Programmierpraktikum [WS23]

Aufgabenstellung

Geöffnet: Monday, 10. July 2023, 09:30

Fällig: Tuesday, 30. January 2024, 12:00

Aufgabenstellung *Hase und Igel*

Hase und Igel ist ein klassisches Brettspiel (übrigens das erste "Spiel des Jahres" aus dem Jahr 1979), in dem die Mitspieler ein Rennen gegeneinander veranstalten und möglichst schnell ins Ziel kommen wollen. Aber nur mit der richtigen Strategie kann dies gelingen, denn stumpfes Losrennen führt nicht weit...

Zur Veranschaulichung gibt es folgendes Beispielprogramm. Bindend bleibt aber in jedem Fall diese Aufgabenstellung.

Regeln

Für das Spiel müssen die Spielregeln des Originalspiels umgesetzt werden (lest Euch diese unbedingt durch, BEVOR Ihr diese Aufgabenstellung durcharbeitet!). Zudem gelten folgende Abweichungen/Ergänzungen:

- die genaue Verteilung der Karottenkarten (1er, 5er, 10er, ...) ist egal, wir rechnen stets nur mit der Summe aller Karotten eines Spielers
- es beginnt nicht der jüngste Spieler, sondern die Reihenfolge der Teilnehmer ist stets Blau, Gelb, Grün, Orange, Rot, Weiß (bzw. nur so viele davon, wie am Spiel teilnehmen)
- bei der Hasenkarte "Rücke sofort um eine Position vor!" darf man ggf. nur dann direkt ins Ziel ziehen, wenn man keine Salate mehr hat und die restliche Karottenzahl dies auch auf normalem Weg zulassen würde, ansonsten bleibt man auf dem Hasenfeld stehen. Ist man schon an 1. Position von allen Spielern im Feld (die bereits im Ziel befindlichen zählen hier nicht mit), passiert nichts (Konkretisierung der Spielregeln)
- die Sonderregeln der "Version für 2 Spieler" aus den Spielregeln wird nicht umgesetzt

Einstellungen

Vor Spielbeginn müssen lediglich die Anzahl der teilnehmenden Spieler (von 2 bis 6) und ihre Namen (alle unterschiedlich und nicht leer) eingestellt werden. Auch während eines laufenden Spiels kann der Nutzer jederzeit ein neues Spiel (ggf. mit mehr oder weniger Spielern und anderen Namen) beginnen.

Darstellung / Verhalten

Das Spielbrett nimmt mit Abstand die größte Fläche des Fensters ein. Darauf werden zu jedem Zeitpunkt alle teilnehmenden Spielsteine dargestellt - entweder am Start, auf einem der 63 Felder oder im Ziel.
Bei einer Darstellung am Start sollen die Steine dort wie im Beispielprogramm zu sehen etwas überlappend dargestellt werden nach ihrer Reihenfolge (von Blau bis Weiß). Im Ziel hingegen soll die (ebenfalls überlappend darzustellende) Reihenfolge nach dem zeitlichen Zieleinlauf erfolgen. Der Spieler ganz links hat am Ende also das Spiel gewonnen.
Die Spielsteine sollen dabei stets transparent (also "in Kreisform") auf den Untergrund gezeichnet werden. Der Stein des aktuellen Spielers wird immer über allen anderen gezeichnet - auch bei der Laufanimation, s.u. - und bekommt zudem zur besseren Sichtbarkeit einen roten Ring von einigen Pixeln Stärke.

Bewegt der aktuelle Spieler seine Maus über die Felder auf dem Spielplan, so soll das jeweils "überfahrene" Feld einen grünen oder roten Kasten bekommen zur Verdeutlichung, ob der Spieler auf dieses Feld ziehen könnte oder nicht. Hierbei sind sämtliche Spielregeln (insb. zu noch vorhandenen Karotten und zu Igelfeldern!) zu beachten! Klickt er ein "grünes" Feld an, wird der Spielstein animiert über alle evtl. noch dazwischenliegenden Felder auf das ausgewählte Zielfeld bewegt. Die Animation soll hierbei jeweils linear zwischen den Mittelpunkten von jeweils zwei Nachbarfeldern erfolgen (siehe Hilfe).

Zusätzlich zu den 63 Feldern kann der Spieler auf das Ziel klicken (das aber keinen grünen oder roten Rahmen bekommt), um seinen Spielstein ins Ziel laufen zu lassen (falls die Regeln dies zulassen). Wenn auch der letzte Spieler das Ziel erreicht hat, soll eine Meldung über den Gewinner des Spiels informieren.

Über ein Menü soll ein neues Spiel begonnen, ein Spiel geladen oder gespeichert oder aber das Spiel komplett beendet werden können. Die im Beispielprogramm dafür vorhandenen Buttons müssen also nicht angelegt werden.

KI

Eine KI muss nicht implementiert werden. Es nehmen nur menschliche Spieler teil.

