SoPra der Toten

Spiel mit dem Schicksal

Einzelphase

Inhaltsverzeichnis

1 Organisatorisches

Hinweis

Bitte beachten Sie für den Ablauf der Einzelphase auch die Informationen aus dem Dokument „Organisatorisches“, das Ihnen bereits zu Beginn des Software-Praktikums im CMS zur Verfügung gestellt worden ist.

Nachdem Sie die Gruppenphase erfolgreich überstanden und gemeinsam mit Ihrer Gruppe das Spiel entworfen, implementiert und getestet haben, müssen Sie jetzt in der Einzelphase alleine zeigen, dass Sie die Konzepte der Vorlesung verstanden haben und umsetzen können. In der Einzelphase werden Sie eine Erweiterung des Spiels aus der Gruppenphase entwerfen, implementieren und testen.

Sie bekommen dazu ein neues *git*-Repository im lehrstuhleigenen *GitLab*[1], auf das nur Sie allein Zugriff haben. Darin stellen wir Ihnen eine Implementierung des Spiels der Gruppenphase zu Verfügung, auf der Sie aufbauen sollen. (Durch das Zurverfügungstellen unserer Implementierung stellen wir sicher, dass alle Teilnehmer die gleiche Ausgangslage der Implementierung haben, unabhängig vom Gruppenerfolg zuvor.) Wir legen Ihnen ans Herz, die zur Verfügung gestellte Implementierung des Spiels der Gruppenphase sowie dessen Entwurf für Ihr Einzelphasenprojekt zu verwenden. Allerdings steht es Ihnen natürlich frei, diesen Entwurf und die zugehörige Implementierung beliebig[2] abzuändern. Um Ihnen das Einarbeiten in unsere Implementierung zu erleichtern, haben wir eine Beschreibung des Entwurfs und der Implentierung sowie ein Klassendiagramm erstellt, welche Sie im CMS finden können.

Die Einzelphase wird vollends online durchgeführt. Während der Einzelphase gibt es *keine* Anwesenheitspflicht, d.h., Sie können sich Ihre Zeit selbst einteilen und müssen auch für niemanden erreichbar sein. **Gruppenarbeit ist in der Einzelphase verboten, jeder Teilnehmer muss das Softwaresystem (d.h., Entwurf, Implementierung, Tests) komplett eigenständig erstellen.** Allerdings stehen Ihnen die Tutoren nach wie vor in Office-Hours[3] zur Verfügung, um Ihnen bei Fragen zu helfen.

Die finale Abgabe der Einzelphase müssen Sie bis spätestens Freitag, 15.10.2021, 23:59 Uhr, in Ihr Einzelphasen-Repository pushen.

[1] https://sopra.se.cs.uni-saarland.de/

[2] Plagiieren ist natürlich trotzdem verboten, Sie dürfen lediglich Ihre eigenen Ideen oder Konzepte Ihrer Gruppe aus der Gruppenphase wiederverwenden.

[3] ab Montag, 04.10.2021, 14:00 Uhr. Genauere Details zu den Office-Hours entnehmen Sie bitte dem Terminkalender im CMS.

1.1 Aufgabe

Um die Einzelphase zu bestehen, müssen Sie einen Entwurf für die unten folgende Spielerweiterung erstellen, der den Konzepten der Vorlesung folgt, und am Ende der Einzelphase eine lauffähige Implementierung abgeben, die Ihrem Entwurf entspricht. Dazu müssen Sie auch ein Dokument erstellen, das erklärt, wie die Konzepte aus der Vorlesung in Ihren Entwurf eingeflossen sind. Wir stellen Ihnen dazu ein Template im CMS zur Verfügung, das Ihnen als Vorlage und Beispiel für die Form dieses Dokuments dienen soll. Außerdem müssen Sie für Ihre Einzelphasen-Implementierung sinnvolle Unit-Tests und Systemtests schreiben und bestehen. Das bloße Bestehen sämtlicher Tests reicht nicht aus, Ihr Code muss auch gut strukturiert sein und den Konzepten der Vorlesung folgen.

