Praktische Informatik I
TutorIn: Berghöfer/Senger

WS 2012/13
Gruppe: 8

Beate Ruffer
Mohamadreza Khostevan
Leopold Schulz-Hanke

Übungsblatt 10

Aufgabenlösung
Abgabe: 27.01.2013

Aufgabe 1 Single Player (30%)

Um das Visuelle von der Berechnung zu trennen, haben wir diese Aufgabe nach genau dem selben Schema umgesetzt, wie es im Aufgabenblatt vorgeschlagen wurde. Unsere Klasse `LocalClient` ist von nun an unsere einzige angezeigte Welt.
Da wir vom Client und vom Server, jeweils nur eine Instanz benötigen, diese aber von überall erreichbar sein muss, können wir diese im Weltenverwalter als `static` deklarieren.

```
24 /**
25      * Server
26      */
27     public static HeldInServer heldInServer;
28 
29     /**
30      * Client
31      */
32     public static LocalClient localClient;
33 
34     /**
35      * Aktuelle Welt
36      */
37     public static Class currentWorld;
38 
39     /**
40      * Konstruktor für Objekte der Klasse WorldManager. Hier werden alle
        gebrauchten Welten
41      * instanziert und zur Weltenliste myWorlds hinzugefügt. Beginnende Welt
        wird gesetzt.
42      */
43     public WorldManager()
44     {
45         super(1000,700,1);
46         localClient = new LocalClient(this);
47         heldInServer = new HeldInServer(localClient);
48         currentWorld = Start.class;
49         myWorlds = new ArrayList<World>();
50 
51         myWorlds.add(new Start());
52         myWorlds.add(new Ocean1());
53         myWorlds.add(new Ocean2());
54         myWorlds.add(new Ocean3());
55         myWorlds.add(new Cave1());
56         myWorlds.add(new Cave2());
57         myWorlds.add(new Cave3());
58         myWorlds.add(new Cave4());
59         myWorlds.add(new Aquarium());
60 
```

```
61          mySurfaces = new ArrayList<Integer>();
62          mySurfaces.add(260);//start1
63          mySurfaces.add(260);//Ocean1
64          mySurfaces.add(260);//Ocean2
65          mySurfaces.add(260);//Ocean3
66          mySurfaces.add(1);//Cave1
67          mySurfaces.add(1);//Cave2
68          mySurfaces.add(1);//Cave3
69          mySurfaces.add(1);//Cave4
70          mySurfaces.add(255);//Aquarium
71          surfaceY =(mySurfaces.get(0));
72
73          localClient.setBackground(myWorlds.get(0).getBackground());
74
75          Greenfoot.setWorld(localClient);
76      }
```

Im Konstrkutor werden die Instanzen von Client und Server erzeugt. Mit Hilfe von `currentWorld` können wir festlegen, welche die im LocalClient angezeigte Welt sein soll. In Zeile 73 setzen wir den Hintergrund der ersten Welt als Hintergrund von LocalClient. Zeile 75 setzt LocalClient zur aktuellen Welt. Dies wird sich im Spielverlauf nicht mehr ändern. Der Weltenverwalter, bekommt eine act() Methode, die von der act() Methode des LocalClients aufgerufen wird. Da die einzige angezeigte Welt, LocalClient ist - und nur Avatare enthält, werden die act() Methoden in den realen Welten nicht mehr aufgerufen. Diese act Methode, geht nun alle Welten durch, und führt für jeden Actor in ihnen die act() Methode manuell aus.

```
135 public void act(){
136         for(World w : myWorlds){
137             List<Actor> actors = w.getObjects(null);
138             for (Actor actor : actors){
139                 actor.act();
140             }
141         }
142     }
```

