Praktische Informatik I | WiSe 2023/24
Tutor:in: Pascal Himmelmann | Bearbeiter:in: Öykü Koç

Übungsblatt 12

Abgabe: 28.01.2024

Das Ziel

Diese Aufgabe soll den Beginn der Multiplayer-Version unseres Spiels markieren.

Die Bewegung einer Figur in einer Instanz des Spiels wird auf eine Figur in einer zweiten Instanz übertragen, die über das Netzwerk verbunden ist. Alle Aktionen des steuernden Charakters werden auf den gesteuerten Charakter übertragen, so dass sich beide Charaktere auf die gleiche Weise verhalten.

Die Netzwerkverbindung wird zwischen einer Instanz der Klassen RemotePlayer und ControlledPlayer hergestellt. Das Spiel wird gestartet, indem eine Kopie des Projekts und zwei verschiedene BlueJ-Instanzen von sich selbst ausgeführt werden. Beim Ausführen der ersten Instanz wird der IP-Adressparameter auf null oder "" gesetz. Beim Ausführen der zweiten Instanz wird der IP-Adressparameter auf "127.0.0.1" oder "localhost" gesetzt. Die Portnummer muss in beiden Beispielen identisch sein. Der Wert 5000 kann als Portnummer verwendet werden.

Aufgabe 1 Fernsteuernd

Die RemotePlayer-Klasse ist von der Player-Klasse abgeleitet.

```
12 class RemotePlayer extends Player
```

Zusätzlich zu den Parametern der Grundklasse erhält der Konstruktor dieser Klasse die IP-Adresse und die Portnummer des Computers des Zeichners als Parameter

```
26     RemotePlayer(final int x, final int y, final int rotation, final Field field, final
           String serverAddress, final int serverPort)
```

Das Kommunikationsnetzwerk wird über den im Konstruktor erstellten Socket vorbereitet. Wenn die ControlledPlayer-Instanz für die Netzwerkumgebung geöffnet ist, werden IP-Adresse und Portnummer veröffentlicht, und die RemotePlayer-Verbindung zur entsprechenden Instanz kann hergestellt werden. Bei Ausnahmen, die während der Verbindungsphase auftreten, wird eine Fehlermeldung zurückgegeben und der Player wird ausgeblendet.

```
26     RemotePlayer(final int x, final int y, final int rotation, final Field field, final
           String serverAddress, final int serverPort)
27     {
28         super(x, y, rotation, field);
29         sleep(300);
30         try{
31             socket = new Socket(serverAddress, serverPort);
32         }
33         catch(final IOException e){
34             System.err.println(serverAddress + ":" + serverPort + " Verbindung zu
                       Serveradresse und Port fehlgeschlagen!");
35
36             setVisible(false);
37         }
38     }
```

Diese Klasse überschreibt die act()-Methode. Die Richtungs- und Bewegungsaktionen des Spielers werden über die Tastatur ausgeführt. Die Richtungsinformation des Spielers wird dann über den

Socket an die ControlledPlayer-Instanz weitergegeben. Die Richtungsinformation besteht aus den Werten 0, 1, 2 und 3. Die Methode flush() wird aufgerufen, um die Richtungsdaten sofort in den Puffer zu übertragen. Bei Ausnahmen, die während der Übertragungsphase auftreten, wird eine Fehlermeldung zurückgegeben und der Spieler ausgeblendet.

```
45      @Override
46      void act(){
47          super.act();
48
49          try{
50              socket.getOutputStream().write(getRotation());
51              socket.getOutputStream().flush();
52          }
53          catch(final IOException e){
54              System.err.println("Richtungsangaben wurden nicht gesendet!");
55              setVisible(false);
56          }
57      }
```

Diese Klasse überschreibt die Methode setVisible(final boolean visible). Wenn der Spieler unsichtbar ist, wird das Spiel beendet, indem der Socket geschlossen wird, falls er offen ist.

```
65      @Override
66      public void setVisible(final boolean visible)
67      {
68          super.setVisible(visible);
69
70          if(!visible && socket != null) {
71              try {
72                  socket.close();
73              }
74              catch(final IOException e){
75                  System.err.println("Die Netzwerkverbindung konnte nicht geschlossen
                            werden!");
76              }
77          }
78      }
79  }
```

Aufgabe 2 Ferngesteuert

Eine ControlledPlayer-Klasse ist von der Player-Klasse abgeleitet

```
12  class ControlledPlayer extends Player
```

Der Konstruktor dieser Klasse erhält als Parameter zusätzlich zu den Parametern der Basisklasse die Portnummer des Ports, auf dem die Netzwerkverbindung genutzt werden soll.

```
31      ControlledPlayer(final int x, final int y, final int rotation, final Field field,
                final int serverPort)
```

