Programmierpraktikum 2010 – Gruppe 19

Projektbericht

Conquest of Runes – Bedienungsanleitung

Spielziel

Spielziel von Conquest of Runes ist es möglichst viele, der quer über die Karte verteilten Runen in seinen Besitz zu bringen. Es gibt genau so viele Runen wie Spieler.
Hat ein Spieler keine Rune mehr, beginnt ein zwölfminütiger Countdown. Der Spieler kann den Countdown stoppen, indem er (mindestens) eine beliebige Rune zurückerobert. Gelingt es ihm nicht, hat er verloren und wird vom Spiel ausgeschlossen.
Hat ein Spieler genügend Runen erobert, um zu gewinnen, beginnt ebenfalls ein zwölfminütiger Countdown. Dieser kann von den andern Spielern gestoppt werden, indem sie dem Spieler eine oder mehrere Runen abnehmen. Am Ende des Countdowns gewinnt der Spieler das Spiel.

Anzahl Runen zum Sieg:

Spieleranzahl:	Nötige Runen:
2	2
3	3
4	3
5	3
6	4
7	4
8	5

Wenn in Teams gespielt wird, muss ein Spieler des Teams alle Runen besitzen, um das Spiel zu gewinnen.

Spielstart

Zu Beginn muss sich jeder Spieler bei der Datenbank einloggen. Dazu muss im Login-Schirm der Nickname und Passwort eingegeben werden. Beim ersten Spielen müssen zusätzlich Name und Vorname zur Erstregistrierung angegeben werden.
In der Lobby können Chatnachrichten ausgetauscht und Spiele verabredet werden. Möchte ein Spieler in Spiel eröffnen, klickt er auf den Spiel erstellen Knopf. Er wird dadurch zum Administrator des Spiels und kann die nötigen Einstellungen vornehmen.
Anschließend können die anderen Spieler dem Spiel beitreten und die Änderungen mit verfolgen.
In den Spieleinstellungen kann der Administrator nun die Anzahl der Computergegner (durch Aktivieren des dritten Kontrollkästchens) und Spieleinstellungen wie Startrohstoffe, Sichtbarkeit der Karte und den Kartentyp vornehmen. Verlässt der Administrator das Spiel, wird automatisch ein Nachfolger bestimmt.

Zur Karte

Es gibt grundsätzlich drei verschiedenen Kartentypen.
Große Inseln, mit überwiegend Festland, aber auch Meer.
Kleine Inseln, mit vielen kleinen Inseln und viel Wasser.
Festland, mit ausschließlich festem Boden und Flüssen.

Zur Sichtbarkeit

Bei Auswahl „Nebel des Krieges“ können später im Spiel nur erkundete Bereiche gesehen werden, der Rest der Karte ist verdeckt. Zuvor erkundete und wieder verlassene Bereiche werden grau und ohne die dort befindlichen Einheiten dargestellt.

Außerdem ist es möglich das Spiel in Teams zu spielen. Mehrere Spieler kämpfen dann zusammen um die Vorherrschaft über die Runen. Die Teamzugehörigkeit kann nach Aktivierung der Auswahl „Teamauswahl“ durch den Administrator jeder Spieler selber vornehmen. Die Teams der Computergegner gibt der Administrator in das entsprechende Kästchen ein.
Überflüssige Slots müssen geschlossen werden, damit der „Spiel starten“-Knopf erscheint. Mit Klick auf den Selbigen beginnt das Spiel.

Im Spiel

Grundsätzlich werden Einheiten, Gebäude und sonstige Objekte mit einem Linksklick ausgewählt. Um eine (oder mehrere) aktivierte Einheiten zu bewegen, muss mit der rechten Maustaste geklickt werden.
Über die Karte gescrollt wird, indem man die rechte Maustaste gedrückt hält und über das Spielfeld zieht.

Der Spielschirm

Der Spielschirm besteht im Wesentlichen aus vier Komponenten. Oben links aus der klickbaren Minimap, an der linken Seite der Toolbar, die Informationen und Aktionsmöglichkeiten der aktivierten Einheiten anzeigt, dem Chat am unteren Bildrand und dem Hauptfenster, in dem der aktuelle Kartenausschnitt gezeigt wird.

