Dr. Steffen Jost
Stephan Barth

Ludwig-Maximilians-Universität München
Institut für Informatik
WS 2017/18

Softwareentwicklungspraktikum Nebenfach

Blatt 3

Dieses Arbeitsblatt ist innerhalb der nächsten drei Wochen in der Gruppe zu bearbeiten.

Gruppenprojekt Flughafen

Aufgabe ist die Implementierung eines *Flughafensimulators* mit JavaFX.

In diesem wird die Bewegung von Flugzeugen auf einem Flughafen simuliert, insbesondere die Wegfindung der Flugzeuge.

Der Flughafen ist hierzu in Knotenpunkte und Pfade unterteilt, auf denen sich die Flugzeuge bewegen können.

Die Flughafensimulation ist in sogenannte Ticks unterteilt, also automatisch ablaufenden Runden.

Jedes Flugzeug bewegt sich pro Tick maximal einen Knotenpunkt weit.

Die Flugzeuge kommen dabei in einem vorgegebenen Muster an, ebenfalls wollen sie gewisse Knotenpunkte mit gegebenen Aufenthaltszeiten besuchen. Die Wege der Flugzeuge auf dem Flughafen hingegen werden automatisch gesucht.

Aufgabe

- Ihr Programm soll beliebige Flughäfen simulieren können. Dazu soll ihr Programm zu Beginn den Flughafen, wie auch die Flugzeuge, welche den Flughafen anfliegen, von einer JSON-Datei einlesen. Das Dateiformat der Kartendaten ist unten beschrieben, Beispiele finden Sie auf der Praktikumshomepage. Es genügt, wenn der Dateiname im Programmcode eingestellt werden kann.
- Damit einzelne Bereiche besser beobachtet werden können, soll Ihr Programm deshalb Zoomen und Scrollen der Karte erlauben, wie aus gängigen Landkartenbetrachtungsprogrammen gewöhnt.
- Es gibt viele weitere Entscheidungsmöglichkeiten, die Sie selbst geeignet festlegen können. So gibt es verschiedene Möglichkeiten für die Anzeige von Informationen zum Spiel (etwa die Unterscheidung zwischen Hangar und Startbahn). Ein weiteres Beispiel für Ihre Entscheidungsmöglichkeiten wäre es, den Algorithmus zum Finden der Wege zu verändern.
- Zum Abnahme des Projekts soll eine Präsentation vorbereitet werden, in der Sie das Ergebnis vorstellen und Ihre Entscheidungen erklären. Details zum Ablauf der Präsentation werden im Plenum besprochen.

Datenformat des Flughafens

Der Flughafen ist im JSON-Format gegeben. Die Beschreibung einer Landkarte besteht aus einem einzigen JSON-Objekt, das folgende Attribute enthält:

- `maxplanes`: Eine Zahl, die angibt, wie viele Flugzeuge gleichzeitig auf dem Flughafen sein dürfen. Ist die maximale Zahl an Flugzeugen erreicht, so warten die weiteren Flugzeuge, bis sie die Karte betreten.
- `planes`: Ein JSON-Objekt mit zwei Attributen:

  `waypoints` ist ein Array von ZielIds, welche das Flugzeug der Reihe nach versucht anzufahren. ZielIds sind Strings, welche im Attribut `targettype` von `nodes` vorkommen.

  `inittime` ist eine Zahl, welche die Runde angibt, in der das Flugzeug das erste Mal versucht die Karte zu betreten.

  Ein Flugzeuge betritt frühestens zu diesem Zeitpunkt die Karte. Ist kein Platz zum Betreten der Karte frei, oder ist die Maximalzahl an Flugzeugen erreicht, so warten die Flugzeuge auf eine spätere Runde.
- `generators`: Ein JSON-Objekt mit zwei Attributen:

  `chance` ist diese Chance als Wert von 0 bis 1.

  `waypoints` ist wie bei `planes`

  Jeder Generator erzeugt mit einer festen Chance pro Tick ein Flugzeug. Wenn das Flugzeug unter den gleichen Bedingungen wie bei `planes` den Flughafen betreten kann, so macht es dies. Falls nicht, so wartet es auf einen späteren Zeitpunkt.
- `nodes` ist ein Array von Knoten und enthält den eigentlichen Flughafen.

