Der Tank-Algorithmus

Intensiv hat uns die Frage beschäftigt, wie die KI am günstigsten tanken fahren kann. Hierbei handelt es sich um den wichtigsten und gleichzeitig auch kompliziertesten Teil der KI.

Es gibt mehrere Möglichkeiten, die wir auch alle im Laufe unserer Programmierung ausprobiert haben:

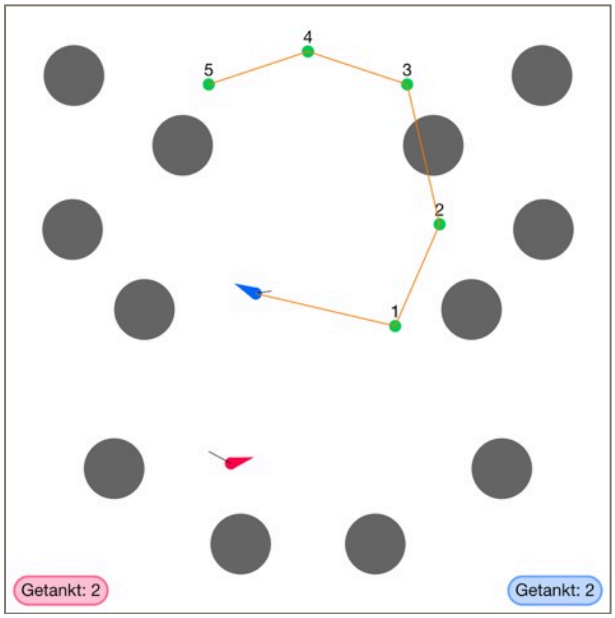
1. **Die einfachste Methode:** Unsere KI findet die dichteste Tankstelle (Luftlinie) und fährt sie an. Sobald diese eingesammelt wurde, wir die nächst-dichteste Tankstelle angefahren und so weiter.
2. **Die etwas kompliziertere Methode:** Unsere KI bewertet alle Tankstellen nach mehreren Kriterien und fährt dann die günstigste an:
    1. Entfernung (Luftlinie)
    2. Liegt eine Mine im Weg?
    3. Wie nahe ist der Gegner?
    4. Liegen noch weitere Tankstellen in einem gewissen Winkel hinter dieser Tanke? Wenn ja, wie viele?
3. **Die beste / komplexeste Methode:** Unsere KI durchläuft alle möglichen Wege, fünf Tankstellen anzufahren und bewertet diese Wege nach folgenden Kriterien:
    1. Entfernung (Luftlinie)
    2. Notwendige Geschwindigkeitsänderung (je größer der Winkel zur nächsten Tankstelle, desto mehr Strafpunkte)
    3. Liegt eine Mine im Weg?

Hierbei ist zu beachten, dass ein recht großer Rechenaufwand notwendig ist, da es bei fünf Tankstellen schon 120 mögliche Reihenfolgen gibt, diese anzufahren. Anfangs hatten wir noch mit allen 9 Tankstellen (362.880 Möglichkeiten) gerechnet, was allerdings strategisch ungünstig ist (siehe weiter unten).

Für die Berechnung der Tankstellen benutzen wir eine Abwandlung des sogenannten **Minimax-Algorithmus**. Eine detailliertere Beschreibung befindet sich in der Funktionsdokumentation.

Nachdem alle Wege bewertet wurden, gibt unser Programm eine Liste mit den Nummern der Tankstellen aus, die wir am besten in dieser Reihenfolge anfahren.

Der besondere Clou ist, immer nur den besten Weg für die Anzahl Tankstellen, die wir noch einsammeln müssen, zu suchen. Liegen beispielsweise drei Tankstellen auf einem Haufen und wir benötigen nur noch drei, dann ist es günstiger zu dieser Häufung Tankstellen zu fahren, als den Weg rechts herum (Abb. X). Unser Algorithmus würde aber als besten Weg für fünf Tankstellen den ersten wählen, da es für fünf Tankstellen tatsächlich günstiger wäre, dort lang zu fahren.



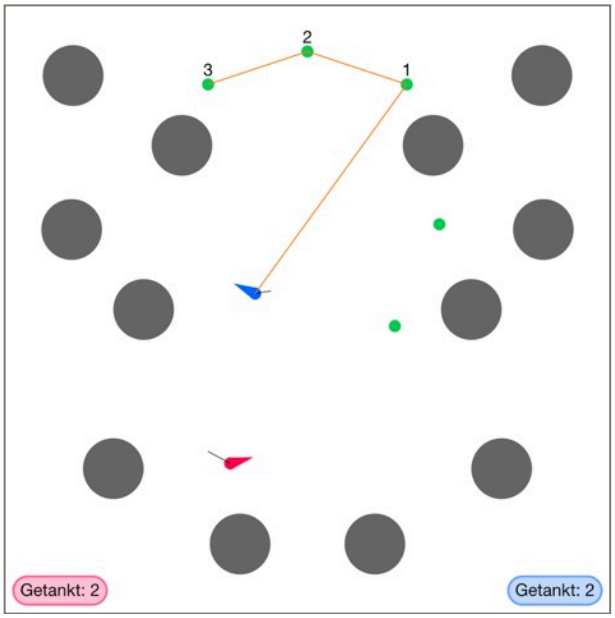


Abb. X

Dieser „Tank-Pfad-Findungs-Algorithmus“ wird zu folgenden Zeitpunkten erneut ausgeführt:

