Bedienungsanleitung

9 Holes

Gabriel Schafflüztel

Inhalt

[2 Spielregeln 3](#_Toc530561919)

[3 Controller 4](#_Toc530561920)

[4 Klassendiagramm 5](#_Toc530561921)

[5 Beispiel eines Spiels 6](#_Toc530561922)

[6 Tests 9](#_Toc530561923)

# Spielregeln

Die Spielregeln des Spiels 9 holes funktionieren folgendermassen:

Material

Es findet sich ein Brett vor, welches 9 Löcher besitzt. Es benötigt zwei Spieler, wobei jeder 3 Spielsteine besitzt.

Platzierungsphase

Je ein Spieler platziert rundenbasiert eines seiner Spielsteine in ein Loch, bis keine Spielsteine übrig sind.

Bewegungsphase

Je ein Spieler bewegt rundenbasiert seine Steine so, dass schlussendlich 3 Steine des Spielers nebeneinanderliegen. Die Spieler dürfen nicht auf Felder, die von gegnerischen Steinen besetzt sind und gewinnen kann man nur dann, wenn die eigenen Steine vertikal oder horizontal 3 Löcher besetzen.

# Controller

Brett

Das Brettobjekt, auf welchem das eigentliche Spiel realisiert wird, wird im Controller mit dem Befehl «Board board = **new** Board();» erstellt.

Spielmodus

In der Klasse Controller wird der Spielmodus dadurch bestimmt, dass der player1 und der player2 entweder als AI oder Human instanziiert werden. Diese können problemlos den Wünschen entsprechend bestimmt werden.

brett ausgeben

Das Brett auszugeben ist ermöglicht, wie im Controller ersichtlich, mit der entsprechende Methode des Brettes mit dem Befehl «board.display();»

Setz-Phase

Um die drei Spielsteine je vom Spieler setzen zu lassen, befindet sich im Controller eine Schlaufe, die solange läuft, bis das Brett nicht mehr den Enum Wert DROP zurückgibt. Innerhalb dieser Schlaufe wird ein Spielerobjekt aufgefordert, den nächsten Spielstein zu setzen. Das Brett wird anschliessend ausgegeben und der nächste Spieler kommt an die Reihe, indem er der Variable currentPlayer zugewiesen wird.

Bewegungs-Phase

Um die Spielsteine zu bewegen, ist wie bei der Setz-Phase eine **while** Schlaufe vorhanden, die den Spielstatus abruft, ob dieser noch den Status ***MOVE*** zurückgibt. Anschliessend wird die Methode nextMove() des den momentanen Spieler currentPlayer gerufen, das Brett ausgegeben und schlussendlich wird der momentane Spieler zu Gegenspieler gewechselt.

# Klassendiagramm

Da nicht alle Klassen im Handbuch im Detail erwähnt sind, werde ich diese im nachfolgendem Klassendiagramm aufzeigen.

Move

y:int

x:int

direction:char

Move(int,int,char)

1…\*

0…\*

2

2

1…\*

1…\*

Board

PIECE\_X:char

PIECE\_Y:char

EMPTY:char

field:char[][]

amountPieces:int

lastPiece:char

line:String

roundsCount:int

Board()

Board(char[][],int,char)

getCopy():Board

getState():State

isRowComplete(int):boolean

isColComplete(int):boolean

isARowComplete():boolean

isAColComplete():boolean

setPiece(int,int,char):boolean

moveRight(int,int,char):boolean

moveLeft(int,int,char):boolean

moveUp(int,int,char):boolean

moveDown(int,int,char):boolean

isFieldEmpty(int,int):boolean

isDirectionEmpty(int,int,char):boolean

getAvailableFields():List<int[]>

getAvailableMoves(char):List<Move>

display():void

RandomPlayer

DROP:State

RandomPlayer(Board,char)

randomNumberInRange(int,int):int

nextDrop():void 🡩Player

nextMove():void 🡩Player

Controller

main(String[]):void

State

DROP:State

MOVE:State

X\_WON:State

O\_WON:State

DRAW:State

Move(int,int,char)

