

VU Software Engineering 1

Abgabedokument

Teilaufgabe 2

Nachname, Vorname : Milinkovic Aleksandar

Matrikelnummer : 01646014

E-Mail Adresse : milinkovicaleks@gmail.com

Datum : 05.12.2017

Algemeine Anforderungen an die implementierung :

!!!! Wichtig !!!!! : Falls die einzelnen Screenshots ein bisschen abweichen von den Code ist es deswegen, weil ich noch Kommentare eingefügt habe und den Code ein bisschen in Ordnung gebracht habe, es gibt ein Video am ende das zeigt wie das Spiel funktioniert(Ohne Spielregeln) mit Server und Client, Ich habe bemerkt das Windows meine Socket Connection schließt und das Programm geht, dann nicht aber auf Linux hat es normalerweise geklappt, also es kann passieren das

sich der zweite Klient nicht verbinden kann auf einer Windows Maschine. Falls Sie keine Code-Kommentare für manche Sachen finden dann bitte in der Doku nachschauen. Die erreichte Test-Coverage war 52% aber es kann sein da ich Kleinigkeiten verändert habe das es um +/-5% abweicht. Der AStar Algorithmus ist zu 80% von Stack-Overflow mit gewissen Anpassungen. Die Business-Rules können einfach an oder aus gemacht werden in dem man sie in der Methode checkMove auskommentiert.

1. Source-Code-Verwaltung : ich habe ein git archive file erstellt und auch ein bundle alle befinden sich in der zip Datei bei SWE1_SERVER/src/main/java

2. Logging : Für logging benutze Ich einfach die Java.util.logging.* weil für unser Projekt wirklich nicht mehr nötig ist und auch einen fileHandler um dann eine File zu erstellen wo die ganzen logging Einträge stehen.

Beispiel :

Nov 26, 2017 6:02:55 PM Server runServer

INFO: Server is Ready and waiting for connection ...

Nov 26, 2017 6:03:02 PM Server runServer

INFO: Server is accepting connection ...

Nov 26, 2017 6:03:02 PM Server runServer

INFO: Thread started

Nov 26, 2017 6:03:02 PM Server runServer

INFO: Server is Ready and waiting for connection ...

Nov 26, 2017 6:03:04 PM ServerThread run

INFO: Checking Map ...

Nov 26, 2017 6:03:05 PM ServerThread run

INFO: Message Received From Client And Deligated To Server

Nov 26, 2017 6:03:05 PM Server updateBoard

INFO: Board Is Being Updated

Nov 26, 2017 6:03:05 PM Server updateBoard

INFO: PlayerOneOnline

Nov 26, 2017 6:03:05 PM ServerThread run

INFO: Player Tried To Play But Not His Turn : 20

Nov 26, 2017 6:03:07 PM Server runServer

INFO: Server is accepting connection ...

Nov 26, 2017 6:03:07 PM Server runServer

INFO: Thread started

Nov 26, 2017 6:03:07 PM Server runServer

INFO: Server is Ready and waiting for connection ...

Nov 26, 2017 6:03:08 PM ServerThread run

INFO: Checking Map ...

Nov 26, 2017 6:03:08 PM ServerThread run

INFO: Message Received From Client And Deligated To Server

Nov 26, 2017 6:03:08 PM Server updateBoard

INFO: Board Is Being Updated

Nov 26, 2017 6:03:08 PM Server updateBoard

INFO: PlayerTwoOnline

Nov 26, 2017 6:03:08 PM Server updateBoard

INFO: Server sent new Map to Players

Nov 26, 2017 6:03:09 PM ServerThread run

INFO: Player Tried To Play But Not His Turn : 36

Nov 26, 2017 6:03:10 PM ServerThread run

INFO: Message Received From Client And Deligated To Server

Nov 26, 2017 6:03:10 PM Server updateBoard

INFO: Board Is Being Updated
Nov 26, 2017 6:03:10 PM Server updateBoard
INFO: Server sent new Map to Players
Nov 26, 2017 6:03:10 PM ServerThread run
INFO: Player Request To Move And Approved : 20
Nov 26, 2017 6:03:10 PM Server updateBoard
INFO: Board Is Being Updated
Nov 26, 2017 6:03:10 PM Server updateBoard
INFO: Server sent new Map to Players
Nov 26, 2017 6:03:10 PM ServerThread run
INFO: ServerThread Sent New Player Position
Nov 26, 2017 6:03:10 PM ServerThread run
INFO: Message Received From Client And Deligated To Server
Nov 26, 2017 6:03:10 PM Server updateBoard
INFO: Board Is Being Updated
Nov 26, 2017 6:03:10 PM Server updateBoard
INFO: Server sent new Map to Players
Nov 26, 2017 6:03:10 PM ServerThread run
INFO: Player Request To Move And Approved : 36
Nov 26, 2017 6:03:10 PM Server updateBoard
INFO: Board Is Being Updated

Ich habe auch ein file inkludiert "LogFile" wo ein Beispiel eines Spiels ist, in Windows bitte mit Notepad++ öffnen in Ubuntu geht es auch mit gedit.