- Bei Spielbeginn
- Wir schalten vom Angriffs- oder Verteidigungsmodus zurück in den Tankmodus
- Eine Tankstelle verschwindet (wird eingesammelt, egal, von wem)
- Der *competitionMode* wird beendet
- *ignoreTanke* wird gesetzt, d.h. wir ignorieren eine Tankstelle, da der Gegner diese bald einsammeln wird

Damit ist der wichtigste Teil des Tankprozesses fertig. Wir benutzen den *Pathfinder*, um den tatsächlichen Pfad zwischen den Tankstellen zu berechnen und geben die Wegpunkte in die *waypointList*. Den Rest erledigt *calculateBES()*.

doesEnemyGetTanke()

Diese Funktion wird während des Tankens in jedem Rechenschritt aufgerufen. Sie besteht inhaltlich aus mehreren Teilen.

ignoreTanke

Häufig kommt es vor, dass der Gegner eine Tankstelle vor uns erreicht, die auf unserer *p_TankList* steht. Wie finden wir heraus, welche Tankstelle das ist? Wir berechnen bei jedem Rechendurchgang die Zeit, die der Gegner zu allen Tankstellen benötigt und suchen die geringste, die auch noch unter einem bestimmten Threshold liegen muss (*constIgnoreTankeTime*). Wenn es sich nicht um die erste Tankstelle handelt (der Gegner also nicht auf die Tankstelle zufährt, auf die wir auch gerade zufahren), und keine Mine zwischen Gegner und Tankstelle liegt, wird diese von uns ignoriert. Dazu schreiben wir den Index dieser Tankstelle in die Variable *p_ignoreTanke* und berechnen die Reihenfolge der Tankstellen neu, wobei die in *p_ignoreTanke* gespeicherte Tankstelle ignoriert wird.

keepFirstTanke

Sollte es vorkommen, dass sich durch die Neuberechnung die erste Tankstelle ändert, unser Spaceball also eine neue Tankstelle zuerst anfahren soll, dann kann dies dazu führen, dass wir eine Tankstelle knapp verfehlen und wertvolle Zeit verlieren. Zwar mag es sein, dass die neue Reihenfolge zeitlich günstiger ist, allerdings wird der Gegner nicht in die Überlegung mit einbezogen. In der Zeit, die wir benötigen, um unsere „neue“ erste Tankstelle einzusammeln, hat der Gegner oft schon eine andere Tankstelle eingesammelt und unser Algorithmus entscheidet sich wieder um. Dann sammeln wir wieder keine Tankstelle ein und verlieren wertvolle Zeit.

Daher gibt es nach dem Ausführen von *CreatePathAllTanken()* eine Überprüfung, ob sich die erste Tankstelle in unserer *p_TankList* geändert hat. Ist dies der Fall und wir befinden uns zeitlich nahe an dieser Tankstelle, wird diese in der *p_waypointList* an erster Stelle erhalten.

compMode

Sollten wir merken, dass der Gegner dieselbe Tankstelle anfährt, wie wir, berechnen wir zuerst, ob wir diese etwa gleichzeitig erreichen. Ist dies der Fall, wechselt unsere KI in den sogenannten „competition Mode“. Dafür wird die gesamte *p_waypointList* gelöscht und zwei neue Wegpunkte eingesetzt:

1. Die Position der Tankstelle, um die wir konkurrieren
2. Ein Punkt hinter der Tankstelle in einer Flucht mit dieser und unserem Spaceball

Der zweite Punkt wird mithilfe der Funktion *getAccPos()* berechnet. Dadurch hält unser Spaceball mit voller Beschleunigung auf die Tankstelle zu und wir erhöhen unsere Chance, diese vor dem Gegner zu erreichen, enorm. Oftmals ist dies spielentscheidend, da der Unterlegene meist sofort getroffen wird und das Spiel verliert.

Aufgrund gewisser Toleranzen in der Berechnung kommt es vor, dass der Gegner die Tankstelle vor uns erreicht. Wir versuchen, dies rechtzeitig zu merken und dann mit einer Vollbremsung die Kollision zu vermeiden. Dies klappt leider nicht immer.