Zunächst brauchen wir eine Möglichkeit, einige Methoden unserer Actors “mitzuhören“. Alle Actors haben die Superklasse `ServerActor`. In dieser Klasse werden alle “mitzuhörenden“ Methoden überschrieben.
Die Klasse ist *abstract* Definiert - wir sichern also zu, dass wir keine Instanzen der Klasse erzeugen werden.

```
12 public abstract class ServerActor extends Actor
13 {
14     public void setLocation(int x, int y){
15         WorldManager.heldInServer.setLocation(this, x, y);
16         super.setLocation(x, y);
17     }
18
19     public void setImage(String filename){
20         WorldManager.heldInServer.setImage(this, filename);
21         super.setImage(filename);
22     }
23
24     public void setImage(GreenfootImage image){
25         WorldManager.heldInServer.setImage(this, image);
26         super.setImage(image);
27     }
```

```
28
29      public void setRotation(int rotation){
30          WorldManager.heldInServer.setRotation(this, rotation);
31          super.setRotation(rotation);
32      }
33
34      public void setText(String string, int x, int y){
35          WorldManager.heldInServer.setText(this, string, x, y);
36      }
37
38      public void setWorld(World world){
39          WorldManager.heldInServer.setWorld(world);
40      }
41  }
```

Jedes der Methoden, ruft die dazugehörige Methode im Server auf um diesen von der Aktion zu "berichten". Anschließend wird die (ursprüngliche) Methode der Superklasse aufgerufen, damit die Aktion auch durchgeführt wird. Bei *setText()* haben wir darauf verzichtet, da grafische Änderungen in der Berechnungswelt uns sowieso nicht sichtbar werden.
Der Server seinerseits verfügt über eine HashMap Actors als Key und Integers als Value um den Actors eine ID zuweisen zu können. Anhand dieser ID können die passenden Avatare im Client angesprochen werden.

```
 9  public class HeldInServer extends Server
10  {
11      private HashMap<Actor, Integer> actorMap;
12      private LocalClient client;
13      private int idCounter = 0;
14
15      public HeldInServer(LocalClient client){
16          actorMap = new HashMap();
17          this.client = client;
18      }
19
20      /**
21       * Stellt sicher, dass ein Eintrag mit zu der id bereits in der HashMap
          existiert.
22       */
23      private void ensureMapContains(Actor actor){
24          if(!actorMap.containsKey(actor)){
25              actorMap.put(actor, idCounter);
26              idCounter++;
27          }
28      }
29
30      /**
31       * Die Avatare werden aus dem Client entfernt
32       * Anschließend werden neue Avatare nach dem Vorbild der neuen Welt
          erschaffen.
33       */
34      public void setWorld(World world){
35          List<Actor> actors = client.getObjects(null);
36          for (Actor a : actors){
37              client.removeObject(a);
38          }
39          client.setBackground(world.getBackground());
40          actors = world.getObjects(null);
41          for (Actor a : actors){
42              client.addObject(actorMap.get(a), a.getX(), a.getY());
```

```
43          }
44      }
45
46      public void removeObject(Actor actor){
47          ensureMapContains(actor);
48          client.removeObject(actorMap.get(actor));
49      }
50
51      public void addObject(Actor actor, int x, int y){
52          ensureMapContains(actor);
53          client.addObject(actorMap.get(actor), x, y);
54      }
55
56      public void setLocation(Actor actor, int x, int y){
57          ensureMapContains(actor);
58          client.setLocation(actorMap.get(actor), x, y);
59      }
60
61      public void setRotation(Actor actor, int rotation){
62          ensureMapContains(actor);
63          client.setRotation(actorMap.get(actor), rotation);
64      }
65
66      public void setImage(Actor actor, String filename){
67          ensureMapContains(actor);
68          client.setImage(actorMap.get(actor), filename);
69      }
70
71      public void setImage(Actor actor, GreenfootImage image){
72          ensureMapContains(actor);
73          client.setImage(actorMap.get(actor), image);
74      }
75
76      public void setText(Actor actor, String text, int x, int y){
77          ensureMapContains(actor);
78          client.setText(actorMap.get(actor), text, x, y);
79      }
80
81      public void setBackground(String filename){
82          client.setBackground(filename);
83      }
84
85      public boolean isKeyDown(String key){
86          return client.isKeyDown(key);
87      }
88
89      public void playSound(String filename){
90          Greenfoot.playSound(filename);
91      }
92  }
```