Besispiele für die Loggingeinträge sind :

1. `logger.info("Server is Ready and waiting for connection ...")` befindet sich in der `runServer` Methode in der Klasse `Server`, dieser loggingeintrag ist schon ziemlich wichtig sagt den klienten das da jemand auf eine connection wartet.

2. `logger.info("Server is accepting connection ...")` befindet sich auch in der selben Methode und sagt das eine connection akzeptiert wird was dann auch den klienten bewusst macht im welchen status er sich befindet.

3. `logger.info("player requested to play but could not")` befindet sich auch in der selben Klasse in der Methode `isPlayerTurnAndChangePlayer`, diese info ist sehr wichtig weil sie andeutet das ein klient versucht hat zu spielen obwohl er nicht an der Reihe war was auch sehr wichtig für Fehlerbehandlung.

4. `server.logger.info("Checking Map ... ")` befindet sich in der Klasse `ServerThread` in der Methode `run` und dient dazu anzuzeigen das die Map erhalten wurde und überprüft wird. Da sich der logger im server befindet rufe ich die Methode über `server.` auf.

5. `server.logger.info("!!!!!! MAP FAILED !!!!!!!")` befindet sich in der Klasse `ServerThread` auch in der Methode `run` und zeigt das die Map nicht die Anforderungen erfüllt.

6. `server.logger.info("Message Received From Client And Deligated to Server")` befindet sich auch in der Klasse `ServerThread` in der Methode `run` und dient dazu anzudeuten das die Nachricht erhalten wurde und an den Server weitergeleitet wurde.

7. server.logger.info("Player Requested to Move And Approved : " + message.PlayerID) befindet sich in der Klasse ServerThread in der Methode run und zeigt ob der Spieler der die Message geschickt hat auch wirklich an der reihe ist und spielen darf zusammen auch mit anderen logging infos in der selben Methode um jetzt nicht alle explizit aufzulisten.

Manche nachrichten habe ich nicht geloggt wie zB. aus welchen Grund ein Spieler verloren hat. Das könnte man machen aber ich wollte nicht das logging file damit überfluten weil es nicht so wichtig für den Ablauf des Spiels ist.

Ein Paar Screenshots :

```
public void setLog() {
    if(!logSet) {
        try {
            fh = new FileHandler("LogFile.log");
            logger.addHandler(fh);
            SimpleFormatter formatter = new SimpleFormatter();
            fh.setFormatter(formatter);
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
    logSet = true;
}

//This is the core method where the Server starts
//The server waits for a connection with accept()
//When a connection is there it creates a thread and saves them

public void runServer() throws IOException{
    ServerSocket serverSocket = new ServerSocket(PORT);
    while(true) {
        try {
            setLog();
            logger.info("Server is Ready and waiting for connection ...");
            Socket socket = serverSocket.accept();
            logger.info("Server is accepting connection ...");
            ServerThread thread = new ServerThread(socket, this);
            save(thread);
            logger.info("Thread started");
        }
    }
}
```

3. Fehlerbehandlung :

Ein Beispiel für eine Behandlung eines internen Fehlers kann man in der Klasse Hibernate finden bei den Methoden saveUser und savecurrentGame wo falls alles ok der eintrag committed wird aber falls ein fehler passiert in der Exception ein rollback durchgeführt wird was die Datenbank in einen Konsistenten zustand hinterlässt.

```
        }
        session.saveOrUpdate(r);
        session.flush();

    }

    tx.commit();

    catch(Exception e) {
        e.printStackTrace();
        tx.rollback();
    } finally {
        if(session != null) {
            session.close();
        }
    }
}
```

Ich habe keine eingabeFehler behandelt da überhaupt nichts eingegeben werden soll beim Server, für die Klienten gibt es ein Feld wo man die SpielerID eingeben kann aber beim Server habe Ich das weggelassen um es benutzerfreundlicher zu machen.

