GameRuleLogic

Hilfreiche Hilfsfunktionen

Simon Döring

Inhalt

Die Klasse GameRuleLogic hat viele Hilfsfunktionen, mit welchen man die Regeln des Spieles überprüfen kann. Hierbei sind alle Funktionen static. D.h., dass man sie ohne eine Instanz aufrufen kann.

static int	*calculateCarrots(int moveCount)* Berechnet wie viele Karotten für einen Zug der Länge moveCount benötigt werden.
static int	*calculateMoveableFields(int carrots)* Berechnet, wie weit man mit carrots Karotten gehen kann. Beispiel: Mit 68 Karotten kann man 11 Felder weit gehen.
static boolean	*canPlayCard(GameState state)** Gibt zurück, ob der derzeitige Spieler eine Karte spielen kann.
static boolean	*isValidToAdvance(GameState state, int distance)* Überprüft *Advance* Aktionen auf ihre Korrektheit. *Distance* steht für die Distanz die man zurücklegen will.Three
static boolean	*isValidToEat(GameState state)* Überprüft *EatSalad* Züge auf Korrektheit. Diese Funktion bezieht sich **nicht** auf die Karte *TAKE_OR_DROP_CARROTS*
static boolean	*isValidToExchangeCarrots(GameState state, int n)* Überprüft ob der derzeitige Spieler 10 Karotten nehmen oder abgeben kann. „n“ kann entweder 10 oder -10 sein. Je nachdem ob man annehmen oder abgeben will. Diese Funktion bezieht sich **nicht** auf die Karte *EAT_SALAD*.
static boolean	*isValidToFallBack(GameState state)* Überprüft *FallBack* Züge auf Korrektheit
static boolean	*isValidToPlayCard(GameState state, CardType c,int n)** Überprüft ob der derzeitige Spieler die Karte spielen kann. „n“ wird für die *TAKE_OR_DROP_CARROTS* Karte benötigt (s.u). „n“ kann die Wert 0, 20 oder -20 annehmen.
static boolean	*isValidToPlayEatSalad(GameState state)** Überprüft ob der derzeitige Spieler die *EAT_SALAD* Karte spielen darf.
static boolean	*isValidToPlayFallBack(GameState state)** Überprüft ob der derzeitige Spieler die *FALL_BACK* Karte spielen darf.
static boolean	*isValidToPlayHurryAhead(GameState state)* * Überprüft ob der derzeitige Spieler die *HURRY_AHEAD* Karte spielen darf.

static boolean	*isValidToPlayTakeOrDropCarrots(GameState state,int n)** Überprüft ob der derzeitige Spieler die *TAKE_OR_DROP_CARROTS* Karte spielen darf. „n“ kann entweder 0, 20 oder -20 sein.
static boolean	*isValidToSkip(GameState state)* Überprüft, ob der derzeitige Spieler aussetzen darf.
static boolean	*mustEatSalad(GameState state)* Überprüft ob man einen Salat fressen muss. Dies ist immer der Fall, wenn man in der vorherigen Runde ein Salatfeld betreten hat.
static boolean	*mustPlayCard(GameState state)* * Überprüft ob eine Karte gespielt werden muss.
static boolean	*playerMustAdvance(GameState state)* Überprüft ab der derzeitige Spieler im nächsten Zug einen Vorwärtszug machen muss.

*Diese Funktionen werden im Kapitel „Erweiterte Beispiele“ besprochen.

Die komplette API-Dokumentation ist in doc/sc/plugin2018/util/GameRuleLogic.html zu finden.

Simulation des Gegners

Natürlich kann man alle diese Funktionen auch auf den Gegenspieler anwenden. Dafür müssen wir allerdings eine Kopie des GameState erstellen:

```
try {
  GameState otherGame = gameState.clone(); // Deep-Copy
  // Rundenanzahl hochsetzten für switch Befehl:
  otherGame.setTurn(gameState.getTurn()+1);
  // Tausche currentPlayer (hängt von der Rundenanzahl ab):
  otherGame.switchCurrentPlayer();
  // Gib Informationen über den Gegner aus:
  System.out.println("CurrentPlayer von otherGame:" + otherGame.getCurrentPlayer());
} catch (CloneNotSupportedException e1) { // Fehlerbehandlung
  e1.printStackTrace();
}
```

Alle Funktionen von GameRuleLogic können wir nun, mithilfe der Variable otherGame, auf den Gegner anwenden.