  Jeder Knoten ist ein JSON-Objekt, welches folgende Attribute enthält

  - `x`: Die x-Koordinate für die Darstellung als Double.
  - `y`: Die y-Koordinate für die Darstellung als Double.
  - `name`: Ein eindeutiger String als Knotenname, welcher von anderen Knoten verwendet werden kann, um auf diesen Knoten Bezug zu nehmen.
  - `kind`: Art des Knotens. Zur Auswahl stehen folgende String-Werte: `air`, `concrete`, `hangar`, `runway`. Dies kann für die graphische Darstellung der Knoten verwendet werden. Zudem hat die Art des Knotens Auswirkungen auf die Wegsuche.
  - `to`: Dieses Attribut enthält eine Liste von Knotennamen zu denen es eine Verbindung gibt.
  - `conflicts`: Dieses optionale Attribut enthält eine Liste von Knotennamen, welche diesem Knoten im Weg sind, d.h. wenn auf diesem Knoten ein Flugzeug steht, dann müssen alle hier angegeben Konfliktknoten gleichzeitig frei sein.
  - `waittime`: So lange verbleibt ein Flugzeug mindestens auf diesem Knoten, sofern er als Zielknoten angesteuert wurde. Dieses Attribut ist optional.
  - `targettype`: Dieses optionale Attribut enthält eine ZielId als String, welche von den Flugzeugen in dem Attribut `waypoints` verwendet werden kann.

Flugzeugbewegungen

Jedes Flugzeug betritt zur angegebenen Zeit die Karte auf einem zufälligen freien Knoten, dessen `targettype`-Wert dem Anfangswert in `waypoints` entspricht, also die gleiche ZielId hat.

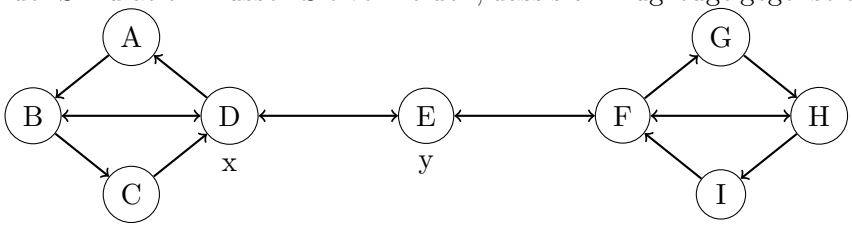
Pro Tick kann sich ein Flugzeug von einem Knoten zu einem benachbarten Knoten bewegen. Beide Knoten gelten in diesem Tick als belegt. Steht ein Flugzeug einen Tick lang still, was nur für Knoten mit `kind "concrete"` oder `"hangar"` erlaubt ist, dann belegt es nur diesen einen Knoten. Nicht stehenbleiben kann es hingegen in `"air"`- oder `"runway"`-Knoten.

Jedes Flugzeug arbeitet der Reihe nach die ZielIds ab, welche in `waypoints` spezifiziert sind.

Wenn nun ein Flugzeug den letzten Eintrag aus `waypoints` erreicht, so wird es aus der Simulation entfernt.

Wegsuche

Beim Ablauf der Simulation müssen Sie vermeiden, dass sich Flugzeuge gegenseitig blockieren:



In den Beispielen stehen kleine Buchstaben für Flugzeuge, große für Knoten. Wenn Flugzeug x nach H möchte und y nach B, so kann sich hier kein Flugzeug mehr weiter bewegen.

Es ist im Allgemeinen nicht leicht, solche „Deadlocks“ zu verhindern, wir empfehlen daher folgende einfache Vorgehensweise: Für jede Flugzeugbewegung eines Flugzeuges zur nächsten ZielId, wird immer der gesamte gefundene Weg vorab reserviert, d.h. es wird für jeden Knoten auf dem Weg für zwei Ticks gespeichert, dass dieses Flugzeug diesen Knoten zu dem entsprechenden Zeitpunkt belegen möchte. Der Zielknoten wird ab der Ankunftszeit dauerhaft reserviert.

