OMI Abschlussprojekt

Organisatorisches

Dieses Abschlussprojekt stellt eine Einzelarbeit dar. Kopieren Sie daher nicht den Source Code Ihrer KommilitonInnen. Sollten Sie dies dennoch tun, ist Ihre Arbeit ein Plagiat und kann daher nicht positiv benotet werden. Sie dürfen externe Quellen (Stackoverflow etc.) verwenden, sollten Sie aber größere Teile Sourcecode aus dem Internet kopieren, markieren Sie diese bitte mitsamt Quellenangabe in Ihrem Sourcecode. Kleinere Blöcke und einzelne Zeilen Code sowie trivialen Code müssen Sie nicht markieren.

Übersicht

Im Rahmen Ihres Abschlussprojektes sollen Sie ein generisches Wirtschaftssimulationsspiel entwickeln. Dazu erhalten Sie ein Rumpfprojekt, sowie eine Spezifikation für das restliche Projekt. Ihre Aufgabe ist es das Rumpfprojekt entsprechend der Spezifikation zu vervollständigen. Beachten Sie bitte, dass die Spezifikation im Detailgrad variiert. Treffen Sie für etwaige fehlende Informationen vernünftige Annahmen. Sie dürfen den vorgegebenen Code verändern, sofern Sie dies für sinnvoll erachten. Achten Sie allerdings dabei darauf, dass Sie nur den Code und nicht die Spezifikation verändern. Markieren Sie in Ihrem Code bitte Veränderungen mittels Kommentare. Beachten Sie außerdem gängige Coding Konventionen so wie das Benutzen der englischen Sprache, das Groß/Kleinschreiben von Klassen, Variablen etc...

In dieser Angabe werden nur die wichtigen Teile der Spezifikation behandelt. Implementieren Sie zusätzlich alle notwendigen Konstruktoren sowie Getter und Setter Methoden. Sie können (und sollen an sinnvollen Stellen) zusätzliche Helfermethoden implementieren. Des Weiteren finden Sie mehrere TODO Kommentare im Sourcecode, die Sie implementieren und abarbeiten sollen.

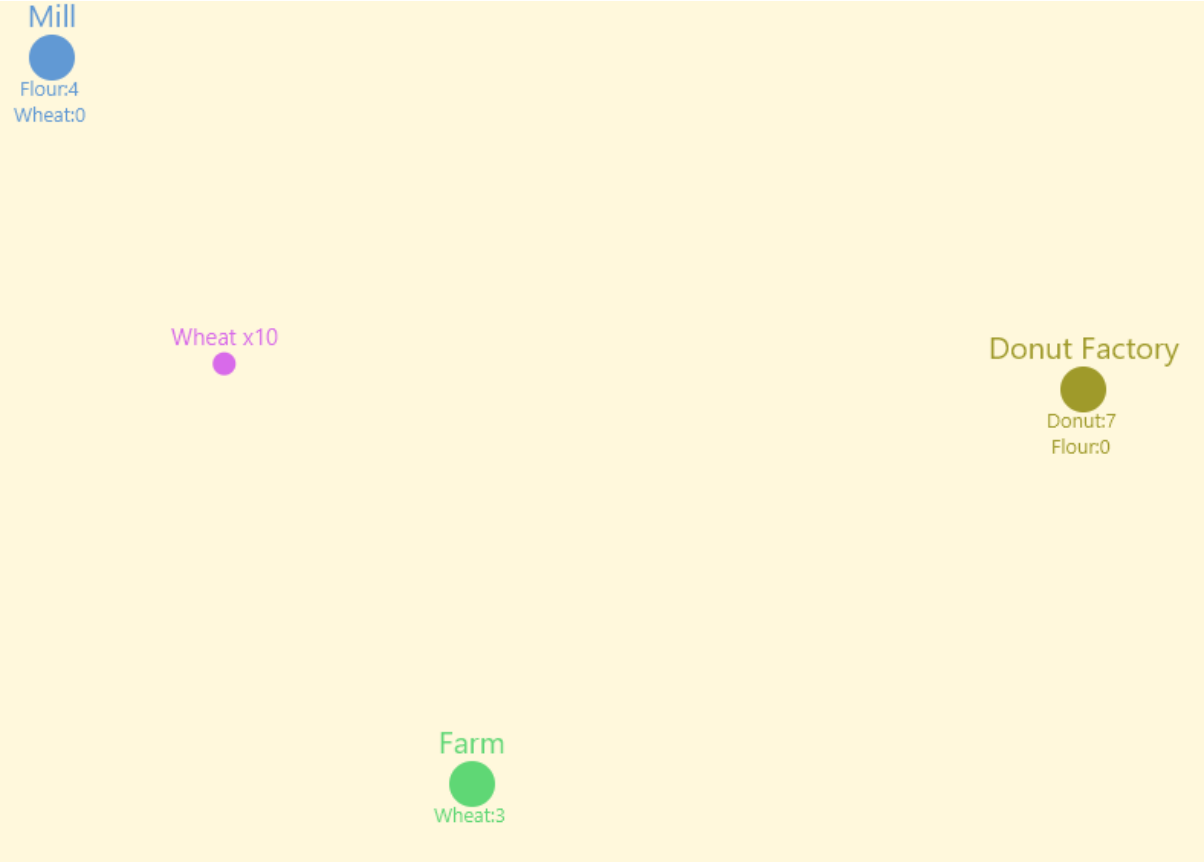
In der Wirtschaftssimulation gibt es verschiedene Standorte (location) für Betriebe. An einem Standort werden bestimmte Ressourcen erzeugt (z.B. Weizen am Standort Bauernhof) und/oder weiterverarbeitet (z.B. Weizen wird am Standort Mühle zu Mehl verarbeitet). Die Weiterverarbeitung von Ressourcen an einem Standort geschieht nach einem bestimmten Verfahren bzw. Rezept (recipe). Ein Rezept legt grundsätzlich fest wie viele Einheiten einer Ressource A notwendig sind, um eine bestimmte Anzahl an einer Ressource B zu erzeugen (z.B. wie viele Einheiten Weizen braucht man für wie viele Einheiten Mehl).