Einführende Beispiele

Die meisten Funktionen dieser Klasse sind selbsterklärend. Dennoch werden einige der häufig

verwendeten Funktionen mit kleinen Beispielen vorgestellt.

```
if (GameRuleLogic.isValidToEat(gameState) != GameRuleLogic.mustEatSalad(gameState)) {
  System.out.println("Unmöglich");
}
```

Nach den Regeln muss man immer ein Salat essen, wenn man im vorherigen Zug ein Salatfeld betreten hat. Außerdem ist dies die einzige Möglichkeit die Aktion EatSalad auszuführen (nicht mit dem Spielen der EAT_SALAD Karte verwechseln).

Dadurch wird auch die Unerfüllbarkeit des folgenden Ausdrucks impliziert:

```
if (GameRuleLogic.isValidToEat(gameState) &&
    (GameRuleLogic.isValidToFallBack(gameState) || GameRuleLogic.canMove(gameState)))
{
  System.out.println("Unmöglich");
}
```

Eine weiter hilfreiche Funktion ist *isValidToAdvance*. Mit dieser Funktion wird überprüft, ob ein Vorwärtszug mit der übergeben Distanz überhaupt möglich ist:

```
// berechne die maximale Entfernung, welche man laufen darf
int max_move = GameRuleLogic.calculateMoveableFields(currentPlayer.getCarrots());

if (GameRuleLogic.isValidToAdvance(gameState, max_move +1)) {
  System.out.println("Unmöglich");
}
```

Die Funktion *calculateMoveableFields* gibt hierbei die maximale Entfernung zurück, welche man mit den übergebenen Karotten laufen darf. Diese maximale Entfernung wird immer um 1 erhöht, was dazu führt, dass der Zug immer unmöglich ist.

Erweitertes Beispiel

Alle Funktionen die mit einem * markiert wurden (s.o) haben eine Gemeinsamkeit. Sie beziehen sich auf das Spielen von Karten. Das Spielen von Karten ist allerdings nur erlaubt, wenn man das entsprechende Hasenfeld in der selben Zug betreten hat. Deshalb müssen wir GameState bearbeiten, damit diese Funktionen überhaupt Sinn haben. Das folgende Beispiel gibt eine Möglichkeit an, wie man diese Funktionen einsetzten kann:

```
int nextHareFieldIndex = gameState.getNextFieldByType(FieldType.HARE,
                                                      currentPlayer.getFieldIndex())
// wenn es ein nächstes Hasenfeld gibt
if (nextHareFieldIndex > 0) {
  try {

    GameState gameHare = gameState.clone(); // erstelle Deep-Copy
    Player harePlayer = gameHare.getCurrentPlayer(); // erstelle Shallow-Copy

    //setzte den aktuellen Spieler auf ein Hasenfeld
    harePlayer.setFieldIndex(nextHareFieldIndex);
    System.out.println(gameHare.getCurrentPlayer().getCards()); // gib alle Karten aus

    // Welche Karten kann man spielen?
    System.out.println("Play EatSalad: " +
                       GameRuleLogic.isValidToPlayEatSalad(gameHare));
    System.out.println("Play TakeOrDropCarrots: " +
                       GameRuleLogic.isValidToPlayTakeOrDropCarrots(gameHare,20));
    System.out.println("Play HurryAhead: " +
                       GameRuleLogic.isValidToPlayHurryAhead(gameHare));
    System.out.println("Play FallBack: " +
                       GameRuleLogic.isValidToPlayFallBack(gameHare));

  } catch (CloneNotSupportedException e1) {
    e1.printStackTrace();
  }
}
```

Hierfür müssen wir den Spieler einfach nur auf das Hasenfeld setzten. Allerdings wird nicht überprüft, ob der Spieler überhaupt bis zum nächsten Hasenfeld laufen kann.

Aufgaben

1. Ist das Ausführen der inneren If-Bedingung wirklich unmöglich. Erkläre warum oder gib ein Gegenbeispiel an:

```
int dif = gameState.getOtherPlayer().getFieldIndex() - currentPlayer.getFieldIndex();

if (dif >= 0 &&
    dif <= GameRuleLogic.calculateMoveableFields(currentPlayer.getCarrots())) {
  if (GameRuleLogic.isValidToAdvance(gameState, dif)) {
    System.out.println("Unmöglich");
  }
}
```

2. Erweiterte das Beispiel aus dem Kapitel „Erweitertes Beispiel“ so, dass sicher gestellt wird, dass der aktuelle Spieler auf ein Hasenfeld gesetzt wird, welches er wirklich erreichen kann.