1.2 Entwurf

Sie müssen für Ihren Entwurf ein Klassendiagramm erstellen. Dabei können Sie auf dem zur Verfügung gestellten Klassendiagramm aufbauen. In Ihrem Klassendiagramm müssen alle Klassen vorkommen, die Sie in Ihrem eigenen Entwurf neu hinzufügen, und ebenso alle Klassen, die Sie selbst ändern oder erweitern. Alle übrigen Klassen aus dem zur Verfügung gestellten Entwurf brauchen Sie nicht in Ihr Klassendiagramm aufzunehmen, sofern Sie diese unverändert lassen. Achten Sie beim Hinzufügen neuer Klassen jedoch auch darauf, dass Beziehungen zwischen Klassen auch dann im Klassendiagramm enthalten sein müssen, wenn eine unveränderte Klasse betroffen ist. (Wenn sie z.B. eine neue Klasse haben, die von einer bestehenden Klasse erbt, welche Sie unverändert lassen, so muss die Klasse, von der Ihre neue Klasse erbt, in Ihrem Diagramm enthalten sein, obwohl diese unverändert ist, um die Vererbungsbeziehung im Klassendiagramm darstellen zu können.)

Für die Entwurfsabgabe am Freitag, 08.10.2021, muss Ihr Entwurf (in Form eines UML-Klassendiagramms in PDF-Format) bis 23:59 Uhr im CMS hochgeladen werden. Ebenso muss Ihre Projektbeschreibung bis Freitag, 08.10.2021 um 23:59 Uhr im CMS separat hochgeladen werden.

1.3 Abgabe

Am Ende der Einzelphase (Freitag, 15.10.2021) muss Ihre finale Implementierung (inkl. Tests[4]) auf dem `master`-Branch des für Sie zur Verfügung gestellten Repositorys liegen. Es zählt der letzte Commit auf dem `master`-Branch, der bis spätestens 15.10.2021, 23:59 Uhr MESZ im Repository gemacht wurde. Ebenfalls muss eine aktualisierte Version Ihrer Projektbeschreibung im CMS hochgeladen werden. Diese aktualisierte Version soll die Änderungen enthalten, welche Sie bei Ihrer finalen Implementierung gegenüber dem Entwurf vorgenommen haben.

[4] Weitere Information zu den erwarteten Tests finden Sie in Tests.

Es liegt in Ihrer Verantwortung zu überprüfen, dass der Push im Repository angekommen ist und alles rechtzeitig auf dem aktuellen Stand ist. Ihre Abgabe muss auf unserer Referenzplattform (*Debian 10* mit *OpenJDK 16*) ausführbar sein.

1.4 Überblick

In Die Spielerweiterung stehen alle Einzelheiten zur Spielerweiterung, die Sie in der Einzelphase implementieren müssen. Technische Details und Hinweise zur Implementierung in der Einzelphase finden sich in Technische Details. Zuletzt folgen noch Hinweise zum Testen Ihrer Implementierung in Tests.

Hinweis

Beachten Sie, dass alle Infos aus dem Spezifikations-Dokument der Gruppenphase weiterhin Gültigkeit haben (sofern diese nicht mit den Neuerungen aus den nachfolgenden Kapiteln in Konflikt stehen).

2 Die Spielerweiterung

Vier Wochen sind vergangen, seitdem wir uns an der Universität verbarrikadiert haben. Wir mussten leider feststellen, dass das Leben in dieser postapokalyptischen Welt nicht ganz so linear ist, wie wir ursprünglich dachten...

2.1 Schicksalskarten

Pro Runde wird für jeden Spieler eine neue Schicksalskarte gezogen, wenn er am Zug ist. Je nach dem welche Aktionen in dieser Runde ausgeführt werden, wird das Schicksal getriggert und nimmt seinen Lauf. Dabei kann z.B. Essen verschwinden, neue Überlebende auftauchen oder Zombies spawnen, obwohl man sich doch gerade so schön in Sicherheit gewogen hat. Manchmal haben die Spielerinnen und Spieler aber auch die Qual der Wahl über das Schicksal der Gruppe...

