiel-Modi“passt sich die Schlange immer der aktuellen Spielsituation an.

Für den A* haben wir uns entschieden, da dieser effizient den kürzesten Weg um Hindernisse herum zum Ziel ermitteln kann. Während andere Such-Algorithmen (zB. Breiten/Tiefensuche) ineffizient arbeiten würden. So findet die Schlange mit dem A* schnell und gezielt Futter/Köpfe und kann so einen entscheidenden Spielvorteil erlangen. Zudem ist A* sehr dynamisch und kann sich

immer der gegebenen Situation anpassen. Für „Kopf-zu-Kopf Vermeidung “und der Kollisionsvermeidung mit den anderen Schlangen haben wir uns entschieden um die Überlebensfähigkeit der Schlange zu vergrößern.

Aufgabe 3: Weitere Uberlegungen

1. Repräsentation der Map inklusive Wänden, Schlangen und Futter. Um eine einheitliche Grundlage für Bewegungsentscheidungen und Algorithmen zu dienen, wurde ein Numpy Array aus den Daten des *game_state* Objekts angelegt.

```
[['w_' 'w_' 'w_' 'w_' 'w_' 'w_' 'w_' 'w_' 'w_' 'w_' 'w_' 'w_' 'w_']
 ['w_' '__' '__' '__' '__' '__' '__' '__' '__' '__' '__' '__' 'w_']
 ['w_' '__' '__' '__' '__' '__' '__' '__' '__' '__' '__' '__' 'w_']
 ['w_' '__' '__' '__' '__' '__' '__' '__' '__' '__' '__' '__' 'w_']
 ['w_' '__' '__' '__' '__' '__' 'b1' '__' '__' '__' '__' '__' 'w_']
 ['w_' '__' '__' '__' '__' '__' 'b1' '__' '__' '__' '__' '__' 'w_']
 ['w_' '__' '__' '__' '__' '__' 'b1' '__' '__' '__' '__' '__' 'w_']
 ['w_' '__' '__' '__' '__' 'h1' 'b1' '__' '__' '__' '__' '__' 'w_']
 ['w_' '__' '__' '__' 'f_' '__' '__' '__' '__' '__' '__' '__' 'w_']
 ['w_' '__' '__' '__' '__' '__' '__' '__' '__' '__' '__' '__' 'w_']
 ['w_' '__' '__' '__' '__' '__' '**' 'b0' 'b0' '__' '__' '__' 'w_']
 ['w_' '__' '__' '__' '__' '__' '__' 'b0' 'b0' '__' '__' '__' 'w_']
 ['w_' 'w_' 'w_' 'w_' 'w_' 'w_' 'w_' 'w_' 'w_' 'w_' 'w_' 'w_' 'w_']]
```

Listing 1: Board Arrayrepräsentation

   Diese Map wird bei jedem Aufruf des Agentenserver aus dem *game_state* Objekts angelegt.

2. Ausgehend vom A*-Algorithmus wird eine hierarchische Prüfung verschiedener Bedingungen durchgeführt.
   Jagmodus: (wenn unsere Schlange 2 größer ist als alle anderen) → A* zum nächsten Kopf
   Findet A* keinen Pfad oder nicht sicher: → Zufälliger sicherer Zug
   Wird gar kein sicher Zug gefunden: → {”move”: ”up”}
   Futtermodus: (sonst)
   Existiert kein Futter: → Zufälliger sicherer Zug
   Exisitiert Futter: → A* zum nächsten Futter
   Head-Kollision möglich und Gegner kleiner: → A* zum nächsten Futter
   Findet A* keinen Pfad oder Head-Kollision nicht sicher: → Zufälliger sicherer Zug
   Wird gar kein sicher Zug gefunden: → {”move”: ”up”}

Aufgabe 4: Umgebung von Battlesnake

Die Umgebung von BattleSnake-spiel ist :
vollständig observierbar, Bei jedem Schritt (Turn) erhält jeder Agent vollständige Informationen über den Zustand des Spielfelds, einschließlich der Position aller Schlangen und des Essens.
nicht-deterministisch, die Entscheidung für nächsten Schritt hängt von die Reaktion der anderen Agenten und auch das Auftauchen von neuem “Food” ist zufällig.
sequentiell, da die Entscheidungen in einem Zug die zukünftigen Zustände beeinflussen.
statisch, die Umgebung bleibt in jeder “Turn” (während des Aktionsauswahlprozesses) unverändert
diskret, die Zustände sind diskret
multi-Agent: weil es gibt andere Mitspieler und sie stehen im Wettbewerb