Ein weiterverarbeitendes Unternehmen (z.B. der Standort Mühle) muss natürlich mit Ressourcen beliefert werden. Die benötigten Ressourcen ergeben sich aus den angewandten Rezepten gemäß denen Bestellungen (order) nach Ressourcen ausgegeben werden. Die Weitergabe von Ressourcen bzw. die Übermittlung einer Bestellung zwischen den einzelnen Standorten der erfolgt mittels eines

Transportmittels (carrier). Für die Abarbeitung der Bestellungen können verschiedene Algorithmen angewendet werden.

Grafische Benutzerschnittstelle

Das ausgegebene Projekt enthält auch eine grafische Benutzerschnittstelle um die Simulation darzustellen bzw. um diese teilweise zu beeinflussen:



Die grafische Benutzerschnittstelle basiert auf dem JavaFX Framework. Je nach Setup Ihres Systems (ab Java 11) kann es sein, dass Sie den SDK bzw. Libraries dafür noch installieren müssen. JavaFX kann hier heruntergeladen werden: https://openjfx.io/

Eclipse Hinweise:

Ein Beispiel für die Einrichtung unter Eclipse finden Sie hier:
https://www.youtube.com/watch?v=xTSxNw7UHng

Ihre module-info.java Datei sollte wie folgt aussehen:

```
module simulation {
        requires javafx.controls;
        requires javafx.base;
        requires javafx.fxml;
        requires javafx.graphics;
        requires javafx.media;
        requires java.desktop;
```

```
        exports main.java;
        exports main.java.animation;
        exports main.java.logic;
        exports main.java.logic.algorithms;
        exports main.java.logic.entities;
        exports main.java.views;
}
```

IntelliJ Hinweise:

Ein Beispiel für die Einrichtung unter IntelliJ finden Sie hier:
https://openjfx.io/openjfx-docs/#IDE-Intellij

Sourcecode

Game:

Diese Klasse stellt die Hauptklasse des Programmes dar. Diese Klasse ist größtenteils fertiggestellt, achten Sie allerdings auf die Kommentare im Sourcecode.

src.main.java.animation

Im Package animation befinden sich die Klassen, die auf der GUI animiert.