Fehler die beim Austausch von nachrichten passieren sind nicht so leicht zu behandeln, weil es meistens bitstream Fehler sind und das würde heissen das man den Klienten sagen müsste das er sein Paket nochmals schickt wie bei einer TCP Verbindung was sehr kompliziert wäre deswegen werden diese Fehler nicht abgefangen, die einzige Prüfung die existiert ist das die Messages auf null Werte geprüft werden :

```

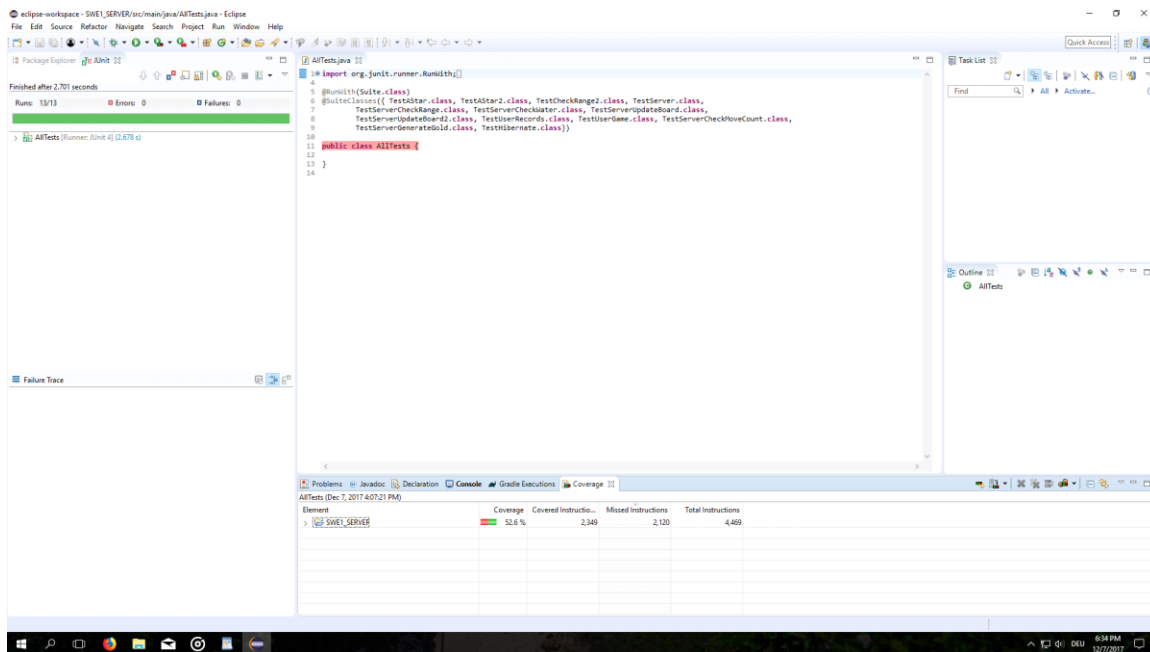
while((message = (Message)input.readObject())!=null)
{

    if(!accountSet && message.PlayerID != 0) {

```

4. Unit Tests :

Ich habe alle Test casses in ein Suit Test namens AllTests eingefügt wo alles dann aufeinmal getestet wird.



Beschreibung der einzelnen Tests :

1. TestAStar :

Ich lege bei diesen Test ein paar Wasserfelder an also die "blocked" Zellen besagen welche felder nicht erreichbar sind. Dann versucht der AStar zu finden ob es einen Weg gibt von einer Anfangsposition zur Destination. Da die Felder blockiert sind soll es nicht möglich sein einen Weg zu finden.

2. TestAStar2 :

Hier passiert das selbe nur soll es hier möglich sein einen Weg zu finden.

3. TestCheckRange2 :

Hier werden die werte xNewPos und yNewPos auf Werte gesetzt die sich auserhalb des erlaubten Bereichs befinden und das soll ein false zurückliefern.

4. TestHibernate :

Meine Datenbank funktioniert so das daten gesammelt werden und dann in einen Vector gespeichert werden der dann am Ende wenn jemand verliert in die Datenbank eingefügt werden, Ich finde es so effizienter und hier wird getestet ob der vector in Hibernate auch alle Daten hat den der testvector hat.

5. TestServer :

Es ist sehr schwer den Server zu Testen weil auch Klienten gebraucht werden vorallem den ServerThread weil da eine Connection benötigt wird. Hier habe Ich einfach eine Map, Message und Server erstellt und getestet ob die Map ok ist, somit wird auch getestet ob der Server überhaupt erstellt wurde.