Mit `ensureMapContains()` wird immer sichergestellt, dass ein entsprechender Eintrag schon existiert. Der Client wird in jeder Methode über die jeweilige Aktion informiert.
Auch der Client hat seinerseits eine HashMap. Diese allerdings mit der ID als Key und einem Avatar als Value.

```
 9  public class LocalClient extends World implements Client
10  {
11      public HashMap<Integer, Avatar> avatarMap;
12      private WorldManager worldManager;
```

```
13
14      /**
15       * Konstruktor für Objekte der Klasse LocalClient.
16       */
17      public LocalClient(WorldManager worldManager)
18      {
19          // Erzeuge eine neue Welt mit 600x400 Zellen und einer Zellgrö ß e von
              1x1 Pixeln.
20          super(1000, 700, 1);
21          avatarMap = new HashMap();
22          this.worldManager = worldManager;
23      }
24
25      /**
26       * Stellt sicher, dass ein Eintrag mit zu der id bereits in der HashMap
           existiert.
27       */
28      private void ensureMapContains(int id, int x, int y){
29          if(!avatarMap.containsKey(id)){
30              avatarMap.put(id, new Avatar());
31          }
32      }
33
34      /**
35       * Führt im worldManager act() auf, um die act() Methoden aller Actors
           aller Welten auszuführen.
36       */
37      public void act(){
38          worldManager.act();
39      }
40
41      public boolean isKeyDown(String key){
42          return Greenfoot.isKeyDown(key);
43      }
44
45      public void setText(int id, String text, int x, int y){
46          ensureMapContains(id, 0, 0);
47          avatarMap.get(id).setText(text, x, y);
48      }
49
50      public void setImage(int id, String filename){
51          ensureMapContains(id, 0, 0);
52          avatarMap.get(id).setImage(filename);
53      }
54
55      public void setImage(int id, GreenfootImage image){
56          ensureMapContains(id, 0, 0);
57          avatarMap.get(id).setImage(image);
58      }
59
60      public void setRotation(int id, int rotation){
61          ensureMapContains(id,0,0);
62          avatarMap.get(id).setRotation(rotation);
63      }
64
65      public void setLocation(int id, int x, int y){
66          ensureMapContains(id, x, y);
67          avatarMap.get(id).setLocation(x, y);
68      }
69
70      public void removeObject(int id){
71          ensureMapContains(id,0,0);
```

```
72          removeObject(avatarMap.get(id));
73      }
74
75      public void addObject(int id, int x, int y){
76          ensureMapContains(id, x, y);
77          addObject(avatarMap.get(id), x, y);
78      }
79  }
```

Hier werden nun alle "mitgehörten" Aktionen im LocalClient - also in der sichtbaren Welt - durchgeführt. Auch hier wird immer mit `ensureMapContains()` vorher sichergestellt, dass ein entsprechender Eintrag existiert.

Tests Zunächst sollte sich das Spiel komplett so verhalten, wie vor dieser Implementierung - was sie auch tut. Man kann das U-Boot steuern, bleibt vor Hindernissen stehen, kann Objekte ins Inventar legen und wieder ablegen. Auch ein Weltenwechsel ist möglich.
Um den Erfolg der Implementierung zu testen muss haben wir das Spiel begonnen, dann pausiert. Nun können wir auf der Welt und den einzelnen Objekte mit der rechten Maustaste *Inspizieren* ausführen. Uns wird nun angezeigt, dass die Welt eine Instanz von LocalClient ist und die Objekte Instanzen von Avatar. Auch nach einem Weltenwechsel sind keine Instanzen einer anderen Klasse zu sehen.
Desweiteren haben wir unseren Helden manuell mit der Maustaste verschoben. Wenn wir nun das Spiel weiterführen, springt das Uboot an seine ursprüngliche stelle, da dieser in der Berechnungswelt ja nicht verschoben wurde.