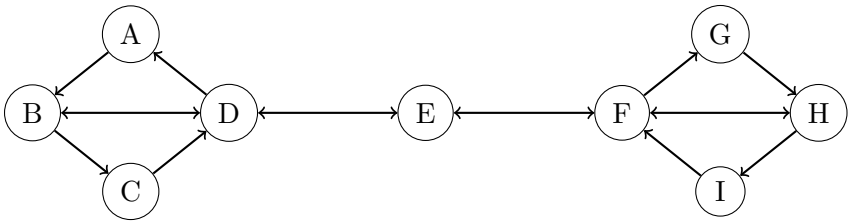
Um einen Weg zu finden, wird eine Breitensuche[1] durchgeführt, in der alle erreichbaren Knoten zu den Erreichbarkeitszeiten, die nicht bereits durch andere Flugzeuge reserviert sind, abgearbeitet werden, bis ein Ziel erreicht ist. Dabei sind jeweils die in `to` angegebenen Knoten erreichbar. Beachten Sie dabei, dass es nicht zwingend auch einen Weg zurück gibt.

Wenn der Flughafen n Knoten hat und der letzte Zeitpunkt, an dem sich laut Reservierung ein Flugzeug bewegen wird, t ist, so kann man annehmen, dass ab dem Zeitpunkt $t+n$ kein Weg zum Ziel gefunden werden kann, wenn bis dahin noch kein Weg gefunden wurde. In diesem Fall soll stattdessen ein Weg zu einem Knoten mit dem `targettype "wait"` gesucht werden. Kann auch zu diesem kein Weg gefunden werden, so verbleibt das Flugzeug bis zum nächsten Tick auf diesem Knoten. Dies ist möglich, da der Knoten ja bereits dauerhaft reserviert wurde.

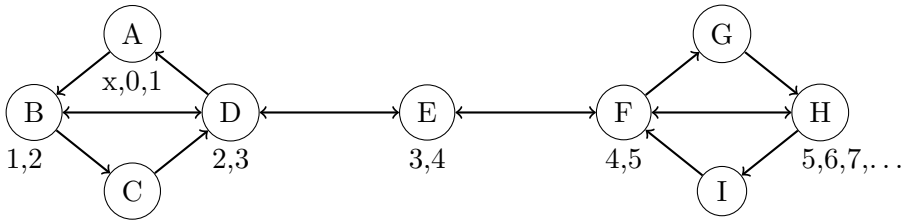
Wird ein Weg gefunden, muss man darauf achten, die bereits getätigte Dauerreservierung für den aktuellen Knoten wieder zu entfernen.

[1] wird im Plenum noch besprochen

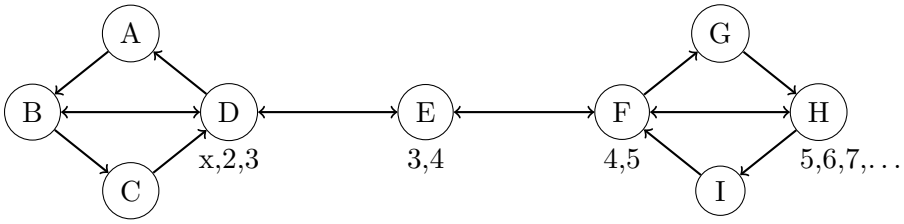
Beispiel 1: Unser Flughafen hat folgende Gestalt, wobei es alles Bodenfelder sind, auf denen ein Flugzeug auch anhalten kann. Was dies im Einzelfall für Felder sind, ist für das Beispiel nicht relevant.