6. TestServerCheckMoveCount :

Hier wird getestet ob die Anzahl an Schritten 200 überschreitet.

7. TestServerCheckRange :

Ist genau wie das andere Beispiel nur hier wird mit einen validen Wert getestet.

8. TestServerCheckWater :

Hier wird einfach getestet ob der Spieler ins Wasser gelaufen ist, eine Zelle wird

auf 1 gesetzt was Wasser heist und der Spieler setzt seine neue Position genau auf dieses Feld was heist er ist ins Wasser gelaufen.

9. TestServerGeneratedGold :

Hier wird getestet ob das Gold auf einen Feld plaziert wurde das Land ist, also halt nicht auf Wasser oder einen Berg.

10. TestServerUpdateBoard :

Hier werden Koordinaten übergeben vom Klienten an den Thread der dann das Board updated am Server und hier wird getestet ob die Map die im Server gespeichert ist der Map entspricht den der thread geschickt hat.

11. TestUserGame :

Hier wird einfach der Konstruktor der Klasse getestet.

Teilaufgabe 2 :

Die Datenbank sieht so aus. Sie besteht aus 2 Klassen, eine die den Spieler speichert und die andere speichert das Spiel also alle züge die durchgeführt wurden. Ich fand Hibernate ziemlich langsam aber da wir es unbedingt nutzen müssen habe Ich es so gemacht indem ich während des Spiels die Daten in einen Vector speichere und dann das ganze über die Klasse hibernate mit den Methoden save speichere. Dieser Vector wird dan in diesen Methoden ausgelesen und in die Datenbank eingefügt.

Codebeispiel für das speichern in der Klasse Server :

```

mapOut.playersOnlineAndPlayerOneStarts = true;
String move = "Position X : " + message.xNewPos + " Position Y : " + message.yNewPos + " ";
String startPos = "Position X : " + message.xPositionPlayerCastleOne + " Position Y : " + message.yPositionPlayerCastleOne + "
CurrentGame g = new CurrentGame(moveCount, startPos, "00", move, "N", "?", message.PlayerID);
if(moves.isEmpty()) { moves.add(g); }
if(moves.lastElement().PlayerID != g.PlayerID) {
    moves.add(g);
}

```

Es wird ein neues CurrentGame erzeugt mit den Daten und dann in den vector eingefügt.

Speichern am ende des Spiels :

```

if(message.stopGame) {
    stopped = true;
    PlayerOne.sendMessage(message);
    PlayerTwo.sendMessage(message);
    save.saveUser(users);
    save.saveCurrentGame(moves);
    save.print(PlayerID_1, PlayerID_2);
    return;
}

```

Wenn das Spiel stoppt werden dann die Methoden in der Hibernate Klasse aufgerufen. saveUser, Game und print was ein file erzeugt vom Spielverlauf.

So sieht dann das produzierte "gameFile" aus :

Player 1 :

- PlayerID : 27 | GamesPlayed : 1 | GamesWon : 0 |
NoOfGamesPlayerMadeErrors : 0 | Raiting : -1-

Player 2 :

- PlayerID : 16 | GamesPlayed : 1 | GamesWon : 0 |
NoOfGamesPlayerMadeErrors : 0 | Raiting : -1-

Game protocol :

- MoveID : 1 | startPos : Position X : 0 Position Y : 0 | EndPos : 00 |
MovePlayed : Position X : -99 Position Y : -99 | Direction : N | Discovery : ? |

PlayerID : 16-

- MoveID : 2 | StartPos : Position X : 0 Position Y : 0 | EndPos : 00 |
MovePlayed : Position X : 2 Position Y : 4 | Direction : N | Discovery : ? |
PlayerID : 27-

- MoveID : 4 | StartPos : Position X : 0 Position Y : 0 | EndPos : 00 |
MovePlayed : Position X : 6 Position Y : 4 | Direction : N | Discovery : ? |
PlayerID : 16-

- MoveID : 6 | StartPos : Position X : 0 Position Y : 0 | EndPos : 00 |
MovePlayed : Position X : 2 Position Y : 5 | Direction : N | Discovery : ? |
PlayerID : 27-

- MoveID : 8 | StartPos : Position X : 0 Position Y : 0 | EndPos : 00 |
MovePlayed : Position X : 6 Position Y : 5 | Direction : N | Discovery : ? |
PlayerID : 16-

- MoveID : 10 | StartPos : Position X : 0 Position Y : 0 | EndPos : 00 |
MovePlayed : Position X : 2 Position Y : 6 | Direction : N | Discovery : ? |
PlayerID : 27-