AI

AI(Board,char)

nextDrop():void 🡩Player

dropRatings(List<int[]>, int[], char, int):void

moveRatings(List<Move>,Board):int[]

rateMove(List<Move>,int[],char,int):void

bestMove(int[],List<Move>):Move

bestPos(int[],List<int[]>):int[]

indexOfMax(int[]):int[]

nextMove():void 🡩Player

Player

opponent:char

piece:char

board:Board

Player(Board,char)

nextDrop():void

nextMove():void

Human

errorMsg:String

yPosMsg:String

xPosMsg:String

directionMsg:String

direction:String

x:int

y:int

Human(Board,char)

nextDrop():void 🡩Player

nextMove():void 🡩Player

# Beispiel eines Spiels

Nachfolgend befindet sich ein Durchlauf der Spielversion v1 am 19.11.2018. Dieser dient dazu, eine Ahnung der Funktionalität bzw. das ungefähr zu erwartende Resultat dieser Spielversion zu erhalten.

\_\_\_ \_ \_ \_\_\_\_ \_ \_\_\_\_\_\_ \_\_\_\_\_

/ \_ \ | | | | / \_\_ \ | | | \_\_\_\_| / \_\_\_\_|

| (\_) | | |\_\_| | | | | | | | | |\_\_ | (\_\_\_

\\_\_, | | \_\_ | | | | | | | | \_\_| \\_\_\_ \

/ / | | | | | |\_\_| | | |\_\_\_\_ | |\_\_\_\_ \_\_\_\_) |

/\_/ |\_| |\_| \\_\_\_\_/ |\_\_\_\_\_\_| |\_\_\_\_\_\_| |\_\_\_\_\_/

Version 1.0

Topomedics

-------------------------------------------------------

0 1 2 < x

0( )-( )-( )

1( )-( )-( )

2( )-( )-( )

^

y

-------------------------------------------------------

>> AI is picking a position.

-------------------------------------------------------

0 1 2 < x

0(X)-( )-( )

1( )-( )-( )

2( )-( )-( )

^

y

-------------------------------------------------------

>> Player O y-Pos: 1

>> Player O x-Pos: 1

-------------------------------------------------------

0 1 2 < x

0(X)-( )-( )

1( )-(O)-( )

2( )-( )-( )

^

y

-------------------------------------------------------

>> AI is picking a position.

-------------------------------------------------------

0 1 2 < x

0(X)-( )-( )

1( )-(O)-( )

2(X)-( )-( )

^

y

-------------------------------------------------------

>> Player O y-Pos: 0

>> Player O x-Pos: 0

------------------------INVALID------------------------

>> Player O y-Pos: 3

>> Player O x-Pos: 5

------------------------INVALID------------------------

>> Player O y-Pos: 1

>> Player O x-Pos: 0

-------------------------------------------------------

0 1 2 < x

0(X)-( )-( )

1(O)-(O)-( )

2(X)-( )-( )

^

y

-------------------------------------------------------

>> AI is picking a position.

-------------------------------------------------------

0 1 2 < x

0(X)-( )-( )

1(O)-(O)-(X)

2(X)-( )-( )

^

y

-------------------------------------------------------

>> Player O y-Pos: 2

>> Player O x-Pos: 1

-------------------------------------------------------

0 1 2 < x

0(X)-( )-( )

1(O)-(O)-(X)

2(X)-(O)-( )

^

y

-------------------------------------------------------

>> AI is picking a Move.

-------------------------------------------------------

0 1 2 < x

0( )-(X)-( )

1(O)-(O)-(X)

2(X)-(O)-( )

^

y

-------------------------------------------------------

>> Player O y-Pos: 1

>> Player O x-Pos: 0

Pick direction(u,d,r,l): u

-------------------------------------------------------

0 1 2 < x

0(O)-(X)-( )

1( )-(O)-(X)

2(X)-(O)-( )

^

y

-------------------------------------------------------

>> AI is picking a Move.