Log

Während eines laufenden Spiels soll ein umfangreiches Ereignislog erstellt und in eine Textdatei im selben Ordner wie die jar geschrieben werden. Dieses ist von den Einträgen her so aufzubauen, dass der Spielverlauf vollständig und möglichst einfach lesbar nachvollzogen werden kann, um z.B. in Fehlersituationen den Zustand vor dem Fehler nachstellen zu können. Eine bestehende Logdatei wird jeweils beim Beginn eines neuen Spiels überschrieben.

Spielstand

Es soll möglich sein, den aktuellen Spielstand zu speichern und auch wieder zu laden. Dabei sollen Textdateien mit der Endung "json" verwendet werden, die folgenden Aufbau haben:

```
{
  "currPlayer": 2,
  "onTarget": [],
  "players": [
    {"name": "Anton", "suspended": true, "eatsSalad": true, "position": 1, "carrots": 67, "salads": 3},
    {"name": "Berta", "suspended": false, "eatsSalad": false, "position": 2, "carrots": 65, "salads": 3},
    {"name": "Cäsar", "suspended": false, "eatsSalad": false, "position": 0, "carrots": 68, "salads": 3},
    {"name": "Dora", "suspended": false, "eatsSalad": false, "position": 0, "carrots": 68, "salads": 3}
  ]
}
```

- "currPlayer" beschreibt, welcher Spieler am Zug ist (eine ganze Zahl von [Änderung vom 29.8.] ~~1 bis zur Anzahl der Teilnehmer~~ 0 bis zur Anzahl der Teilnehmer - 1, "Cäsar" ist also am Zug)
- "onTarget" enthält, welche Spieler schon im Ziel angekommen sind und in welcher Reihenfolge. Die Spieler sind dabei ab 0 durchnummeriert. Beim Eintrag *[4, 1, 2]* wären also schon 3 Spieler im Ziel und der 5. Spieler wäre der Gewinner. Keine Ziffer darf doppelt vorkommen und auch nur solche Ziffern, die zur Anzahl der teilnehmenden Spieler passt. Ist noch kein Spieler im Ziel, bleibt das Array leer.
- "players" enthält ein Array mit den teilnehmenden Spielern, muss also 2 bis 6 Einträge enthalten. Zu jedem Spieler gibt es folgende Info:
  - "name": einen Namen (ein nicht leerer String, kein Name darf doppelt vorkommen)
  - "suspended": ob er gerade aussetzen muss
  - "eatsSalad": ob er gerade einen Salat frisst
  - "position": seine aktuelle Position (0 = Start, 1..63 für die einzelnen Felder, 64 = Ziel)
  - "carrots": die Anzahl seiner noch verfügbaren Karotten (ganze positive Zahl ab 0)
  - "salads": die Anzahl seiner noch vorhandenen Salate (von 0 bis 3)

Achtet vor allem beim Laden darauf, alle denkbaren Fehler in den Dateien abzufangen und die in dieser Auflistung vorgegebenen erlaubten Werte genau abzuprüfen! Der obige Beispielspielstand, bei dem vier Spieler teilnehmen, von denen zwei noch am Start und zwei schon auf den Positionen 1 und 2 im Feld sind (der 3. also am Zug ist), wobei der 1. aussetzen muss, kann zur Kontrolle dienen.
(Auch das Beispielprogramm kann json-Dateien lesen/schreiben. Da hierin keine hilfreichen Bibliotheken eingesetzt wurden, müssen die Angaben allerdings in der selben Reihenfolge wie oben stehen.)

Vorgaben

Beigefügte Dateien und Angaben könnt Ihr gerne in Eurer Lösung oder bei der Entwicklung nutzen:

- Bild des Spielplans (1000 * 750), Bild des Spielplans mit Kennzeichnung der Feldindizes
- Bilder der 6 Spielsteine mit ihren RGB-Werten
- Meldungen der 12 Hasenkarten (einige Karten gibt es mehrfach)
- Koordinaten der Mittelpunkte der 63 Felder auf dem Spielplan in Originalgröße (ein Feld ist dort 60x60 Pixel groß)

Legt Ihr die json-Dateien im Projekt als Resource im Projekt ab, so ist keine umfangreiche Fehlerprüfung beim Einlesen notwendig, da die Dateien nicht vom Nutzer manipuliert werden konnten.

Erwartete Inhalte für den Zwischenstand

Bei der Abnahme des Zwischenstands müssen mindestens folgende Vorgaben erfüllt werden:

GUI

- Alle für die Bedienung des Spiels notwendigen Komponenten müssen in einer FXML-Datei vorhanden, aber noch nicht nach Programmstart bedienbar sein. Komponenten, die dynamisch erstellt werden, müssen nach Programmstart bereits sichtbar sein.
- Alle (auch optional) sichtbaren Anzeigen müssen dargestellt werden.
- Wird ein zusätzliches Fenster verwendet, so muss dies mit seinen Komponenten vorhanden sein, aber noch nicht in einem Programmfluss geöffnet werden.
- Der Spielplan soll bei Programmstart angezeigt werden. Für den Zwischenstand sollen die 6 Spielsteine auf den ersten 6 Spielfeldern platziert sein.
- Das Spielfeld muss den maximalen freien Raum einnehmen und sich in seiner Größe anpassen, wenn die Fenstergröße verändert wird. Auch die Spielsteine müssen ihre Größe entsprechend verändern. Die Bilder müssen ihre Proportionen nicht behalten.