DrawableObject

Dieses Interface ist bereits vorgegeben. Sie sollen hierzu das Composite Pattern anwenden.
Schreiben Sie hierzu die Klasse DrawableObjectsComposite.

Schreiben Sie die Datenklasse Point, die zwei doubles x und y hat. Die Klasse beschreibt die x und y Koordinaten eines am Bildschirm dargestellten Objekts. Erweitern Sie die Klassen Label, Rectangle und Circle. Alle sollen Drawable Implementieren. Implementieren Sie hier alle nötigen Felder, Konstruktoren sowie Getter und Setter Methoden. In den Klassen Label und Rectangle sind die draw() Methoden jeweils bereits schon implementiert. Die Klasse Circle erfordert etwas mehr.

Circle

Diese Klasse funktioniert sehr Ähnlich zu den Klassen Label und Rectangle. Sie soll allerdings zusätzliche von DrawableObjectsComposite erben. Die Idee hier ist, dass Circle selbst drawable Objekte halten kann. In dieser Applikation wird das dazu benutzt um zusätzliche Informationen anzuzeigen (Text und Description). Dazu sollen zwei Methoden (public void setText(String s, int fontsize) und public void setDescription(String s, int fontsize)) angeboten werden. Diese Methoden fügen jeweils ein Label Objekt hinzu (entsprechend des Composite Patterns). Beachten Sie hierbei, dass jeweils nur immer eine Description und ein Text angezeigt werden kann, Sie müssen hierzu eventuell alte Objekte aus der Liste nehmen.

Zudem gibt es die Klasse AnimationHelper. Hier ist nur eine kleine Anpassung nötig.

src.main.java.logic

Im Package logic befinden sich 2 weitere Packages (Algorithms und Entities). Zudem befindet sich hier die Klasse GameLogic, die einen großen Teil der Spielmechaniken abbildet. Für diese Klasse wird Ihnen ein Stub (Eine Rumpfklasse die noch nicht fertig ausimplementiert ist) bereitgestellt, um Ihnen die Implementierung zu erleichtern. Einige Methoden sind zudem bereits ausimplementiert. Die Klasse

GameLogic soll als Singleton implementiert werden. In der Methode init() werden sowohl die einzelnen Standorte sowie deren Bestellungen an Ressourcen initialisiert.

algorithms

Dieses package enthält den Code und die Klassen für die Entscheidungsalgorithmen zur Abarbeitung der orders. Diese Algorithmen sollen entscheiden welche order einer location zuerst erfüllt wird, wenn eine location mehrere order hat, die erfüllt werden können. Hierzu hat die location ein field algorithm, das mit einem beliebigen Algorithmus befüllt werden kann.

Die Algorithmen sollen dabei beliebig austauschbar sein. Verwenden Sie zur Implementierung hierzu das Strategy Pattern. Im Projekt finden Sie bereits das Interface Algorithm. Implementieren Sie vier sinnvolle Entscheidungsalgorithmen, die beliebig austauschbar sind. Darunter müssen zumindest ein rein zufallsbasierter Algorithmus sowie ein „default“ Algorithmus der die order einfach der Reihen nach abhandelt sein.

Zur Verwaltung der einzelnen Algorithmen gibt es die Klasse AlgorithmRegistry, in der Algorithmen registriert und bei Bedarf abgerufen werden können. Diese Klasse ist vorgegeben.

entities

Dieses Package enthält die Klassen, die die eigentlichen Spielelemente darstellen.

Es gibt zwei Grundlegenden Spielobjekte: Carrier und Locations. Beide erben von der abstrakten Klasse GameObject. Jedes GameObject hat einen Namen sowie ein DrawableObject (z.B. einen Kreis) zur Darstellung am Bildschirm. Zentral ist die Methode update, welche von der Gameengine in regelmäßigen Abständen aufgerufen wird. Subklassen können diese Methode dazu nutzen um Ihren Zustand zu aktualisieren beispielsweise um Bestellungen abzuarbeiten oder neue Ressourcen zu produzieren. Ein Objekt der GameObjects sollen nicht direkt erstellt werden, sondern mittels Factory Pattern. Schreiben Sie hierzu die Klasse GameObjectFactory. Des Weiteren beinhaltet das Package auch noch die Logik für Orders und Recipes. Im Folgenden erhalten Sie zusätzliche Informationen zu den Klassen.