2.1.1 Trigger

Eine der folgenden Aktionen können das Schicksal (*Crossroads Event*) triggern:

- Anlegen von Ausrüstung
- Änderung des Müllstapels
- Durchsuchen eines (optional: bestimmten) Standortes
- Bewegen (optional: zu einem bestimmten Ort)
- Verbarrikadieren (optional: eines bestimmten Ortes)

Dabei triggert immer die Erste dieser Aktionen, die ein Spieler bzw. eine Spielerin in einer Runde ausführt, die Schicksalskarte. Das geschieht jedoch nicht sofort, sondern erst, nachdem die entsprechende Aktion des Spielers oder der Spielerin abgeschlossen ist. Mehr zu den neuen Commands und Events finden Sie in Abschnitt 3.3.

2.1.2 Konsequenzen

Jede Schicksalskarte zieht eine Konsequenz nach sich. Entweder es folgt genau eine durch die Schicksalskarte festgelegte Konsequenz, oder aber es muss von allen Spielerinnen und

Spielern über zwei verfügbare Optionen abgestimmt werden. Dabei hat jeder Spieler und jede Spielerin genau eine Stimme.

Mögliche Konsequenzen sind:

(1) Der Nahrungsvorrat ändert sich (`changeFood`). Sollte dem Nahrungsvorrat mehr Essen entnommen werden müssen, als vorhanden ist, wird stattdessen ein Hungermarker hinzugefügt.

(2) Die Moral ändert sich (`changeMoral`). Sollte die Moral dadurch auf 0 sinken, gilt das Spiel wie gehabt *sofort* als verloren.

(3) Neue Überlebende spawnen ()`spawnSurvivors`) mit optionalem Feld `children`, falls zusätzlich genauso viele Kinder wie neue Überlebende spawnen sollen.

(4) Neue Zombies spawnen (`spawnZombies`). Falls die optionale `locationId` nicht gegeben ist, spawnen so viele Zombies, wie angegeben sind (`amount`) an *jedem* Standort.

(5) Abstimmung über zwei Konsequenzen (`choice`).

Bei allen Konsequenzen (bis auf `choice`) ist in der Konfigurationsdatei spezifiziert, um wie viel sich der entsprechende Wert verändert (`amount`).

Zum Beispiel bedeutet die Konsequenz `"changeFood"` mit `"amount: 2"`, dass dem Nahrungsvorrat zwei Nahrungsmarker hinzugefügt werden, wohingegen bei `"amount: -2"` zwei Nahrungsmarker entfernt werden würden.

Es gilt, dass immer zuerst eine Aktion des Spielers *komplett* vom Server abgehandelt wird, bevor das Schicksal gehandhabt wird. Insbesondere gilt auch, dass falls ein Charakter durch das Ausführen einer Aktion stirbt, danach das Schicksal trotzdem normal abgehandelt wird. Kann die Konsequenz, welche bei der Schicksalskarte angegeben ist nicht eintreten, so passiert nichts.

Die neuen Commands und Events, die durch die Schicksalskarten versendet und empfangen werden müssen, sind in 3.3 spezifiziert.

***Beispiel:* Triggern einer Schicksalskarte mit foodChange**

Es gibt noch drei Essensmarker im Nahrungsvorrat. Für den Spieler Jonathan wird eine Schicksalskarte gezogen, die den Trigger *Anlegen von Ausrüstung* mit der Konsequenz `changefood` um -2 hat.

Da Jonathan das natürlich nicht wissen kann, spielt er nichtsahnend als erstes eine Ausrüstungskarte und zieht seiner Überlebenden Carla `Snowboots` an. Die Aktion *Ausrüstung anlegen* ist somit beendet und das Schicksal nimmt seinen Lauf.

→ Der Server schickt das `Crossroad`-Event an den Spieler und der Nahrungsvorrat wird um 2 verringert, die entsprechenden Events versendet und dann auf das nächste Command von Jonathan gewartet.

3 Technische Details

In diesem Kapitel werden die technischen Details beschrieben, die für Ihre Implementierung relevant sind.