Rohstoffe

Es gibt fünf Rohstofftypen. Holz, Erz, Stein, Nahrung und Gold. Die ersten vier sind auf der Karte verteilt und eine erste Quelle findet sich direkt in der Nähe des Startpunktes. Diese vier Rohstoffe müssen mit Bauarbeitern abgebaut werden. Der Rohstoff Gold kann nicht abgebaut werden. Jede dem Spieler gehörende, noch lebende Einheit zahlt pro Minute 9 Gold Steuern. Wird mehr Gold benötigt, können die nur Nahrung kostenden Arbeiter schnelle Abhilfe schaffen. Außerdem wird Runenbesitzt belohnt. Für jede Rune gibt es pro Minute und Einheit ein Gold zusätzlich.
Für den Rohstoffabbau müssen beliebig viele, aber mindestens ein Bauarbeiter ausgewählt sein und anschließend auf eine Rohstoffquelle geklickt werden. Sind auch andere Einheitentypen aktiviert, werden diese automatisch aussortiert. Haben anschließend ein oder mehrere Arbeiter keinen Zugang mehr zu dieser Quelle (da voll belegt), suchen sie sich automatisch eine andere gleichen Typs. Wie viel Rohstoff ein Arbeiter aktuell bei sich trägt ist in der Toolbar zu sehen.

Dort sieht man nach Klick auf die Ressource ebenfalls wie viel Rohstoff in dieser noch vorhanden ist.
Trägt ein Arbeiter die maximale Menge an Rohstoffen (zehn), dann bringt er diese zu einem passenden Lagerort und arbeitet anschließend weiter. Ist eine Quelle leer, sucht sich der Arbeiter automatisch eine neue.
Die aktuellen Rohstoffvorräte sieht der Spieler unterhalb der Toolbar.

Gebäude

Jeder Spieler beginnt mit einem Marktplatz, in dem Bauarbeiter erstellt werden können.
Außerdem gibt es folgende Gebäude:
Barracke: Ausbildungsort für Ritter, Axtkämpfer und Bogenschützen.
Hafen: Ausbildungsort für Schiffe
Wachturm: greift automatisch feindliche Einheiten an, hat große Sichtweite
Stein-/Holz Lager, Mühle, Sägewerk: Lagerorte für Rohstoffe (Mühle – Nahrung)
Schmiede: hier können Verbesserungen für Kämpfer entwickelt werden

Um ein Gebäude zu bauen, muss ein Arbeiter aktiviert sein und anschließend in der Toolbar auf das Bausymbol (zwei gekreuzte Hämmer) geklickt werden. Nach Auswahl des Gebäudes kann dieses auf der Karte platziert werden. Es wird grün unterlegt, falls der ausgewählte Platz frei ist, sonst rot. Voraussetzung zum Bau ist, dass genügend Rohstoffe auf Lager sind.
Die Kosten für die Gebäudetypen werden direkt unter dem entsprechenden Symbol angezeigt.
Ein Arbeiter kann beim Bau von anderen unterstützt werden, indem diese aktiviert werden und anschließend auf die Baustelle geklickt wird. Die Namen der verschiedenen Gebäude werden par Hinweis auf der Karte angezeigt, sobald Sie angeklickt wurden.

Einheiten

Grundsätzlich werden Einheiten erstellt, indem nach Anklicken eines Marktes oder einer Barracke auf ein Einheitensymbol geklickt wird. Auch hier genügend Rohstoffe vorausgesetzt.
Es können beliebig viele Einheiten in Auftrag gegeben werden, diese werden nach und nach ausgebildet. Der aktuelle Fortschritt wird im Spielfenster oberhalb des Gebäudes angezeigt.

Eigenschaften

	Lebenspunkte	**Angriffspunkte**	**Besonderheit**
Arbeiter	100	3	
Ritter	250	20-25	Langsamste Einheit, schlechte Kampfhaltung am Tag (siehe unten)
Axtkämpfer	250	15-20	
Bogenschütze	150	9-14	Schnellste Einheit, schlechte Sicht in der Nacht (siehe unten)

Insgesamt besitzen die Einheiten verschiedene Geschwindigkeiten je nach Bodentyp auf dem sie sich gerade bewegen. Allgemein bewegen sich Einheiten auf Schnee und Eis langsamer als auf Gras oder über Brücken. Betreten Sie flüssige Lava oder Sand so verlangsamt sich ihre Geschwindigkeit noch einmal.

Bewegen/Kämpfen

Einheiten können einzeln oder als Gruppe über die Karte bewegt werden. Aktiviert man mehrere Einheiten und klickt auf ein Ziel, bewegen sich die Einheiten in einer Formation von

Dreierreihen dorthin. Die Geschwindigkeit richtig sich nach dem langsamten Mitglied. Die Gruppe bleibt dabei stets zusammen. Klickt man auf eine gegnerische Einheit bewegen sich die Einheiten als Gruppe dorthin und greifen dann an. Flüchtende Einheiten werden automatisch verfolgt. Stirbt eine Einheit, sucht der Angreifer automatisch nach einem neuen Ziel in der Nähe. Um Einheiten in einem Schiff zu transportieren, müssen sie in Ufer- und Schiffnähe platziert und dann mit Rechtsklick in das Schiff geladen werden. Analog verlassen die Einheiten das Schiff. Wird auf einen Zielpunkt geklickt, der nicht erreichbar ist, nähert sich die Einheit diesem Punkt so weit wie möglich.
Um eine Rune zu erobern, müssen zunächst alle feindlichen Einheiten aus dem Umkreis eliminiert und dann bei aktivierter Einheit, die sich in der Nähe der Rune befindet, auf die geklickt werden (Rechtsklick).