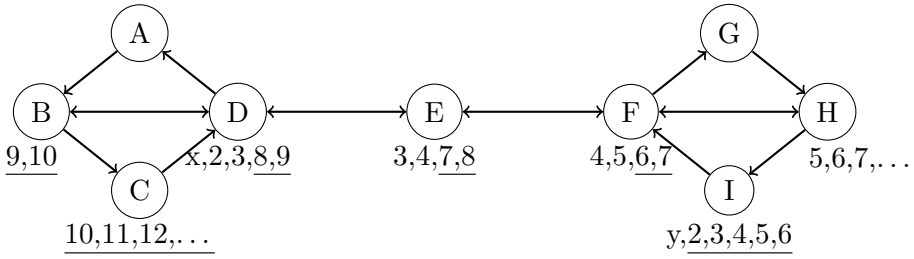
Nun möchte ein Flugzeug x von Feld A zu Feld H. Es reserviert den kompletten Weg zu Feld H. Dazu notieren wir jeden Zeitpunkt an dem das Feld durch das Flugzeug belegt sein wird. Dabei ist zu beachten, dass ein Flugzeug ein Feld immer zwei Ticks lang belegt (da es von einem Knoten in den nächsten fährt, muss es sowohl den Knoten belegen, aus dem es ausfährt, wie auch denjenigen, in den es einfährt). Wir haben also folgende Situation



Jeden Tick zieht das Flugzeug einen Knoten weiter, wobei die veralteten Reservierungen nun keine Rolle mehr spielen. Einige Ticks später ist die Situation damit dann wie folgt



Taucht in diesem Moment ein Flugzeug auf Feld I auf und möchte zu Feld C, so wird folgende Reservierung festgelegt; die Reservierung von Flugzeug y ist jeweils unterstrichen

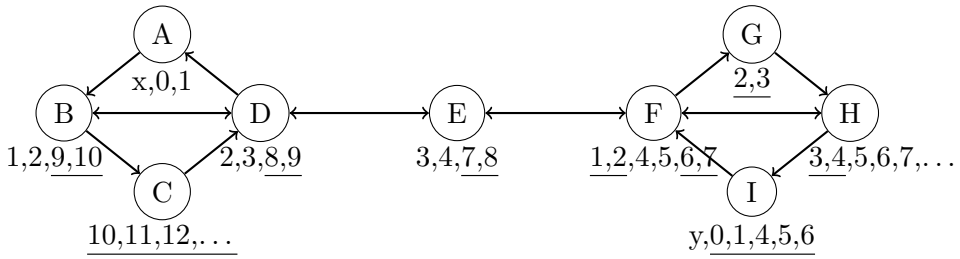


Das neue Flugzeug y geht nicht direkt zu Feld F, da es dann F:3,4 E:4,5 reservieren müsste, wobei es dann feststellt, dass E:4 bereits reserviert ist.

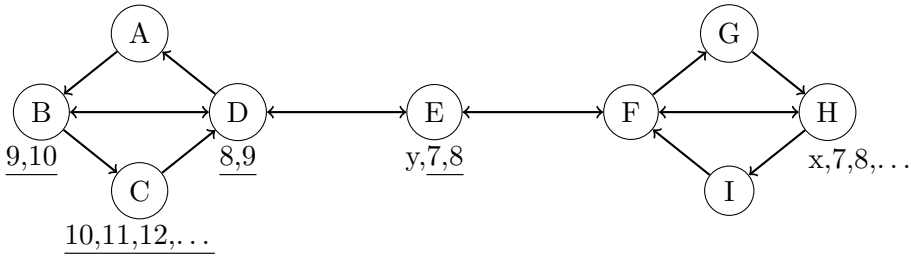
Die Flugzeuge arbeiten nun ihren Weg ab. Sobald sie an ihren Zielknoten sind, (x:H, y:C) verbleiben sie dort.

Beispiel 2: Das Flugzeug x hat die Wegliste A,H,C; das Flugzeug hat y die Wegliste I,C,B.

Die gewählten Wege könnten am Anfang so wie in Beispiel 1 sein, bis auf eine längere Wartezeit von Flugzeug y. Es gibt aber noch weitere gleichkurze Wege, zum Beispiel

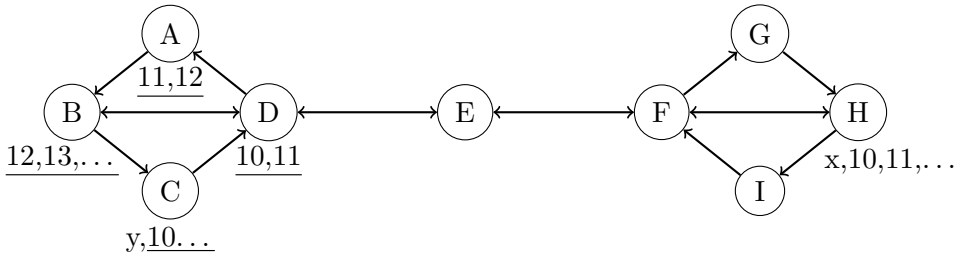


So hat x einen Weg bis H reserviert, sowie y einen Weg bis C. Der Weg wird nun abgearbeitet, etwas später ist die Situation dann wie folgt:



Das Flugzeug x kann keinen Weg zum Knoten C reservieren, da die Reservierungen beliebig weit in der Zukunft durch y belegt sind. Selbst wenn x rechtzeitig zum Knoten C kommen könnte, zu einem Zeitpunkt zu dem dieser noch nicht reserviert ist, so wäre es dennoch kein gültiger Weg, da x den Knoten C nicht in alle Zukunft reservieren kann. Darum muss x warten, bis y die Reservierung irgendwann aufgibt.

Dies geschieht an folgendem Zeitpunkt



Ab diesem Zeitpunkt ist der Knoten C nicht mehr in alle Zukunft reserviert, darum kann x nun einen Weg bis zum Knoten C suchen.

Hinweise

a) Lösen Sie die Aufgabe zunächst ohne Reservierung. Dabei werden dann vermutlich immer wieder Flugzeuge auf dem gleichen Knoten stehen. Dies ist zu diesem Zeitpunkt kein Problem.

   Wenn das zuverlässig funktioniert, bauen Sie die Knotenreservierung ein.

   Beachten Sie hierbei auch, dass manche Flughäfen Knoten enthalten, die an der gleichen Koordinate liegen und den `conflicts`-Mechanismus verwenden, damit keine zwei Flugzeuge auf diesen Knoten gleichzeitig sind. Wenn Sie also den `conflicts`-Mechanismus noch nicht eingebaut haben, kann es also so aussehen, als ob sich zwei Flugzeuge auf dem gleichen Knoten befinden, obwohl sie das nicht sind.

b) JSON ist nur zum Datenaustausch gedacht und nicht dafür, um intern mit Daten zu arbeiten. Verwenden Sie JSON nur beim Einlesen der Daten ins Modell und nicht zur Datenhaltung. Wir empfehlen die Bibliothek org.json zum Umgang mit JSON-Daten.

c) Es empfiehlt sich, frühzeitig mit der Planung des Projektablaufs zu beginnen und beides im nächsten Treffen mit dem Tutor zu besprechen.

   Als Hilfestellung sind im Folgenden einige Aufgaben umrissen, die auch parallel bearbeitet werden können.

   - Einlesen und Verarbeiten der JSON-Kartendaten.

     Berechnen Sie Minimum und Maximum der x- und y- Werte der eingelesenen Daten und lassen Sie sich die Karte in einem ganz einfachen Fenster anzeigen (z.B. `Circle`-Objekte auf einem `Pane`; zunächst ohne Verschieben und Zoomen).
   - Entwicklung einer Datenstruktur, in der Flughafen und die Flugzeuge eingelesen werden. Wählen Sie eine Datenstruktur, welche den häufigsten Zugriffen gerecht wird. Bevor diese Überlegungen jedoch zu einer Blockade führen, entwerfen Sie zuerst ein einfaches Modell mit einer möglichst kleinen Schnittstelle.

     Ebenfalls sollte Ihr Model bereits ein Objekt mit fest kodierten Daten liefern können, damit z.B. die Implementierung der Anzeige nicht auf die Implementierung des Einlesens von JSON-Daten warten muss.
   - Implementierung von View und Controller für das Spiel.

     Beginnen Sie zunächst mit einer einfachen Ansicht, z.B. ohne Verschieben und Zoomen, damit Sie das Modell testen können.

d) Beachten Sie, dass der Bearbeitungszeitraum auch die Vorbereitung der Abschlusspräsentation einschließt. Sie sollten planen, mit der Programmierung etwas eher fertig zu sein.

Für alle Aufgaben (und das ganze Praktikum) gilt: Bei Unsicherheiten und Fragen wenden Sie sich bitte an Ihren Tutor.