- MoveID : 12 | StartPos : Position X : 0 Position Y : 0 | EndPos : 00 |
MovePlayed : Position X : 6 Position Y : 6 | Direction : N | Discovery : ? |
PlayerID : 16-

- MoveID : 14 | StartPos : Position X : 0 Position Y : 0 | EndPos : 00 |
MovePlayed : Position X : 2 Position Y : 7 | Direction : N | Discovery : ? |
PlayerID : 27-

- MoveID : 16 | StartPos : Position X : 0 Position Y : 0 | EndPos : 00 |
MovePlayed : Position X : 6 Position Y : 7 | Direction : N | Discovery : ? |
PlayerID : 16-

- MoveID : 18 | StartPos : Position X : 0 Position Y : 0 | EndPos : 00 |
MovePlayed : Position X : 2 Position Y : 0 | Direction : N | Discovery : ? |
PlayerID : 27-

- MoveID : 20 | StartPos : Position X : 0 Position Y : 0 | EndPos : 00 |
MovePlayed : Position X : 6 Position Y : 0 | Direction : N | Discovery : ? |
PlayerID : 16-

- MoveID : 22 | StartPos : Position X : 0 Position Y : 0 | EndPos : 00 |
MovePlayed : Position X : 2 Position Y : 1 | Direction : N | Discovery : ? |
PlayerID : 27-

- MoveID : 24 | StartPos : Position X : 0 Position Y : 0 | EndPos : 00 |
MovePlayed : Position X : 6 Position Y : 1 | Direction : N | Discovery : ? |
PlayerID : 16-

- MoveID : 26 | StartPos : Position X : 0 Position Y : 0 | EndPos : 00 |
MovePlayed : Position X : 2 Position Y : 2 | Direction : N | Discovery : ? |
PlayerID : 27-

- MoveID : 28 | StartPos : Position X : 0 Position Y : 0 | EndPos : 00 |
MovePlayed : Position X : 6 Position Y : 2 | Direction : N | Discovery : ? |
PlayerID : 16-

- MoveID : 30 | StartPos : Position X : 0 Position Y : 0 | EndPos : 00 |
MovePlayed : Position X : 2 Position Y : 3 | Direction : N | Discovery : ? |
PlayerID : 27-

- MoveID : 32 | StartPos : Position X : 0 Position Y : 0 | EndPos : 00 |
MovePlayed : Position X : 6 Position Y : 3 | Direction : N | Discovery : ? |
PlayerID : 16-

- MoveID : 34 | StartPos : Position X : 0 Position Y : 0 | EndPos : 00 |
MovePlayed : Position X : 2 Position Y : 4 | Direction : N | Discovery : ? |
PlayerID : 27-

- MoveID : 36 | StartPos : Position X : 0 Position Y : 0 | EndPos : 00 |
MovePlayed : Position X : 6 Position Y : 4 | Direction : N | Discovery : ? |
PlayerID : 16-

- MoveID : 38 | StartPos : Position X : 0 Position Y : 0 | EndPos : 00 |
MovePlayed : Position X : 2 Position Y : 5 | Direction : N | Discovery : ? |
PlayerID : 27-

- MoveID : 40 | StartPos : Position X : 0 Position Y : 0 | EndPos : 00 |
MovePlayed : Position X : 6 Position Y : 5 | Direction : N | Discovery : ? |
PlayerID : 16-

- MoveID : 42 | StartPos : Position X : 0 Position Y : 0 | EndPos : 00 |
MovePlayed : Position X : 2 Position Y : 6 | Direction : N | Discovery : ? |
PlayerID : 27-

- MoveID : 44 | StartPos : Position X : 0 Position Y : 0 | EndPos : 00 |

MovePlayed : Position X : 6 Position Y : 6 | Direction : N | Discovery : ? |
PlayerID : 16-

Code in der Klasse Hibernate :

Für das Speichern in der Datenbank

```
session = sessionFactory.openSession();
tx = session.beginTransaction();

for(CurrentGame g : v) {

    //CurrentGame g = (CurrentGame) session.get(Current
    //if(g == null) {
    //CurrentGame g = new CurrentGame(MoveID, StartPo
    //}else {
    // g.MoveID++;
    //}
    //session.saveOrUpdate(g);
    session.save(g);
    session.flush();

}

tx.commit();
```