GameObject:

Ist vorgegeben. GameObject ist abstract. Es ist Ihre Aufgabe Klassen zu implementieren die von GameObject erben.

GameObjectFactory:

Ist eine typische factory Klasse. Bietet die Methode:
public static GameObject getGameObject(String objectType, Point point) an. Der übergebene String gibt dabei den Typ des Objektes. Es ist dabei Ihre Aufgabe sich konkrete Typen zu überlegen und zu implementieren.

Location:

Die Klasse Location ist eine der wichtigsten Klassen im Spiel. Daher erhalten Sie hier einen Stub. Beachten Sie zudem, dass die Klasse selbst abstrakt ist. Ihre Aufgabe unter anderem ist, fünf konkrete Subklassen zu erzeugen, damit diese im Spiel verwendet werden können. Neue Game Objekte werden in der init-Methode der Klasse GameLogic registriert. Ein Beispiel ist gegeben. Verwenden Sie dieses Beispiel nicht weiter.

Diese Klasse ist teilweise vorgegeben. Ihre Aufgabe ist es sie entsprechend der Spezifikation zu erweitern. Beachten Sie hierzu die Kommentare im Sourcecode.

Carrier:

Klasse Carrier stellt ein Transportmittel zwischen den einzelnen Locations am Bildschirm dar. Carrier erweitert GameObject. Ein Carrier hat (zumindest) folgende Felder:

static double DIAMETER = xx; //Change this diameter to something reasonable
Point point;
Circle circle; // Wird bei getDrawable() returniert.

Carrier bietet eine zusätzliche public Methode an:

public void setName(String s): Setzt das Feld name auf s. Und ruft circle.setText(s, xx) auf. xx muss hier selbstverständlich ersetzt werden.

Order:

In der Klasse Order wird eine Bestellung zwischen zwei Locations abgebildet. Entsprechend gibt es Felder für eine Start- und eine Ziel-Location, eine bestimmte Ressource, sowie deren Menge. In der Methode checkOrder wird zunächst überprüft, ob an der Start-Location die bestellte Ressource überhaupt und in der erforderlichen Menge vorhanden ist. Ist eine Ressource in der Start-Location in ausreichender Menge vorhanden, wird deren Menge gemäß der Bestellung verringert und ein neuer Carrier für den Transport erzeugt (spawnCarrier() Methode in GameLogic). Das Ankommen einer Bestellung wird über die Methode onArrival() abgearbeitet, wobei die Menge der Ressource in der Ziel-Location entsprechend erhöht wird.

Recipe:

Ein Object der Klasse Recipe wandelt bestimmte Eingangsressourcen über eine Produktion in neue Ausgangsressourcen um (z.B. Weizen zu Mehl). Die Produktion braucht dafür natürlich eine gewisse Zeit. Weiters kann angegeben werden, ob es sich um eine einmalige Produktion handelt oder ob das Rezept wiederholt angewendet wird. Zu Beginn der Produktion werden die vorhandenen Eingangsressourcen am jeweiligen Standort (Location) entsprechend verringert, nach Abschluss der Produktion die Ausgansressourcen am Standort entsprechend erhöht. Eine Produktion kann natürlich nur gestartet werden, wenn am jeweiligen Standort genügend Eingangsressourcen vorhanden sind. Recipe bietet hierzu eine public Methoden an: public void checkRecipe(Location location, double delta). Diese Methode wird regelmäßig (bei jedem Update) aufgerufen, dabei wird die delta time übergeben. Die delta time stellt die Zeit seit dem letzten Update dar.

Verwenden Sie zudem für Recipes das Factory Pattern indem Sie eine Factory Klasse erstellen.

src.main.java.views

Im Package views befinden sich die Klassen, die für die JavaFX GUI verantwortlich sind. Die Klassen hier sind größtenteils bereits fertiggestellt. Mach Sie sich dennoch mit den Klassen vertraut. Zudem sollen Sie die Klasse GameView verbessern. Anweisungen hierzu finden Sie in der Klasse selbst als Kommentar.