Die Hausaufgaben müssen von jedem Studierenden einzeln bearbeitet und abgegeben werden. Für die Hausaufgabe sind die aktuellen Informationen vom Blog https://seblog.cs.uni-kassel.de/ws2324/programming-and-modelling/ zu berücksichtigen.

Abgabefrist ist der 11.01.2024 - 23:59 Uhr

Abgabe

Wir benutzen für die Abgabe der Hausaufgaben Git. Jedes Repository ist nur für den Studierenden selbst sowie für die Betreuer und Korrektoren sichtbar.

Für die Hausaufgabe benötigst du **kein neues** Repository. Es wird das gleiche Repository benutzt, das bereits in Hausaufgabe 6 angelegt wurde. Dieses kann über folgenden Link erstellt werden, falls nicht bereits geschehen:

https://classroom.github.com/a/kl_i4rdv

Nicht oder zu spät gepushte (Teil-)Abgaben werden mit 0 Punkten bewertet!

Abgaben, die nicht lauffähig sind, werden mit 0 Punkten bewertet!

Alle Tests (alt/neu) müssen nach wie vor funktionieren (ggf. angepasst werden), sollte dies nicht der Fall sein, wird mit 0 Punkten bewertet!

Aufgabe 1 - Shop und Liste der Autos anzeigen (37P)

Ziel dieser Aufgabe ist es, dass der User im Spielgeschehen über den Shop neue Autos kaufen kann. Weitere Hilfsmethoden, -variablen und -konstanten sind erlaubt.

Vorgegebene Dateien

Füge die vorgegebenen Dateien in das entsprechende Package unter src/main/java in dein Projekt ein.

- `de.uniks.pmws2324.tiny.controller.CarSubController` [1]
- `de.uniks.pmws2324.tiny.controller.ShopController` [2]

CarComponent.fxml

Lagere die UI-Elemente zum Anzeigen eines Autos in eine eigene fxml-Datei mit dem Namen `CarComponent.fxml` aus. Diese soll im gleichen Verzeichnis und Package abgelegt werden wie die anderen FXML-Dateien.

Die Benennung der fx:ids soll gleich bleiben. Zur Erinnerung:

- carDriverLabel für den Text, der den Namen des Fahrers enthält
- carDestinationLabel für den Text, der die Zielstadt enthält oder leer bleibt, wenn das Auto keinen Auftrag hat

StackPane und Anpassungen in Game.fxml

Ändere deine Game.fxml, wie in der Übung gezeigt, so ab, dass der oberste Knoten eine `StackPane` ist. Als fx:id soll `rootPane` vergeben werden.

In der `Game.fxml` müssen nun die alten Blöcke zur Anzeige von Autos entfernt werden. Das Container-Element, in dem die Autos angezeigt werden sollen, soll die fx:id `carBox` erhalten.

Neues Attribut im HeadQuarter

Füge im GenModel beim HeadQuarter eine neue `int` Variable mit dem Namen `newCarPrice` hinzu. Nachdem das GenModel angepasst wurde, muss das Klassenmodell erneut generiert werden. Die Variable soll künftig dafür genutzt werden, sich den Preis des nächsten Autos zu merken, das im Shop verfügbar ist.

CarSubController

Implementiere die Methoden des vorgegebenen CarSubControllers entsprechend der // TODO-Kommentare.

[1] https://github.com/sekassel/pmws2324-files/blob/main/HA08/CarSubController.java
[2] https://github.com/sekassel/pmws2324-files/blob/main/HA08/ShopController.java

Neue Logik im GameService

In der `init`-Methode muss der `newCarPrice` zunächst initial auf einen angemessenen Wert gesetzt werden. Füge dann dem GameService eine Methode `void buyCar(String driver)` hinzu. In der Methode soll zunächst geprüft werden, ob der HeadQuarter genug Geld hat, sich das nächste Auto zu kaufen. Im positiven Fall wird ein neues Auto angelegt. Der Fahrer des Autos bekommt den Namen, der der Methode übergeben wurde. Als Owner des Autos wird der HeadQuarter gesetzt, ebenso als aktuelle Position. Der Preis des Autos muss vom Kontostand des HeadQuarters abgezogen werden. Zu guter Letzt wird dem `newCarPrice` ein neuer zufälliger Wert aus einem angemessenen Bereich zugewiesen.

ShopController

Implementiere die Methoden des vorgegebenen ShopControllers entsprechend der `// TODO`-Kommentare.

GameController anpassen

Um die neue Funktionalität ins Spiel einzubinden, muss der GameController entsprechend erweitert werden.

CarSubController einbinden: An den Stellen, an denen vorher die UI-Komponenten des einzigen Autos angezeigt und aktualisiert wurden, muss nun für jedes Auto des HeadQuarters ein CarSubController-Objekt angelegt und initialisiert/gerendert werden. Um auf das `carBox`-Element zugreifen und dieses an den Konstruktor des CarSubControllers übergeben zu können, muss dafür eine Variable angelegt werden, die mit `@FXML` annotiert werden soll. Die alten Variablen der UI-Elemente, die aus der Game.fxml gelöscht wurden, müssen entfernt werden.

