Suchalgorithmen Ein Agent muss für ein bestimmtes Ziel die richtige Wahl von Aktion treffen und vorausplanen, eine Sequenz an Aktionen erstellen. Der Prozess für die Findung der Sequenz an Aktionen wird als Suche bezeichnet und durch Suchalgorithmen gefunden. Im Bereich der Suchalgorithmen wird zwischen informierten und uniformierten Algorithmen unterschieden. Informierte Algorithmen können dabei die Distanz zum Ziel schätzen, während uniformierte eine solche Schätzung nicht durchführen können. Sollte der Agent keine Informationen über seine Umwelt haben, so muss dierser mit Zufällen arbeiten. Mit Informationen über die Umwelt hat dieser aber die Möglichkeit den Weg zum Ziel über Suchalgorithmen zu finden.

1 Suchproblem

Ein Suchproblem wird definiert über einen Satz an möglichen Zuständen (*states*), einen Ausgangszustand (*initial state*), Zielzustände (*goal states*), Aktionen (*actions*), Übergangsmodel (*transition model*) und Aktion-Kosten (*action costs*).

Die Zustände beschreiben dabei die Umwelt und ein Ausgangszustand s den Zustand des Agenten in dem der Agent startet.

$$s = \{AtCover, BulletsAvailable, PlayerAlive\}$$

Ein Agent bekommt ein oder mehrere Ziele die in Zielzuständen z beschrieben werden.

$$z = \{\neg PlayerAlive\}$$

Die Aktionen die ein Agent besitzt können in bestimmten Zuständen *ACTIONS(s)* ausgeführt werden.

$$ACTIONS(BulletsAvailable) = \{Shoot\}$$
$$ACTIONS(\neg BulletsAvailable) = \{Reload\}$$

Ein Übergangsmodell *TRANSITION(s,a)* = *s** beschreibt dabei den resultierenden Zustand *s** der durch Aktionen a im derzeitigen Zustand s resultiert.

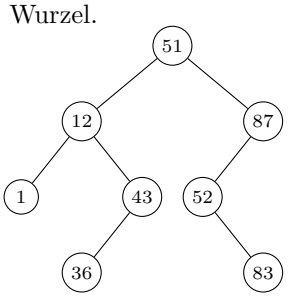
$$TRANSITIONS(BulletsAvailable, Shoot) = \neg PlayerAlive$$

Durch eine Aktion-Kosten Funktion *ACTION-COST(s,a,s*)* erhalten wir die Kosten einer Aktion$a$, welche in einem Zustand s ausgeführt werden und in einen neuen Zustand *s** resultieren.

Die Lösung für ein Suchproblem ist dabei ist die Sequenz an Aktionen, also der Plan der vom Ausgangszustand zum Zielzustand führen soll. Der gewählte Pfad sollte die geringsten Kosten von allen möglichen Lösungen haben und somit eine optimale Lösung darstellen. Der Satz an möglichen Zuständen kann dabei als Graph dargestellt werden. Dabei werden die Kanten als Aktionen und die Knoten als Zustände dargestellt.

2 Suchalgorithmen

Das Suchproblem soll mit seinen Informationen durch einen Suchalgorithmus gelöst werden. Ein Suchalgorithmus erzeugt für seine Suche einen Suchbaum aus dem Suchproblem. Wie beim Graphen sind in Suchbäumen Knoten Zustände und Kanten Aktionen, welche zu weiteren Zuständen führen. Der Wurzelknoten ist dabei der Ausgangszustand des Agenten. Der Suchbaum beschreibt dabei Pfade zu Knoten und zum Ziel und kann mehrere Pfade zu einem gegebenen Zustand haben, aber jeder Knoten im Baum hat einen eindeutigen Pfad zurück zur Wurzel.



3 A* Algorithmus

Gehört zu den heuristischen Suchverfahren. Wird oft für Routenplaner benutzt. So benutzt Godot 4.3 den A* Algorithmus für die Navigation. GOAP benutzt A* für das Suchen eines optimalen Pfades. A* ist ein vollständiger Algorithmus, er findet einen Pfad zur Lösung, wenn einer vorhanden ist. Die Kosten-Optimalität richtet sich nach einer Heuristik.Mit jeder Erweiterung des Pfades (n zu n*) steigen oder bleiben die g- Kosten gleich, was an den positiven Aktion-Kosten der Knoten liegt. g(n) + h(n) =¿ g(n) + c(n,a,n‘) + h(n‘).

3.1 A* Heuristik

Eine Heuristik hat die Eigenschaften der Zulässigkeit und Konsistenz. Bei einer zulässigen Heuristik werden Kosten nue überschätzt. Also im Intervall [0,k] (k: tatsächliche Kosten). Eine zunzulässige Heuristik kann zu nicht Kosten-Optimalen Pfaden führen. Eine konsistente Heuristik muss zulässig sein und die Dreiecksungleichung erfüllen. So ist die Abschätzung durch Luftlinie eine konsistente Heuristik. Die Dreiecksungleichung stellt die Regel, dass eine Seite nicht länger sein darf als die beiden anderen Seiten zusammen. Bezüglich Heuristik stellt sich daraus die Metrik: h(n) ¡= c(n,a,n‘) + h(n‘) c(n,a,n‘): Aktion-kosten von n bis n‘ h(n‘): die Heuristik von n‘ Sollte eine Heuristik konsistent sein, so sind die besuchten Zustände optimal und müssen weder in eine offene noch in eine geschlossene Liste. Mit einer inkonsistenten Heuristik können Pfade auftreten, welche denselben Zustand erreichen. Somit können mehrere Pfade mit unterschiedlichen Kosten aber selben Zustand in der offenen Liste auftreten,

was zur höheren Zeit und Speicherkosten führt. Es gibt A* Implementationen, die dieses Problem berücksichtigen.

$$Voraussetzung : f(Gs) = g(h) + h(n); f(n) \quad = g(n) + h(n); h(n) \geq h * (n) \tag{1}$$

$$Behauptung : f(Gs) > f(n) \tag{2}$$