Runde 1 Aufgabe 4: Würfelglück

Bearbeiter/-in dieser Aufgabe:
Joshua Benning

7. September 2021

Inhaltsverzeichnis

Aufgabenstellung

Barbara findet auf dem Dachboden eine Kiste mit vielen verschiedenen Spielwürfeln. Ein "normaler" ist dabei, der sechs Seiten hat mit den Zahlen 1 bis 6. Und dann gibt es noch welche mit ganz anderen Formen, zum Beispiel mit 4, 10, 12 oder gar 20 Seiten.

Zum großen Einsatz kommt die Kiste beim nächsten gemeinsamen Spielabend mit ihren Freunden. Jeder darf sich einen Würfel aussuchen und es wird "Mensch äregere Dich nicht" gespielt. Es kommt schnell eine Diskussion auf, welcher Würfel denn die besten Chancen einräumt. Anna ist begeistert vom 20-seitigen Würfel, weil sie in einem Zug bis zu 20 Schritte machen kann. Clemens hingegen findet es viel wichtigerm dass er oft eine 6 würfelt. Denn diese benötigt man weiterhin, um eine Figut auf das Startfeld zu stellen.

Hilf den Freunden und schreibe ein Programm, das eine Liste von Würfeln einliest und bestimmt, wie gut diese Würfel geeignet sindm um das Spiel zu gewinnen. Verleiche dazu jeden Würfel mit jedem anderen. Um festzustellen, welcher von zwei gegebenen Würfeln besser ist, simuliere ausreichend viele Spiele zweier Personen, wobei beide je einen der Würfel verwenden. Natürlich sollen in der Hälfte der Simulationen die erste Person anfangen und in der anderen die andere. Eine genaue Beschreibung der für die Simulation gültigen Spielregeln findest du auf den BWINF-Websiten.

Lösungsidee

Zum Vergleichen der Würfel simuliere ich eine Partie Mensch ärgere dich nicht zwischen den einzelnen Würfeln. Zum Vergleich führe ich ein Turnier nach dem „Jeder gegen Jeden“-Prinzip aus und zähle die absoluten Siege. Um hierbei den Zufall einzuschränken wird eine hohe Anzahl an Spielen gespielt.

Umsetzung

Die Implementation erfolgt in Java. Ein Spielobjekt beinhaltet die spielenden Teams und verwaltet welches Team zieht. Die essenzielle Teile ergeben sich einerseits zu der Zugmethode, sprich der Methode welchen den tatsächlichen Zug durchführt, und des Spielobjekts welche die Zugreihenfolge verwaltet

Die Zugmethode

Basierend tut die Zugmethode auf der Regel:

„Wer mehrere Spielsteine auf der Laufbahn stehen hat, muss mit dem vordersten Stein ziehen, der gezogen werden kann.“

Jedes Team besitzt 4 Spieler. Basierend auf dieser Regel wird die Liste der Spieler sortiert, beginnend mit dem am weitesten vorne stehenden Spieler. Für diesen Spieler wird dann gewürfelt mit dem entsprechenden Würfel und es wird in mehrere Fälle unterschieden.

Fall 1: Sollte keine Bewegung möglich sein, wird dreimal gewürfelt und falls eine 6 gewürfelt wurde, entsprechend gezogen.

Fall 2: Es wurde eine 6 gewürfelt und es befinden sich Spieler auf den B-Feldern. Hier wird die 6 entweder auf einen Spieler auf dem A-Feld verwendet (Räumungspflicht), oder auf den am weitesten vorne stehenden Spieler verwendet

Fall 3: Es wurde eine 6 gewürfelt → Spieler bewegen, erneut ziehen

Fall 4: Sollte sich ein Spieler bewegen lassen und die gewürfelte Zahl keine 6 sein, wird der am weitesten vorne stehende Spieler um die gewürfelte Zahl bewegen.

Ablauf der Züge

Das Spielobjekt ruft rekursiv die Zugmethoden der Teams prinzipiell abwechselnd auf. Die Zugmethode eines Teams gibt dabei ein MoveResult zurück.

Dabei sind mehrere Rückgabewerte möglich:

- DEFAULT
- STEP_AGAIN
- TEAM_RED_WON
- TEAM_YELLOW_WON

Von dem abwechselnden Aufrufen kann allerdings je nach Rückgabe abgewichen werden. Sollte die Zugmethode „DEFAULT“ zurückgeben, wird danach die Zugmethode des anderen Teams aufgerufen. Wenn die Zugmethode „STEP_AGAIN“ zurückgibt, wird im nächsten Zug nicht das Team welches am Zug ist gewechselt, sondern das selbe Team zieht erneut. Sollte „TEAM_RED_WON“ oder „TEAM_YELLOW_WON“ zurückgegeben werden, bricht die Rekursion ab und der Gewinner des Spiels steht fest.