-------------------------------------------------------

0 1 2 < x

0(O)-( )-(X)

1( )-(O)-(X)

2(X)-(O)-( )

^

y

-------------------------------------------------------

>> Player O y-Pos: 0

>> Player O x-Pos: 0

Pick direction(u,d,r,l): r

-------------------------------------------------------

0 1 2 < x

0( )-(O)-(X)

1( )-(O)-(X)

2(X)-(O)-( )

^

y

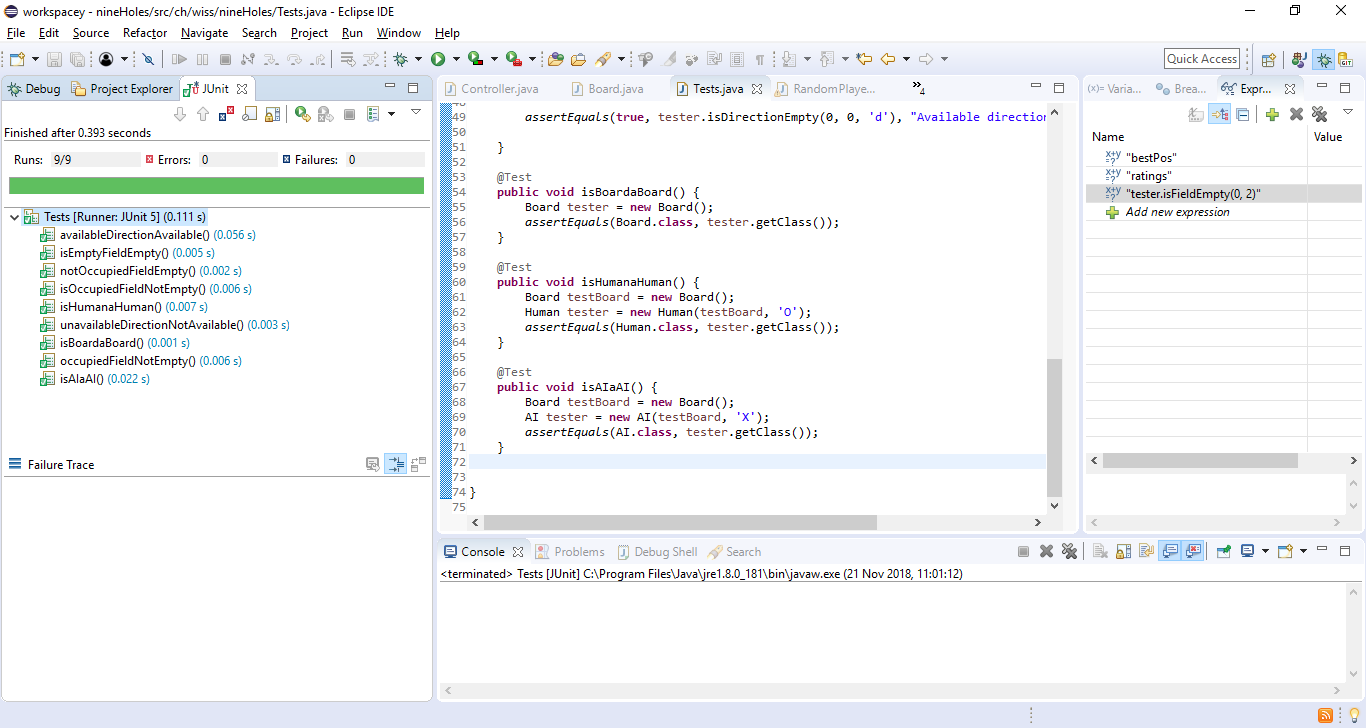
-------------------------------------------------------

Player O won.

# Tests

Ich prüfte das Programm mit JUnit Tests. Hierbei überprüfte ich nachfolgende Kriterien.

* Ist ein besetztes Feld nicht leer?
  + Rückgabewert: Positiv
* Ist ein leeres Feld leer?
  + Rückgabewert: Positiv
* Ist eine unerreichbare Richtung unerreichbar?
  + Rückgabewert: Positiv
* Ist eine erreichbare Richtung erreichbar?
  + Rückgabewert: Positiv
* Ist ein Brett ein Brett?
  + Rückgabewert: Positiv
* Ist ein menschlicher Spieler ein menschlicher Spieler?
  + Rückgabewert: Positiv
* Ist ein KI ein KI?
  + Rückgabewert: Positiv



1: JUnit Testresultate Eclipse