Für das Printen("gameFile") :

```

    }

    PrintWriter pw = new PrintWriter("gameFile.txt");

    configureSessionFactory();
    Session session = null;
    Transaction tx = null;

    session = sessionFactory.openSession();
    tx = session.beginTransaction();

    UserRecords u = (UserRecords) session.get(UserRecords.class, id1);
    UserRecords u2 = (UserRecords) session.get(UserRecords.class, id2);

    pw.println();
    pw.println("Player 1 : ");
    pw.println(u);
    pw.println();
    pw.println("Player 2 : ");
    pw.println(u2);

    pw.println();
    pw.println("Game protocol : ");
    pw.println();



















    for(CurrentGame g : moves){
        pw.println(g);
        pw.println();
    }

    pw.close();

    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

Die Daten werden auch in einen einfachen text file gespeichert um es einfach zu halten da kein Pfad angegeben ist sollte sich das file normalerweise im Projekt befinden.

	.gradle	12/7/2017 4:05 PM	File folder	
	.settings	12/7/2017 4:05 PM	File folder	
	bin	12/7/2017 4:06 PM	File folder	
	build	12/7/2017 4:05 PM	File folder	
	gradle	12/7/2017 4:05 PM	File folder	
	res	12/7/2017 4:05 PM	File folder	
	src	12/7/2017 4:05 PM	File folder	
	.classpath	12/7/2017 4:06 PM	CLASSPATH File	1 KB
	.project	10/13/2017 5:03 PM	PROJECT File	1 KB
	build.gradle	11/9/2017 10:35 AM	GRADLE File	1 KB
	database	11/9/2017 1:44 PM	Data Base File	0 KB
	file	11/9/2017 2:13 PM	TXT File	1 KB
	gameFile	12/7/2017 4:07 PM	TXT File	0 KB
	gradlew	10/13/2017 5:02 PM	File	6 KB
	gradlew	10/13/2017 5:02 PM	Windows Batch File	3 KB
	LogFile	11/26/2017 5:04 PM	Text Document	15 KB
	server	12/7/2017 4:07 PM	Data Base File	0 KB
	settings.gradle	10/13/2017 5:02 PM	GRADLE File	1 KB

Die Business Rules werden zentral überprüft es gibt zwei Methoden eine für die Map und die zweite für die Spielzüge.

In der Methode checkBoard wird die Map überprüft, zB. ob es zu wenig Wasserfelder oder Bergfelder gibt, usw.

1 :

Code-Beispiel :

```

// and also count the amount of water land and mountain
for(int i = 0; i < DIMENSION_X/2; i++)
{
    for(int j = 0; j < DIMENSION_Y; j++)
    {
        if(message.Map.coordinates[i][j] == 1)
        {
            ++countWater;
            blocked[i][j] = -1;
        }
        if(message.Map.coordinates[i][j] == 0)
        {
            ++countLand;
        }
        if(message.Map.coordinates[i][j] == 2)
        {
            ++countMountain;
        }
    }
}

```

Hier werden alle felder die Wasser erhalten auf blocked gesetzt und es wird auch durchgezählt wieviele Felder vom jeden Typ es gibt. Dann wird mit AStar nachgeschaut ob es einen Pfad zu diesen Feldern gibt, also es wird geschaut ob es von irgendwelcher Landfläche einen Pfad zu allen anderen Landflächen gibt, das heist es gibt keine Inseln. Methode : checkBoard.

2 :

Code-Beispiel :

```

//3 Business Rule
//check if the minimum amount of land, water and mountain fields is reached
if(countLand < MIN_VALUE_LAND || countMountain < MIN_VALUE_MOUNTAIN || countWater < MIN_VALUE_WATER) {
    message.lost_Player = message.PlayerID;
    message.stopGame = true;
    return false;
}

return true;

```

Hier wird dann nachgeschaut ob die Anzahl der Flächen den Minimum entspricht. Methode : checkBoard.

3 :

Code-Beispiel :

```

for(int i = 0; i < DIMENSION_X/2; i++)
{
    for(int j = 0; j < DIMENSION_Y; j++)
    {
        if(message.Map.coordinates[i][j] == 1)
        {
            ++counter;
            if(counter > 3) {
                message.lost_Player = message.PlayerID;
                message.stopGame = true;
                return false;
            }
        }
    }
    counter = 0;
}

```

hier wird geschaut ob sich mehr als 3 Wasserfelder auf der langen Achse befinden, das gleiche wird auch für Spieler 2 gemacht. Methode : checkBoard.

So werden alle Spielzüge zentral überprüft, man kann leicht was addieren oder wegmachen.