Da das Spiel sich besonders bei den in den BWInf-Beispielen verwendeten Würfeln gerne in eine Situation begibt, in der keines der beiden Teams mehr ziehen kann, sind zusätzlich die Züge pro Spiel auf 5000 limitiert. Es werden alle Züge gezählt und bei überschreiten dieser Konstanten die Rekursion abgebrochen. In diesem Falle gibt es keinen Gewinner.

Vergleich der Würfel

Die Würfel lassen sich in mehreren Aspekten unterscheiden. Damit ein Würfel im Spiel gut abschneidet muss er unter anderem eine ausgeglichene Anzahl an großen und kleinen Zahlen bieten, um damit sowohl kleine als auch große Distanzen zu bieten. Außerdem spielt die Anzahl an Sechsen auf dem Würfel eine entscheidende Rolle, da die Sechs sowohl einen weiteren Zug als auch das auf das Feld bringen von Spielern ermöglicht. Es gäbe durchaus Möglichkeiten diese Würfel auf einzelne Aspekte zu untersuchen, allerdings lässt sich davon schwer auf den tatsächlichen Spielerfolg schwer zurückschließen. Die Würfel werden daher nach der absoluten Anzahl der Siege in einem „Jeder gegen Jeden“ System verglichen. Um dabei den Zufall möglichst auszuschließen, werden pro Aufeinandertreffen von zwei Würfeln 10.000 Spiele durchgeführt. Es werden demnach für 6 Würfel (6 * 5 * 10.000) 300.000 Spiele gespielt.

Beispiele

Bei den Beispielen werden die Würfel über ihre Ziffern dargestellt. Die Listen sind nach der Anzahl an Siegen absteigend sortiert.

Beispiel 0

86816.0 Siege mit Würfel: [1, 1, 1, 6, 6, 6]
64868.0 Siege mit Würfel: [1, 2, 3, 4, 5, 6]
59221.0 Siege mit Würfel: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
55814.0 Siege mit Würfel: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
33281.0 Siege mit Würfel: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
0.0 Siege mit Würfel: [1, 2, 3, 4]
Benötigte Zeit: 8.698s

Beispiel 1

58312.0 Siege mit Würfel: [1, 2, 3, 4, 5, 6]
54577.0 Siege mit Würfel: [2, 3, 4, 5, 6, 7]
45798.0 Siege mit Würfel: [3, 4, 5, 6, 7, 8]
41203.0 Siege mit Würfel: [4, 5, 6, 7, 8, 9]
25487.0 Siege mit Würfel: [5, 6, 7, 8, 9, 10]
14501.0 Siege mit Würfel: [6, 7, 8, 9, 10, 11]
Benötigte Zeit: 40.823s

Beispiel 2

71400.0 Siege mit Würfel: [1, 6, 6, 6, 6, 6]
61169.0 Siege mit Würfel: [1, 1, 6, 6, 6, 6]
44690.0 Siege mit Würfel: [1, 1, 1, 6, 6, 6]
22323.0 Siege mit Würfel: [1, 1, 1, 1, 6, 6]
418.0 Siege mit Würfel: [1, 1, 1, 1, 1, 6]
Benötigte Zeit: 9.205s

Beispiel 3

71641.0 Siege mit Würfel: [1, 2, 5, 6]
63121.0Siege mit Würfel: [1, 2, 3, 4, 5, 6, 7, 8]
55916.0 Siege mit Würfel: [1, 2, 3, 4, 5, 6]
48554.0 Siege mit Würfel: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
44311.0 Siege mit Würfel: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
16457.0 Siege mit Würfel: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
Benötigte Zeit: 8.215s