ShopController einbinden: Füge zunächst eine Variable hinzu, über die auf die `rootPane` zugegriffen werden kann. In der `render`-Methode des GameControllers muss ein neuer ShopController angelegt werden, dem die `app`, der `gameService` und die `rootPane` übergeben wird. Um auf den `shopButton` zugreifen zu können, muss dafür ebenfalls eine Variable angelegt werden. Der `shopButton` soll in seiner Action die Methoden zum Anzeigen des Shops aufrufen.

Außerdem soll im `carCostLabel` immer der aktuelle Preis des nächsten Autos angezeigt werden. Lege für dieses UI-Element ebenfalls eine Variable an und setze den initialen Text und den benötigten Listener in der `render`-Methode.

Listener abmelden

Melde im GameController in der `destroy`-Methode alle Listener ab, wie in der Übung gezeigt.

Aufgabe 2 - Weitere Spiellogik (9P)

Ziel dieser Aufgabe ist es, das Spielgeschehen durch zufällige Ereignisse spannender zu machen. Weitere Hilfsmethoden, -variablen und -konstanten sind erlaubt.

Vorgegebene Dateien

Füge die vorgegebene Datei in das entsprechende Package unter src/main/java in dein Projekt ein.

- `de.uniks.pmws2324.tiny.service.TimerService` [3]

TimerService

In der Übung wurden Timer vorgestellt. Implementiere die `changeRandomThings`-Methode wie folgt:

- es wird zufällig entschieden, ob eine neue Order generiert wird oder die Blockade einer Straße geändert wird
- eine neue Order soll allerdings nur dann generiert werden, wenn jede Stadt weniger als fünf Orders hat
- Straßen sollen nur blockiert werden, wenn es weiterhin vom HeadQuarter mindestens eine mögliche Strecke zu jeder anderen Stadt gibt
- als zeitlichen Abstand für die Zufallsereignisse kann ein Bereich von z.B. 1 bis 5 Sekunden gewählt werden

GameController

Lege im Konstruktor des GameControllers einen TimerService an und lege diesen in einer Variable ab, um an geeigneter Stelle die `stop`-Methode darauf aufzurufen.

Passe das Zeichnen der Straßen so an, dass blockierte Straßen ersichtlich dargestellt werden. Passe das Zeichnen von Orders so an, dass ersichtlich ist, wie viele Orders an einer Stadt verfügbar sind.

[3] https://github.com/sekassel/pmws2324-files/blob/main/HA08/TimerService.java

Aufgabe 3 - TestFx (9P)

Als Abschluss für diese Hausaufgabe sollen die eben implementierten Teile getestet werden. Dies geschieht wie in der Vorlesung gezeigt mithilfe der hinzugefügten TestFX-Library.

Vorgegebene Test-Klasse

Nutze die zur Verfügung gestellte Test-Klasse AppTest, um mit der Implementierung des TestFx-Tests zu beginnen. Füge den AppTest unter src/test/java in das Package de.uniks.pmws2324.tiny in dein Projekt ein.

- `de.uniks.pmws2324.tiny.AppTest` [4]

Implementierung

Der buyNewCar-Test soll folgenden Ablauf implementieren:

- den Fenstertitel prüfen
- die Anzahl der angezeigten Autos und den Fahrernamen prüfen
- das Geld des HeadQuarters über den GameService erhöhen, damit ein Auto gekauft werden kann
- über die Oberfläche ein Auto kaufen und Fahrernamen „Bob“ eingeben
- erneut Anzahl der angezeigten Autos und Fahrernamen prüfen
- angezeigtes Geld überprüfen

Hinweise

Beachte, dass alle vorher angelegten Tests ebenfalls erfolgreich durchlaufen müssen! Passe wenn nötig alte Test-Klassen an, damit dies der Fall ist. Die Tests müssen weiterhin sinnvoll sein, insbesondere die geplante Funktionalität prüfen und Assertions enthalten.

Das Auskommentieren oder Löschen von alten Tests ist nicht erlaubt und führt zu Punktabzug!

Sollten die Tests nicht lauffähig sein oder nicht erfolgreich durchlaufen, führt dies zu 0 Punkten!

Bei der Ausführung und Benutzung des Programms sollten keine Exceptions auftreten. Nach Bearbeitung der Aufgabe sollen alle `TODO`s aus den Kommentaren entfernt sein.

Committe und pushe die Änderung abschließend auf den main-Branch.

Achte darauf, das Repository der aktuellen Hausaufgabe zu verwenden.

[4] https://github.com/sekassel/pmws2324-files/blob/main/HA08/AppTest.java

Es folgt eine Auflistung hilfreicher Webseiten und weiteren Erklärungen zu den Themen dieser Hausaufgabe. Die Links sind als Startpunkt zur selbstständigen Recherche angedacht. Das Durcharbeiten der folgenden Quellen ist kein bewerteter Anteil der Hausaufgaben.

Zur Verfügung gestellte Dateien

- https://github.com/sekassel/pmws2324-files/tree/main/HA08

TestFX

- Beispiele auf GitHub: https://github.com/testfx/testfx#examples