```

public boolean checkMove(Message message) {

    if(!checkWater(message)) return false;
    if(!checkRange(message)) return false;
    if(!checkMoveCount(message)) return false;
    if(!checkPlayerDirection(message)) return false;
    if(!checkIfMountain(message)) return false;

    return true;
}

```

4 :

Code-Beispiel :

```

//checking if the client moved onto water
if(map.coordinates[message.xNewPos][message.yNewPos] == 1) {
    UserRecords u = new UserRecords(message.PlayerID, 0, 0, 0, 0);
    users.add(u);
    return false;
}
return true;

```

Hier wird geschaut ob der nächste Spielzug in ein Wasserfeld zeigt, falls ja werden

die Daten an die Datenbank übertragen, das Spiel wird gestoppt. Methode : `checkWater`.

5:

Code-Beispiel :

```
if(message.xNewPos > DIMENSION_X || message.yNewPos > DIMENSION_Y) {
    UserRecords u = new UserRecords(message.PlayerID, 0, 0, 0, 0);
    users.add(u);
    return false;
}
return true;
```

Es wird geschaut ob der Spieler ein Feld betreten will das sich ausserhalb des Feldes befindet, indem einfach nachgeschaut wird ob seine neue Position ausserhalb der Grenzen geht. Methode : `checkRange`.

6:

Code-Beispiel :

```
if(moveCount>MAX_MOVES) {
    UserRecords u = new UserRecords(message.PlayerID, 0, 0, 0, 0);
    users.add(u);
    return false;
}
return true;
```

Es wird geschaut ob der Spieler die maximale anzahl an Spielzügen erreicht hat.
Methode : checkMoveCount.

7:

Code-Beispiel :

```
UserRecords u = new UserRecords(message.PlayerID, 0, 0, 0, 0);
if(message.xPositionPlayerOne != -99 && message.yPositionPlayerOne != -99 && message.xPositionPlayerTwo != -99 && message.yPositionPlayerTwo != -99)
if(message.PlayerID == PlayerID_1) { //we logically assume that not both y and x can be changed at the same time since we can not move in both directions at the same time
    /*if((((message.xNewPos == message.xPositionPlayerOne + 1) || (message.xNewPos == message.xPositionPlayerOne - 1)) && message.yNewPos == message.yPositionPlayerOne) ||
    || (((message.yNewPos == message.yPositionPlayerOne + 1) || (message.yNewPos == message.yPositionPlayerOne - 1)) && message.xNewPos == message.xPositionPlayerOne))
    System.out.println("Direction ERROR PlayerID 1 : " + message.PlayerID);*/
    if(!aStar.test(testNumber++, 8, 8, message.xPositionPlayerOne, message.yPositionPlayerOne, message.xNewPos, message.yNewPos, bl))
        users.add(u);
    return false;
}
}
```

Es hat nicht das ganze Bild reingepasst aber der Gedanke von dieser Methode ist es zu schauen ob sich ein Spieler diagonal bewegt hat, das darf er nicht also kann es nur in der x oder y Achse bewegen und das wird hier geprüft. Methode : checkPlayerDirection.

8 :

Code-Beispiel :

```
//Check if on mountain has to play two times
UserRecords u = new UserRecords(message.PlayerID, 0, 0, 0, 0);
if(message.xPositionPlayerOne != -99 && message.yPositionPlayerOne != -99 && message.xPositionPlayerTwo != -99 && message.yPositionP
if(message.PlayerID == PlayerID_1) {
    if(map.coordinates[message.xPositionPlayerOne][message.yPositionPlayerOne] == 2 && !p1_Mountain_1) {
        p1_Mountain_1 = true;
        return true;
    }
    if(map.coordinates[message.xPositionPlayerOne][message.yPositionPlayerOne] == 2 && p1_Mountain_1) {
        if(message.xNewPos != message.xPositionPlayerOne || message.yNewPos != message.xPositionPlayerOne)
            p1_Mountain_1 = false;
        users.add(u);
        return false;
    }
}
if(message.PlayerID_2 == PlayerID_2) {
    if(map.coordinates[message.xPositionPlayerTwo][message.yPositionPlayerTwo] == 2 && !p2_Mountain_1) {
        p2_Mountain_1 = true;
        return true;
    }
    if(map.coordinates[message.xPositionPlayerTwo][message.yPositionPlayerTwo] == 2 && p2_Mountain_1) {
        if(message.xNewPos != message.xPositionPlayerTwo || message.yNewPos != message.xPositionPlayerTwo)
            p2_Mountain_1 = false;
        users.add(u);
        return false;
    }
}
```

hier wird geschaut ob sich ein Spieler falls auf einen Berg 2x dort auch bewegt indem einfach gemerkt wird ob er auf einen berg ist, das wird gespeichert und dann beim zweiten mal nachgeschaut, falls alles ok dann werden die Variablen wieder auf false gesetzt und es kann von vorne losgehen. Methode : checkIfMountain.

Es wird auch an anderen stellen geprüft ob Business-Rules befolgt werden vorallem in der Klasse ServerThread aber das wollte Ich nicht in die Doku hinzufügen weil sie nicht so explizit angeführt sind.

Falls ich das Projekt nochmals machen würde würde ich einen Schalter einbauen für die Business-Rules so das man sie aktivieren und deaktivieren kann und nicht einfach im Code auskommentieren.

Netzwerkkommunikation :

Wie im ersten teil beschrieben habe Ich nur eine Nachricht mit der Ich alles abdecken kann, diese Nachricht ist etwas gros aber es macht den Code einfacher und es ist auch viel einfacher die Nachrichten zu verarbeiten.

Es gibt eine Klasse Message in der alle Daten gespeichert sind : Positionen von beiden Spielern, die neue position die der Spieler spielen möchte, IDs, Karte usw.

Code-Beispiel :