Code

Es müssen mindestens Attribute und/oder Klassen existieren für

- ein Spiel
- die Spieler
- den Spielplan

Es muss mindestens jeweils einen Konstruktor geben,

- der aus dem UserInterfaceController heraus aufgerufen wird und ein neues Spiel erzeugt, wie es nach der Eingabe der Spielerinformationen passieren würde.
- der für Testzwecke ein mitten im Spielgeschehen befindliches Spiel erzeugt und dafür alle Angaben entgegennimmt. Hierfür genügt nicht die Übergabe eines Dateinamens!

Klassen müssen die wichtigsten (und die für die Tests notwendigen) Methoden mit ihren Signaturen enthalten (aber noch nicht unbedingt implementiert sein).

Tests

Alle Tests müssen kompilierbar sein, dürfen aber noch fehlschlagen:

- Darf der aktuelle Spieler ein bestimmtes Feld betreten?
  - Ein vorausliegendes Feld (mit größerem Index als dem Feldindex des Spielers).
  - Ein Feld, für das die Karottenanzahl nicht ausreicht.
  - Ein vorausliegendes Igelfeld.
  - Ein zurückliegendes Igelfeld.
  - Ein Salatfeld mit Salat im Besitz des Spielers.
  - Ein Salatfeld, ohne dass der Spieler noch Salate besitzt.
- Wird der korrekte Feldindex ermittelt?
  - vom nächsten freien Karottenfeld
    - wenn das nächste Karottenfeld belegt ist
    - wenn das nächste Karottenfeld frei ist
  - beim Zurückfallen um eine Position
    - Der vorige Spieler steht auf Feldindex 1.
    - Der vorige Spieler steht auf Feldindex 2.
  - beim Vorrücken um eine Position
    - Der nächste Spieler steht auf Feldindex 61.
    - Der nächste Spieler steht auf Feldindex 63.
- Darf der aktuelle Spieler ins Ziel?
  - Als Erster und darf / darf nicht wegen zu viel Karotten / darf nicht wegen Salatbesitzes.
  - Als Zweiter und darf / darf nicht wegen zu viel Karotten / darf nicht wegen Salatbesitzes.

Diese Anforderungen stellen ein Minimum dar, um auch von Teilnehmern mit knapper Zeit in der vorlesungsfreien Zeit erfüllbar zu sein. Sinnvoll ist es, bis zum Beginn der Vorlesungszeit sehr viel weiter gekommen zu sein, da in der Vorlesungszeit meist andere Projekte Zeit beanspruchen.

Hilfe

- Legt gerne eine `Pane` in das CENTER einer `BorderPane`. Die `Pane` kann sowohl das Bild des Spielplanes, als auch die Bilder der Spielsteine und ein Markierungsrechteck als Kindknoten aufnehmen.
- Verwendet Ihr eine `StackPane` zum Aufnehmen von Spielplan- und Spielsteinbildern, so achtet darauf, `getManaged(false)` für die Spielsteinbilder zu setzen, da die `StackPane` ansonsten keine explizite Positionierung zulässt.
- Die Größe eines Spielsteinbildes kann an die Größe des Spielplanbildes gebunden werden:
  `imgViewPl.fitWidthProperty().bind(imgVwGameBoard.fitWidthProperty().divide(BOARD_WIDTH).multiply(FIELDSIZE));`
  mit `imgViewPl` als ImageView mit einem Spielsteinbild,
  `imgVwGameBoard` als ImageView mit dem Spielplanbild,
  `BOARD_WIDTH` als Breite des Spielplanbildes im Original (1000) und
  `FIELDSIZE` als Breite eines Spielfeldes in Originalgröße (60)
- Zur Positionierung des Spielsteinbildes kann die `xProperty` an die aktuelle Spielplanbreite gebunden werden.
- Die **Koordinaten** eines *MouseEvents* beziehen sich auf die Komponente, die das *MouseEvent* entgegennimmt. Ist es das `ImageView` des Spielbretts, so passen die Koordinaten zu den Koordinaten der Felder aus der Datei. Bewegt sich die Maus über das `ImageView` eines Spielsteins, muss dieses einen entsprechenden Eventhandler implementieren und darin die Koordinaten umrechnen.
- Für die Bewegung eines Spielsteins empfiehlt sich eine Translate-Transition.

Viel Erfolg!

Feldmittelpunkte.json

Hasenmeldungen.json

HaseUndIgelBrettspiel.jpg

HaseUndIgelSpielplan.png

HaseUndIgelSpielplanMitIndize...

Spielanleitung_Hase_und_Igel.p...

Spielstand.json

Spielsteine.zip

Abgabestatus

Abgabestatus	Diese Aufgabe benötigt keine Online-Abgabe
Bewertungsstatus	Nicht bewertet
Verbleibende Zeit	Verbleibend: 108 Tage
Zuletzt geändert	-
Abgabekommentare	Kommentare (0)

◄ Anmeldung Direkt zu: Aufgabenvorstellung im Hörsaal ►

https://www.fh-wedel.de/