TTV-Lab2

Übersicht

Strategie

Unsere Anwendung ist hinsichtlich der Strategie modular aufgebaut, sodass unterschiedliche Strategien entwickelt und vor einem Spiel ausgetauscht werden können. Wir unterscheidern hierbei zwischen zwei Strategiearten die wir umsetzen müssen. Zum einen eine Strategie zum Platzieren der Schiffe und zum anderen eine für die Auswahl eines Schiffes auf das geschossen werden soll.

Um dies umzusetzen haben wir eine abstrakte Klasse "Strategy" entworfen, welches die beiden oben genannten Strategiarten durch zwei Methoden definiert. Strategien die diese Klasse implementieren müssen somit auch die Strategien umsetzen.

StrategyOne

Schiffauswahl (Ziel)

Durch die empfangenen Broadcasts sind wir in der Lage in unserer Anwendung alle erfolgreich und missglückten beschossenen Ziele eines Spielers zu versenken zu zählen. Diese Information machen wir uns zunutze, indem wir immer den Spieler wählen der bereits die meisten Schiffe verloren hat. Gibt es mehrere nehmen wir den zuletzt betrachteten. Gibt es keinen einzigen solchen Spieler wählen wir einen Spieler aus bei dem die meisten missglückten beschossenen Ziele aufgetreten sind. Gibt es mehrere nehmen wir den zuletzt betrachteten.

Wenn wir der erste Spieler sind der schießen darf, erfolgt die Auswahl des Ziels zufällig. Durch die laufenden Broadcast erhalten wir Informationen über Spieler die beschossen wurden, deren Felder, die beschossen wurde und ob dabei ein Schiff versenkt wurde. Die Information, welcher Spieler wo getroffen wurde, wird in der Map<ID, ArrayList> getHitEnemyShips gespeichert. Die ArrayList von IDs innerhalb der Map beinhaltet zum einen die Information, wie oft ein Schuss auf einen Gegner erfolgreich war, sowie die Position des Schiffes. Die Information, welcher Spieler wo beschossen wurde und der Schuss daneben ging, wird in der Map<ID, ArrayList> getNoHitEnemyShips gespeichert. Die ArrayList von ID innerhalb der Map beinhaltet zum einen die Information, wie oft ein Schuss auf den jeweiligen Gegner fehlgeschlagen ist, sowie die Position auf die geschossen wurde. Die genannten Informationen nutzen wir zur Bestimmung unseres Ziels. Wenn Gegner bereits erfolgreich beschossen wurden, suchen wir zunächst den Gegner, der den größten Schaden ermlitten hat. Wenn kein Gegner erfolgreich beschossen wurde, wählen wir den Gegner, der bisher die meisten Fehlschüsse erlitten hat. Dabei erstellen wir uns eine Übersicht aller feindlichen Spieler und sortieren diese, damit wir für einen Spieler auf dessen Predecessor schließen können. Anschließend nutzen wir die Id unseres Ziels als Obergrenze und die Id seines "vermutlichen" Predecessors als Untergrenze und generieren daraus ein beliebiges Ziel auf das wir schießen wollen. Gleichzeitig wird überprüft ob, das ermittelte Ziel bereits beschossen wurde. Falls das Ziel bereits beschossen wurde, wird solange ein neues Ziel in dem genannten Intervall erzeugt, bis ein

Ziel gefunden wird, auf das noch nicht geschossen wurde.

Eigene Schiffe platzieren

Beispielstrategien zur Verteilung unserer Schiffe aus dem Spiel "Schiffe versenken" können wir nicht verwenden, da unsere Schiffe jeweils nur mit einem Feld gleichzusetzen sind und die Verteilungsstrategie in dem Spiel "Schiffe versenken" darauf bassiert den Gegner durch sich überlappende Schiffe-Teile zu verunsichern. Aus diesem Grund bleibt uns nichts anderes übrig die Schiffe innerhalb der Szene nach belieben zu verteilen. Dabei stellen wir zudem sicher, dass keine Intervalle doppelt belegt werden.

How To Start

Zum Starten eines Tests oder Spieles muss das Script "run_game.sh" ausgeführt werden. Die erforderlichen Parameter sowie drei Beispiele sind im unteren Codeschnippsel aufgeführt. Der Schnippsel zeigt die Anwendung des Scripts vom Root-Verzeichnis aus gesehen. Für das korrekte Starten wird zudem eine vorhandene Verbindung zu einem CoAP-Server benötigt. Zur Anzeige von Usage und Examples in der Konsole reicht es `./run_game.sh` ohne Parameter auszuführen. Die Datei zum Starten eines CoAP-Servers ist im Ordner 'libs' mit den Namen 'ttvs_coapdummyled.jar' zu finden.

Usage

```
./run_game.sh <test> <bootstrap-ip-address:bootstrap-port> <players-ip-address>
<players-port> <number-of-test-threads> <name-of-strategy> <coap-url:coap-port>

./run_game.sh <contest> <bootstrap-ip-address:bootstrap-port> <players-ip-address>
<players-port> <create | join> <name-of-strategy> <coap-url:coap-port>
```

Examples

(0) Start CoAP-Server:

```
  java -jar ttvs_coapdummyled.jar
```

Tests from root of this project:

(1) run 4 players (one creats a network; three join it) in one console with StrategyOne:

```
  ./run_game.sh test localhost:10000 localhost 10001 4 app.StrategyOne
localhost:5683
```

(2.0) run 1 player in one console with StrategyOne which creates a network on localhost:10000 ('players-ip-address' and 'players-port' are ignored):

```
  ./run_game.sh contest localhost:10000 localhost 0 create app.StrategyOne
localhost:5683
```

(2.1) run 1 player in one console with StrategyOne which joins a network running on localhost:10000:

```
  ./run_game.sh contest localhost:10000 localhost 10001 join app.StrategyOne
localhost:5683
```