3.1 Zufall

Die Schicksalskarten werden bei der Initialisierung des Spiels nach den Krisenkarten gemischt. Dies geschieht durch folgenden Aufruf:

```
Collections.shuffle(crossroadCards, random);
```

Es wird immer die erste Karte vom gemischten Stapel während des Spiels gezogen. Falls ein Spieler in einer Runde keine Aktion ausführt, die seine Schicksalskarte triggern würde, verfällt sie einfach.

3.2 Konfigurationsdatei

In der Konfigurationsdatei müssen ausreichend Schicksalskarten definiert werden:

$$\#CrossroadsCards \geq maxPlayers * numRounds$$

Unvollständige Beispielskonfigurationsdatei:

```
{
  ...
  "crossroads": [
    {
      "barricaded": {
        "identifier": 1,
        "locationId": 42,
        "consequence": {
          "changeFood": {
            "amount": -2
          }
        }
```

```
        }
      },
      {
        "moved": {
          "identifier": 3,
          "locationId": 1,
          "consequence": {
            "spawnSurvivors": {
              "amount": 2,
              "children": true
            }
          }
        }
      },
      {
        "searched": {
          "identifier": 6,
          "locationId": 2,
          "consequence": {
            "spawnZombies": {
              "amount": 1,
              "locationId": 42
            }
          }
        }
      },
      {
        "wasteChanged": {
          "identifier": 7,
          "amount": 2,
          "consequence": {
            "choice": {
              "consequence1": {
                "spawnZombies": {
                  "amount": 2,
                  "locationId": 42
                }
              },
              "consequence2": {
                "changeFood": {
                  "amount": -3
                }
              }
            }
```

```
            }
          }
        },
        {
          "equip": {
            "identifier": 8,
            "consequence": {
              "changeMoral": {
                "amount": -1
              }
            }
          }
        },
        {
          "barricaded": {
            "identifier": 10,
            "consequence": {
              "changeMoral": {
                "amount": 2
              }
            }
          }
        }
    ],
    ...
}
```

3.3 Neue/Geänderte Kommandos und Events

Tabelle 1: Neue Events

Neue Events der Einzelphase
`Crossroad(int crossroadId)` Mit diesem Event teilt der Server dem Client mit, dass das Schicksal getriggert wurde. Die `crossroadId` gibt hierbei die Id der Schicksalskarte an. Dieses Event wird gesendet nachdem die Aktion, welche der Trigger für die Schicksalskarte war, vollständig vom Server abgehandelt wurde.
`VoteNow()` Mit diesem Event teilt der Server dem Clienten mit, dass dieser für eine der beiden Konsequenzen abstimmen muss. Das VoteNow() ist zu verstehen wie ein ActNow(): es wird immer an den Spieler (in aufsteigender Reihenfolge der `playerId`) geschickt, von dem als nächstes ein Vote erwartet wird.
`VoteResult(boolean answer)` Mit diesem Event teilt der Server dem Clienten mit, welche der beiden Konsequenzen bei der Abstimmung gewonnen hat. Hierbei entspricht die Auswahl der ersten angebotenen Konsequenz dem Parameter `answer = True`. Gibt es einen Gleichstand der Stimmen, wird immer die erste angebotene Konsequenz ausgewählt.

Tabelle 2: Neue Commands

Neue Commands
`Vote(boolean anwswer)` Mit diesem Command teilt der Spieler dem Server mit, welche der beiden angebotenen Konsequenzen von ihm gewählt wird.

4 Tests

In diesem Kapitel geht es um die erforderlichen Tests in der Einzelphase des Software-Praktikums.

- Sie müssen Ihre eigene Implementierung sinnvoll testen und Ihre eigenen Tests bestehen. Schreiben Sie dazu sinnvolle Unit- und Integrationstests für die relevanten Teile Ihrer Implementierung der Einzelphase. Wir werden Ihnen auch (ähnlich wie in der Gruppenphase) ein Framework für Systemtests zur Verfügung stellen, mit dem Sie Systemtests erstellen können.
- Ihre Implementierung muss von uns erstellte Systemtests bestehen (welche wir in regelmäßigen Abständen auf Ihrer Implementierung laufen lassen werden).