```
public ViewMap Map;

public int xPositionPlayerOne = -99;

public int yPositionPlayerOne = -99;

public int xPositionPlayerTwo = -99;

public int yPositionPlayerTwo = -99;

public int xPositionPlayerCastleOne;

public int yPositionPlayerCastleOne;

public int xPositionPlayerCastleTwo;

public int yPositionPlayerCastleTwo;

public int xPositionPlayerTreasureOne;

public int yPositionPlayerTreasureOne;

public int xPositionPlayerTreasureTwo;

public int yPositionPlayerTreasureTwo;

public int xNewPos = -99;

public int yNewPos = -99;

public boolean playersOnlineAndPlayerOneStarts = false;

boolean won_lost_P1 = false;

public int won_Player = 0;

public int lost_Player = 0;

boolean won_lost_P2 = false;

boolean stopGame = false;
```

Der Austausch geht über Sockets, es gibt eine run Methode auf der Seite des Servers und Klienten die beide auf Nachrichten warten und sie dann verarbeiten.

```

Message message = null;

while((message = (Message)input.readObject())!=null)
{

```

hier wartet der ServerThread auf ankommende Nachrichten ungleich null.

Wenn eine Nachricht kommt dann wird sie bearbeitet, zb. am Anfang des Spiels wird die Karte erwartet und die Spieler IDs die dann im Server gespeichert werden :

```

if(!accountSet && message.PlayerID != 0) {
    if(this == server.PlayerOne) {
        server.PlayerID_1 = message.PlayerID;
        message.PlayerID_1 = message.PlayerID;
        UserRecords u = new UserRecords(message.PlayerID, 0, 0, 0, -1);
        server.users.add(u);
    }
}

```

oder wenn die Maps schon da sind und der Spieler spielen möchte :

```

if(server.isPlayerTurnAndChangePlayer(this))
{
    server.logger.info(" Player Request To Move And Approved : " + message.PlayerID);
    if(!server.checkMove(message)) {
        message.lost_Player = message.PlayerID;
        message.stopGame = true;
    }
    server.updateBoard(message);
    server.logger.info("ServerThread Sent New Player Position");
}else {
    server.logger.info("Player Tried To Play But Not His Turn : " + message.PlayerID);
}
}

```

wird geschaut ob der Spieler der spielen möchte auch dran ist. Dann wird nachgeschaut ob der Spielzug in Ordnung ist und falls ja wird die neue Map aka Position zu beiden Spielern wieder gesendet.

Anleitung :

Sie können das Projekt importieren in Eclipse und dann über die Klasse runServer starten wie im Video angezeigt.

Falls sie die Datei über das Terminal kompilieren wollen dann bitte 3 Terminals aufmachen einen für den Server und 2 für die Klienten.

Dann mit `java -jar <Name>.jar` ausführen.

Es soll zuerst der Server gestartet werden, dann ein paar Sekunden warten und dann einen Klienten starten und dann den zweiten.

Es wird nicht kompilieren ohne einen Graphic-Driver also kann dann folgendes kommen : No X11 DISPLAY variable was set.

Bitte in den Zip File uner Anleitung die Screenshots und Videos anschauen!!!!