# Protokoll

Datum 30.10.2018

Thema interne Representation Spielfeld,

--[**https://github.com/erikgaal/qwirkle/blob/master/src/main/java/nl/utwente/ewi/qwirkle/model/Board.java**](https://github.com/erikgaal/qwirkle/blob/master/src/main/java/nl/utwente/ewi/qwirkle/model/Board.java)

**Dieses Dokument ist auch als Protokoll für den 06.11.18 Online Servertreffen zu sehen.**

Bogna, Lukas G, Tim

* Algorithmus Spielzug validieren und Punkte validieren

Linus, (Steffen)

* Interface, Konfiguration

Lukas B

* Klassen implementieren

**Spielfeld**

- Wir haben ein Spielfeld als MultiKeyMap ähnlich wie Board(Tile.x,Tile.y)

https://github.com/google/guava/wiki/NewCollectionTypesExplained#table

**Map Spieler auf Feld:**

https://github.com/google/guava/wiki/NewCollectionTypesExplained#bimap

**Spielzug validieren**

- wir müssen aufgrund der obigen Struktur die gesamte MultiKeyMap durchlaufen

o  Meiner Meinung nach aber nicht schlimm, wenn in der MultiKeyMap sowieso nur die Steine drin legen. Selbst wenn wir 10.000 Steine in der MultiKeyMap liegen haben, welches 10.000 valide Spielzüge entsprechen, dann würde der Computer dennoch noch performant arbeiten können.

- den Algorithmus in mehreren Phasen aufbauen:

1. Für alle zu legenden Steine zu prüfen: Gibt es Steine um den gelegten Stein?

1. dazu dann über KeySet entsprechend prüfen, ob in der MultiKeyMap die Koordinatenobjekte(Integer) vorhanden sind, falls nein, dann ablehnen
2. sonst zu 2.

2.  Zu prüfen, ob alle Steine in einer Reihe gelegt wurden

1. Entweder alle Steine haben dieselbe X-Koordinate oder alle Steine haben dieselbe Y-Koordinate

3.  Zu prüfen, ob alle Steine anhand der bekannten Spielregeln gelegt werden können

1.  Sind die Steine, die in die Reihe gelegt wurden alle:

·         1. Von derselben Farbe

·         2. Von derselben Form

2.  mit der kopierten MultiKeyMap entsprechend die Koordinaten abfragen und die Steine in einem Array/Liste speichern, das Array/Liste entsprechen durchlaufen und prüfen, ob die Reihe von derselben Farbe oder Form sind

**Punkte validieren**

1. es liegt ein Array vor, in der alle Spielsteine der Reihe aufgelistet sind
2. es liegt ein separates Array vor, in der alle gelegten Spielsteine aufgelistet sind

Punkteberechnung 1:

* die Tiles, die die Reihe vervollständigen

Punkteberechnung 2:

* die Reihen, die sich ergeben, wenn durch die gelegten Tiles eine zusätzliche Reihe erzeugt wird

**Entwurf (Teilvalidierung fehlend):**

public int validatePoints(Tile[] layed) {

int calPoints = 0;

List<Tile[]> achievedRows = new List<Tile[]>();

for(Tile tile : layed) {

achievedRows.add(tile.getRowHorizontal());

achievedRows.add(tile.getRowVertical());

}

achievedRows = filterDuplicateRows(achievedRows); //Validierung hier da Felditeration

if (achievedRows.isEmpty())

return Integer.MIN\_VALUE;

for(Tile[] row : achievedRows) {

calPoints += (row.length == config.countTiles) ? config.countTiles \* 2 : row.length;

}

return calPoints;

}

**Spielende**

-    siehe Interface-Dokument

**Klasse Map, notwendig zu implementierende Funktionen:**

private validateMove(List<Tile> tilelist);

* return boolean

public getScore(Player player);

* On-Demand Berechnung Score
* return int

public doMove();

* legt nach Validierung die zu legenden Steine auf das Spielfeld
* return bool / Fehlermeldung? zu klären

public getCoordinate(Tile tile);

* return Coordinate;

( remove(); )

public Tile getTile(int x, int y);

public Tile getTile(Tile tile); <--evtl.

private setTile(Tile tile);

**Klasse Tile**

private int identification; # int oder string je nach ID

private Color color;

private Shape shape;

**Klasse Coordinate**

private int x;

private int y;

**Klasse Shape**

enum shape{

};

getShape();  
- return shape

compareWith(Tile tile);

* return shapeType == Tile.getShape()

**Klasse Color**

enum color {

}

compareWith(Tile tile);

-return boolean

**Klasse abstract Participant**

private int identification;

private String name;

**Klasse Player extends Participant**

private int playerId;

private int points;

**Klasse Observer extends Participant**

**Klasse Game**

private int identification;

private Map map;

private List<Player> scorelist;

private List<Observer> joinedPlayers;

private Date StartTime;

private GameState state;

Enum GameState{

NOT\_STARTED, PAUSED, STARTED, FINISHED

}

**Klasse Configuration**

<https://github.com/google/gson>

* UML-Klassendiagramm von Configuration s. Interface Document
* Speicherung als .json Datei auf Festplatte

**Klasse Competition**

private List<Game> games

private int identification

private List<Player> players

private GameState state;

public void startNext();