Kartengenerierung: In der ClientHalfMapGenerator Klasse wurde die Karte zufällig generiert, ohne irgendwelche Regeln zu beachten. Im Konstruktor werden die Methoden für die Kartengenerierung und Burgplatzierung aufgerufen. Die ClientHalfMapValidator übernimmt die Funktionen der Validierung, und schaut, ob am Rande der Karte zuviele Wasserfelder sind oder die Burg richtig platziert ist oder die angegebene Anzahl an Terrains beachtet wird. Zur Inselerkennung und Verarbeitung habe ich den FloodFill Algorithmus verwendet. Da habe ich auf Wikipedia nachlesen müssen was der Algorithmus macht und welche Variationen es gibt. Ich habe mich für die 8-Neighbour entschieden, weil aus irgendeinem Grund bei der 4-Neighbour ich dauernd Inseln bekommen habe. Und ich habe hier ein BFS verwendet um die adjazenten Nachbaren der Insel zu finden, und falls sie Wasserfelder waren mit einem Terraintypen zu konvertieren.

Beim Algorithmus zur Wegfindung habe ich am längsten gebraucht. Ich habe einen Dijkstra verwendet der eine HashMap für adjazente Kartenfelder haben sollte. Die HashMap besteht aus einem PathNode, welcher auf Path Klasse „gemapped“ ist. Diese Path Klasse beinhaltet eine Liste von besuchten und nicht besuchten Feldern. Eigentlich bin ich bei der Umsetzung am meisten Stolz auf meine Strategy Pattern, auch wenn sie sehr ähnlich ausschaut und auf die Strategien, auf die ich gekommen bin. Zum Beispiel der AbstractMapEvaluator der unsere Seite der Karte ermittelt anhand der Position unserer Burg, um den Schatz bzw. die gegnerische Burg einfacher zu finden. Oder auch die Evaluierung der Bergen bei den Klassen MovementToUnknownEnemyFort oder MovementToUnknownTreasure Klasse. Hier werden Berge ermittelt, die potenziell den Schatz entdecken könnten. Hier habe ich dann die adjazenten Grassfelder zu den Bergen in meine besuchte Felder Liste gespeichert, falls der Schatz nicht gefunden wurde. Und die Evaluierung der Berge lief, in dem ich die Distanz zwischen den einzelnen Bergen mit dem Pythagoras gerechnet habe und dann die Grassfelder um den Berg gezählt habe. Dabei habe ich die Berge in einer Queue gespeichert und die als Ziel für meinen Dijkstra gesetzt. Auch wurde ein Bestrafungssystem im Dijsktra eingeführt, um den Spieler zu bestrafen, falls er besuchte Felder betritt. Da wusste ich aber nicht wie ich die Kosten für die Bestrafung setzten sollte. Ich hatte Probleme bei der Findung des kürzesten Weges gehabt. Ich wusste nicht, dass ich eine die Vorgänger der Felder im möglichen Weg setzten musste und vom Ziel ein „Backtrack“ durchzuführen zu der Startposition.

Netzwerk ist eine eigene Klasse (ClientNetwork) bekommt von der GameController Klasse Instruktionen und sendet Anfragen an den Server. Die Konvertierung habe ich in zwei Interfaces aufgeteilt, um die Konvertierung vom und zum Server aufzuteilen. Weil manche Klassen brauchen keine Konvertierung zum andere bräuchten wieder keine Konvertierung vom. Und es wäre dann nicht sinnvoll diese Klassen diese Methoden implementieren zu lassen. Anfangs habe ich mit ordinal() konvertiert, aber dann habe ich gemerkt, dass das nicht der beste Weg ist zu konvertieren.

MVC
Modell: Game
View: GameVisualisation
Controller: GameController
Ich hatte eigentlich nicht viele Modelle, eigentlich läuft alles über mein Game. Die Konvertierung der Spieler Objekte, Karten Objekte und der Spielfigur läuft nur in der Game Klasse. Mein Problem war die Ausgabe der Karte durch den firePropertyChange. Bei Verwendung von Ansi-Codes für die Ausgabe war alles immer so chaotisch und ich konnte die Ausgabe dann nicht lesen. Deswegen musste ich die Ausgabe der GameMap umändern.