Im Konstruktor werden der Server und das Kommunikationsnetzwerk durch ServerSocket und socket vorbereitet. Die ControlledPlayer-Instanz öffnet die Netzwerkumgebung über die als Parameter angegebene Portnummer und nimmt die Verbindung von der RemotePlayer-Instanz an. Sie gibt eine Fehlermeldung für die Ausnahme zurück, die während der Verbindungsphase auftritt, und blendet das Zeichen aus.

```
31      ControlledPlayer(final int x, final int y, final int rotation, final Field field,
                final int serverPort)
32      {
33          super(x, y, rotation, field);
34
35          try{
36              serverSocket = new ServerSocket(serverPort);
37              socket = serverSocket.accept();
```

```
38          }
39          catch(final IOException e){
40              System.err.println(serverPort + " auf dem mit einer Nummer versehenen
                        Anschluss, konnte die Verbindung nicht hergestellt werden.");
41              setVisible(false);
42          }
43      }
```

Diese Klasse überschreibt die act()-Methode. Die Richtungsinformationen des Spielers, die mit der RemotePlayer-Instanz über das Netzwerk gesendet werden, werden über den Socket gelesen und dem Charakter in der ControlledPlayer-Instanz zugewiesen. Entsprechend der vom Spieler empfangenen Richtungsinformation erhält die Figur die Richtung und Bewegung. Für Ausnahmen, die auftreten, wenn die Verbindung beendet wird oder die Richtungsinformationen nicht gelesen werden können, werden Fehlermeldungen zurückgegeben, und der Charakter wird ausgeblendet. Wenn die Verbindung beendet wird, wird der Wert als (-1) gelesen.

```
53      @Override
54      void act(){
55         try{
56              int direction = socket.getInputStream().read();
57
58              if (direction != -1){
59                  setRotation(direction);
60                  setLocation(getX() + offsets[getRotation()][0], getY() + offsets[
                            getRotation()][1]);
61                  playSound("step");
62                  sleep(200);
63
64              }
65              else{
66                  System.err.println("Die Netzwerkverbindung wurde abgebrochen.");
67                  setVisible(false);
68              }
69          }
70          catch(final IOException e) {
71              System.err.println("Es wurde keine Information über die Richtung des Spielers
                        empfangen.");
72              setVisible(false);
73          }
74      }
```

Diese Klasse überschreibt die Methode setVisible(final boolean visible). Wenn die Figur unsichtbar ist, wird das Spiel beendet, indem der Socket geschlossen wird, falls er offen ist.

```
82      @Override
83      public void setVisible(final boolean visible)
84      {
85          super.setVisible(visible);
86
87          if(!visible && socket != null){
88              try{
89                  socket.close();
90              }
91              catch(final IOException e){
92                  System.err.println("Die Netzwerkverbindung konnte nicht geschlossen
                            werden!");
93              }
94          }
95      }
96  }
```

Aufgabe 3 Spielend

Zusätzlich zum Basisparameter nimmt der Konstruktor der Klasse Level die Parameter IP-Adresse und Portnummer des Computers des kontrollierten Charakters entgegen.

```
57      Level(final String fileName, final String serverAddress, final int serverPort)
```

Sie erstellt eine Spielerinstanz entsprechend den IP-Adress- und Portnummern-Daten aus der Hauptmethode. Auf diese Weise hat das Spiel eine parametrische und flexible Struktur.

Wenn die IP-Adresse vom Typ String, die als Parameter der Konstruktormethode übergeben wird, null oder leer ist; Eine Spielfigur des kontrollierten Typs wird von der Klasse ControlledPlayer erstellt.

Wenn die IP-Adresse vom Typ String, die als Parameter der Konstruktormethode übergeben wird, voll ist; Eine Spielfigur vom Typ Controller wird von der Klasse RemotePlayer erstellt

```
85          if(serverAddress == null || serverAddress == "")
86              player = new ControlledPlayer(-1, 0, 0, field , serverPort);
87          else
88              player = new RemotePlayer(-1, 0, 0, field , serverAddress , serverPort);
```

Die Parameter IP-Adresse und Portnummer werden der Hauptmethode der Klasse PI1Game hinzugefügt, um sie an die Klasse Level zu senden

```
21      static void main(final String serverAddress, final int serverPort)
```

Die entsprechenden Parameter werden als Parameter an die Klasse Level gesendet, wenn eine Instanz der Klasse Level erzeugt wird.

```
21      static void main(final String serverAddress, final int serverPort)
22      {
23          // Den Level erzeugen
24          final Level level = new Level("levels/1.lvl", serverAddress, serverPort);
25
26          // Die Hauptschleife des Spiels
27          // Die durch den Datenstrom transportierten Werte werden überhaupt nicht
28          // verwendet. Einzig wichtig ist, dass keine mehr erzeugt werden, sobald
29          // die Bedingung, die als zweiter Parameter formuliert ist, falsch wird.
30          // Dann endet die Verarbeitung.
31          Stream.generate(() -> level.getActors())
32          .takeWhile(actors -> actors.get(0).isVisible())
33          .forEach(actors -> actors.forEach(actor -> actor.act()));
34          Stream.iterate(level.getActors(), actors -> actors.get(0).isVisible(), actors ->
                    actors)
35          .forEach(actors -> actors.forEach(actor -> actor.act()));
36
37          level.hide();
38      }
```