A*-Implementierung mit Boot

Anton Mrosek, Konstantin Radke, TINF-18D

VERWENDETE BIBLIOTHEKEN

Als Programmiersprache wurde Python verwendet. Der Programmcode verwendet fast ausschließlich Standardbibliotheken. Abgesehen davon wurde nur Pygame für die grafische Ausgabe verwendet.

PROGRAMMSTRUKTUR

Für die Implementierung von A* mit Bootnutzung nach Aufgabenstellung wurden verschiedene Klassen erstellt:

- Terrain
    - Enthält einen Index, der das Terrain in der CSV-Datei identifiziert
    - Enthält die Kosten eines Feldes mit diesem Terrain
- Field
    - Entspricht einem Einzelfeld im Spielfeld
    - Enthält Referenzen zu den bis zu vier Nachbarfeldern
    - Enthält Terrain-Objekt
    - Kennt die eigene Position auf dem Spielfeld (Liste aus x und y-Koordinaten)
    - Speichert bei Bedarf das vorhergehende Feld im Pfad
    - Speichert die Wegkosten ab dem Startpunkt
    - Enthält einen Boolean zur Überprüfung, ob das Feld bereits besucht wurde (also in der Open- oder Closed-List geführt wird)
- Area
    - Enthält Open- und Closed-List
    - Enthält das Spielfeld als zweidimensionale Liste aus Field-Objekten
    - Variablen für die grafische Ausgabe

Die Area-Klasse beinhaltet gleichzeitig die Methoden zum Einlesen einer CSV-Datei, die Erstellung der Grafik sowie den A*-Algorithmus selbst. Zur Referenz sind im Folgenden die in der Vorlesung besprochenen Funktionen denen aus dem Python-Programm zugeordnet:

g(x) → `Field.cost_from_start`

h(x) → `Field.get_estimated_cost`

f(x) → `Field.get_total_cost`

Um die Funktionalität des Bootes zu implementieren, wurden einige Methoden ergänzt. So muss der Algorithmus sobald er auf ein Feld mit Wasser trifft prüfen, ob das Boot noch verfügbar ist. Dies geschieht über eine rekursive Methode, die den bisherigen Pfad rückwärts

abläuft und einen Boolean zurückgibt. Dabei werden jeweils die Referenzen auf das vorhergehende Feld verwendet. Ähnlich funktioniert auch die Methode zum Abrufen des Pfades nach dem Durchlauf des A*-Algorithmus.

Das maximal einmal verwendbare Boot führte zu einem Problem. A* würde nämlich, um den günstigsten Weg zum Ziel zu finden, das Boot schon recht früh über ein Gewässer einsetzen, welches er auch umgehen könnte. Wenn das Ziel dann auf einer Insel oder hinter einem Fluss liegt, der das gesamte Spielfeld kreuzt, steht das Boot an diesem Punkt nicht mehr zur Verfügung. Sollte der Algorithmus also beim ersten Durchlauf keinen Pfad finden (`PathNotFoundException`), wird A* erneut aufgerufen. Bei diesem zweiten Durchgang vermeidet der Algorithmus das Überqueren von Gewässern bis es keine andere Möglichkeit gibt. Dies geschieht, indem beim Wechsel von Land auf Wasser sehr hohe Pfadkosten addiert werden. Damit diese Strafkosten immer höher als die sonst möglichen Pfadkosten sind, wird sie aus den höchsten eingestellten Terrainkosten (`get_max_terrain_cost`) multipliziert mit Länge und Breite des Spielfelds erstellt.

Weitere Informationen zu einzelnen Methoden finden sich in den Documentation-Strings und Inline-Kommentaren.

Starten der Anwendung

Getestet wurde die Anwendung mit Python 3.8. Für die Ausführung wird Pygame benötigt, welches entweder manuell über Pip installiert werden kann oder alternativ mit Pipenv. Eine Pipfile liegt im Stammverzeichnis der Anwendung.

Beim Start werden in der Konsole sowohl die CSV Quelldatei als auch die gewünschten Terrainkosten eingegeben. Standardwerte können ebenfalls verwendet werden. Im Pygame-Fenster erscheint die Quelldatei als Karte. Durch Klicken im Feld mit der linken Maustaste können dann Start- und Zielfeld ausgewählt werden. Ein gefundener Pfad wird mit roten Punkten markiert, gelbe Punkte zeigen die Bootnutzung an. Mit einem Linksklick lässt sich das Zielfeld verändern, mit einem Rechtsklick das Startfeld.