Quellcode

```
/**
 * Methode welche über den Zug entscheidet und den Spieler bewegt
 */
public MoveResult move() {
    if (hasWon()) {
        if (this.name.equals("Team Gelb")) {
            return MoveResult.TEAM_YELLOW_WON;
        } else {
            return MoveResult.TEAM_RED_WON;
        }
    }
    this.players.sort((o1, o2) -> o1.position > o2.position ? 1 : 0);
    int playerIndex = 0;
    int distanceToMove = this.dice.roll();
    while (playerIndex < 4) { // Try to move every player, starting with the most far away one
        Player playerToMove = this.players.get(playerIndex); // Get the player
        if (playerToMove.isInBase() && !hasSpaceToMove()) { // Check if the most far away player is in base, if yes there is none on the field
            for (int dice = 0; dice < 2; dice++) { // Roll the dice 3 times
                distanceToMove = this.dice.roll(); // Get random number
                if (distanceToMove == 6) { // If the random number was a 6, move player out of base
                    playerToMove.setInBase(false);
                    playerToMove.setPosition(1);
                    return MoveResult.STEP_AGAIN;
                }
            }
            break;
        } else {
            if (distanceToMove == 6) { // If the move was successful and the diced number was a 6, dice again...
                this.players.sort((o1, o2) -> o1.position > o2.position ? 1 : 0); // Resort list
                if (!hasPlayersInBase()) {
                    int playerIndex2 = 0;
                    boolean moveSucceededSix;
                    while (playerIndex2 < 4) {
                        playerToMove = this.players.get(playerIndex2);
                        moveSucceededSix = movePlayer(playerToMove, distanceToMove);
                        if (moveSucceededSix) {
                            return MoveResult.STEP_AGAIN;
                        } else {
                            playerIndex2++;
                        }
                    }
                } else {
                    Player potentialAFieldPlayer = getPlayerOnA();
                    if (potentialAFieldPlayer != null) {
                        movePlayer(potentialAFieldPlayer, distanceToMove);
                    } else {
                        playerToMove = this.players.stream().filter(Player::isInBase).findFirst().get();
```

```
                    playerToMove.setPosition(1);
                    playerToMove.setInBase(false);
                }
            }
            return MoveResult.STEP_AGAIN;
        } else {
            // Normal move
            boolean moveSucceeded = movePlayer(playerToMove, distanceToMove); // Try to move player
            if (!moveSucceeded) playerIndex++; // Go for next player
            else {
                return MoveResult.DEFAULT;
            }
        }
    }
  }
  if (hasWon()) {
    if (this.name.equals("Team Gelb")) {
      return MoveResult.TEAM_YELLOW_WON;
    } else {
      return MoveResult.TEAM_RED_WON;
    }
  }
  return MoveResult.DEFAULT;
}



public class Spiel {
  public static int MAX_STEPS_PER_GAME = 5000;

  public ArrayList<Team> teams = new ArrayList<>(2);
  public Team currentlyMovingTeam = null;
  private Team winner;

  private int draws;

  private final Team teamRot;
  private final Team teamGelb;
  Iterator<Team> teamIterator;

  /**
   * Simulation eines Spieles
   * @param teamRot Objekt des Team Rot
   * @param teamGelb Objekt des Team Gelb
   */
  public Spiel(Team teamRot, Team teamGelb) {

    this.teamRot = teamRot;
    this.teamGelb = teamGelb;

    teamRot.resetPlayerPositions();
```

```
        teamGelb.resetPlayerPositions();

        teamGelb.setOpponent(teamRot);
        teamRot.setOpponent(teamGelb);

        teams.add(teamRot);
        teams.add(teamGelb);

        teamIterator = teams.iterator();
    }
```

```
/**
 * Methode die über den TeamIterator das Team wechselt
 * und dann die Zugmethode des entsprechenden Teams aufruft
 */
public void step() {
    if (draws >= MAX_STEPS_PER_GAME) {
        return;
    }
    draws++;
    if (!teamIterator.hasNext()) teamIterator = teams.iterator();

    currentlyMovingTeam = teamIterator.next();
    MoveResult result = currentlyMovingTeam.move();
    switch (result) {
        case STEP_AGAIN -> this.stepAgain();
        case DEFAULT -> this.step();
        case TEAM_RED_WON -> winner = this.teamRot;
        case TEAM_YELLOW_WON -> winner = this.teamGelb;
    }
}

/**
 * Methode die nicht das Team wechselt,
 * sondern das selbe Team erneut ziehen lässt
 */
public void stepAgain() {
    if (draws >= MAX_STEPS_PER_GAME) {
        return;
    }
    MoveResult result = currentlyMovingTeam.move();
    switch (result) {
        case STEP_AGAIN -> this.stepAgain();
        case DEFAULT -> this.step();
        case TEAM_RED_WON -> winner = this.teamRot;
        case TEAM_YELLOW_WON -> winner = this.teamGelb;
    }
}

public Team getWinner() {
    return winner;
}
}
```

Quellen / Material

Aufgabestellung: https://bwinf.de/bundeswettbewerb/40/1/ [03.09.2022 ~ 20:15 Uhr]

Beispieldaten: https://bwinf.de/bundeswettbewerb/40/1/ [03.09.2022 ~ 20:15 Uhr]