UNIVERSITÄT OSNABRÜCK

Dr. Thomas Wiemann

11. Übungsblatt zur Vorlesung „Einführung in die Programmiersprache C++“

Wintersemester 2018 / 2019

Aufgabe: Autopilot für den Fighter (100 Punkte)

Ziel:

In dieser Aufgabe werden Sie die Boost Graph Library verwenden, um eine Navigationssoftware für unsere Flieger realisieren. Dazu haben wir ein Sternensystem mit festen Nav-Points modelliert. Auf diesem dreidimensionalen Wegenetz soll mit einer A*-Suche der kürzeste Weg von einem Start- zu einem Zielknoten gefunden werden. Der Flieger fliegt diesen Pfad dann automatisch ab.

Einlesen der Daten (20 Punkte):

Die Daten für den Suchgraphen haben wir in einer einfachen Dateistruktur abgelegt: In der ersten Zeile steht die Anzahl n der Knoten im Graphen. Dann folgen die `n` Knotendefinitionen als Tupel (`Name`, `x`, `y`, `z`). Der Rest der Datei bildet eine Liste von Kanten, wobei jeweils zwei Knotenindices eine Kante definieren. Beispiel:

```
3
Hades 1000 200 1000
Bex 150 3000 -1500
NavPoint1 1000 -1000 -1000
0 1
1 2
2 0
```

Diese Datei beschreibt ein Netz aus drei Knoten, die zusammen einen dreieckigen Routenverlauf darstellen. In der im Repository zur Verfügung gestellten Software wurde die `SpaceCraft`-Klasse um die Funktion erweitert einen gegebenen Pfad abzufliegen. Dazu muss der neuen Methode `navigate()` eine Listen mit Positionen angegeben werden, die dann der Reihe nach angeflogen werden.

Aufgabenstellung:

Schreiben Sie eine Klasse `PathPlanner`. Die Klasse soll mindestens die folgende Signatur haben (eigene Erweiterungen sind notwendig und erlaubt):

```
class PathPlanner {
  public:
    PathPlanner(std::string mapfile);
    std::list<Vector<float> > getPath(Vector<float> position, std::string s, std::string e);
};
```

Der Konstruktor liest den Graphen aus der gegebenen Datei ein. Die Methode `getPath(Vector<float> position, std::string s, std::string e)`

liefert einen Pfad, der von `position` aus zu `s` und vor dort ausgehend auf dem kürzesten Weg zu `e` führt. Die Navigationspunkte `e` und `s` werden dazu über ihren Namen aus der Graph-Datei angesprochen. Um die Indices der Navigationspunkte für die Suche auffinden zu können, legen sie beim Einlesen eine Map an, die unter dem jeweiligen Namen den Index des entsprechenden Graphenknotens einträgt.

Einrichten des Graphen (Konstruktor) (25 Punkte):

Als Datenstruktur wählen wir eine ungerichtete `adjacency_list`. Die Knoten sollen keine ausgezeichneten Eigenschaften haben. Die Kanten im Graphen haben jedoch eine Gewichtung, nämlich ihre Länge. Wir müssen also eine geeignete `property` angeben. Die Kantengewichte werden analog zum Kevin Bacon Beispiel aus der Vorlesung in einer `property_map` gesetzt. Fügen Sie alle Kanten in den Graphen ein. Tragen Sie die jeweilige Kantenlänge an der richtigen Stelle in der `property_map` ein.

Festlegen der Such-Heuristik und Ausführung der Suche (20 Punkte):

Bei der A*-Suche handelt es sich um eine informierte Suche. Wir müssen also eine Heuristik angeben, nach der bei der Traversierung des Graphen die nächste zu expandierende Kante ausgesucht wird. In unserem Beispiel nehmen wir dazu den Abstand des aktuell besuchten Knotens zum Zielknoten. Dazu legen Sie eine eigene Unterklasse von `astar_heuristic` an. Überladen Sie den `()`-Operator

```
CostType operator()(graph_traits<Graph>::vertex_descriptor u);
```

so, dass er für jeden Aufruf den Abstand zum Zielknoten angibt.

Abbrechen der Suche (10 Punkte):

Sobald wir den Zielknoten gefunden haben, soll die Suche beendet werden. Um dieses Verhalten zu erzwingen, implementieren Sie eine Unterklasse des `default_astar_visitors`. Überladen Sie in dieser Klasse die Methode `examine_vertex()`. Werfen Sie eine Exception sobald das Ziel gefunden wurde. Jeder Aufruf von `getPath(...)` soll eine neue Suche anstoßen.

Erklärungen (25 Punkte)

Erläutern Sie Ihrem Tutor alle von Ihnen angelegten Datenstrukturen und Funktionsaufrufe, insbesondere die Boost-Graph-Struktur. Zeigen Sie an einem Beispiel, dass Ihre A*-Suche korrekt arbeitet. Dazu finden Sie in StudIP eine 2D-Version der Graphen-Topologie als Bitmap.

Der Programmaufruf erfolgt folgendermaßen:

```
./asteroids ../models/level.xml ../models/01.map <startpunkt> <endpunkt>
```

Wenn Sie sich an die gegebenen Interfaces halten, werden sie im Viewer die Navigationspunkte und den gefunden Weg für die gegebene Navigationsanfrage angezeigt bekommen. Danach können Sie den Pfad automatisch abfliegen lassen, indem Sie die Taste ‚n‘ drücken. Einen Screenshot dazu sehen Sie auf der folgenden Seite.

Abgabe

Checken Sie Ihre Implementierung bis Montag, 21.01.2019 in Ihr git Repository ein. Bringen Sie einen Ausdruck Ihrer PathPlanner-Implementierung zum Testat mit.