Chat

Durch Eingabe in das Eingabefeld unten und drücken der Eingabetaste oder des Sende-Knopfs wird eine Chatnachricht gesendet. Es besteht außerdem die Möglichkeit private Nachrichten zu senden. Beginnt eine Nachricht mit @nickname, wobei nickname für einen am Spiel teilnehmenden Spieler steht, erhält nur dieser die Nachricht.

Upgrades/ Tech-Tree

Jedes Gebäude kann mit einem Klick auf den Pfeil in der Toolbar ausgebaut werden. Dadurch erhält es mehr Lebenspunkte und ermöglicht den Bau von neuen verbesserten Einheitentypen. Zusätzlich können Einheiten in der Schmieder verbessert werden. In der ersten Entwicklungsstufe gewinnen sie zwei und in der zweiten drei Angriffspunkte hinzu.

Tageszeit:

Conquest of Runes enthält außerdem die Besonderheit, dass alle acht Minuten die Nacht über die Spielwelt hereinbricht. Der Bogenschütze, dadurch in seiner Sicht und Treffsicherheit beschränkt, verliert drei Angriffspunkte, der stark schwitzende Ritter am Tag, bei Sonnenschein, vier. Sonne und Mond wandern über der Karte am Horizont entlang und kennzeichnen so Tag und Nacht.

Die Arbeitsbereiche

Mapgenerator (Christoph Heuser)

Die Mapgenerierung

Wenn eine Instanz einer Map erstellt wird, so werden zwei Werte (Kartentypen und Anzahl der Spieler, egal ob Client oder Computer) der Map-Klasse übergeben.
Bei den Kartentypen werden vier Werte unterschieden: “Festland”, “Große Inseln”, ”Kleine Inseln” und “Zufall” ; hierbei wird nach festen Prozenten eine der drei anderen Karten ausgewählt. Die Spieleranzahl wird zur Berechnung der Mapgröße und weiteren Berechnungen benötigt .
Die Map besteht nun aus drei Ebenen (Fieldarray): Bodentyp, Ressourcen und Objekte (Figuren, Gebäude ect.). Diese drei Ebenen werden je nach Kartentyp gefüllt.

Festland (Methode: creatGroundContinent(..))

Zunächst wird der Bodentyp auf der ganzen Karte auf Gras gesetzt. Es werden nun die Startpunkte der Spieler (Marktplatz) und die Runenpunkte verteilt. Die Methode zur Verteilung der beiden Gebäude ist ähnlich. Es werden genauso viele Punkte wie Spieler auf einem Kreisrand um den Mittelpunkt der Karte gleichmäßig angeordnet, sodass der Abstand der benachbarten Punkte gleich ist. Von diesem Punkt aus wird eine zufällige Richtung und ein eingeschränkt zufälliger Abstand ermittelt. Dieser neue Punkt ist der gesuchte Startpunkt bzw. Runenpunkt. Der Radius für den Kreisrand hängt von der Größe der Map ab; zudem wird er für die Runenverteilung immer kleiner gewählt als für die Startpunktverteilung. Des weiteren hängt die Runenverteilung von der Spieleranzahl ab. So gibt es die Möglichkeit, dass bei vier Spielern die Runen in den Mapecken verteilt werden. Bei fünf Spielern wird eine Rune ins Zentrum gelegt und bei mehr werden die Restlichen (Spieleranzahl - 4) standardmäßig berechnet.

Große Insel (Methode: creatGroundBigIlands(..))

Bei diesem Kartentyp gibt es zwei Möglichkeiten, die teils von der Spieleranzahl, teils vom Zufall abhängen.
Zum einen besteht die Möglichkeit, dass eine große Insel die Karte ausfüllt. Hierbei gibt es zwei Inselmethoden die je nach Zufall benutzt werden. Die eine Insel (Typ1) hat als Grundform ein Quadrat, wobei der Rand dieser Insel durch Zufallszahlen kreiert wird. Die andere Insel (Typ2) ähnelt einem Kreis. Durch Parametrisierung der Kreisgleichung durch Sinus- ,Cosinus- und Quadratfunktionen, die wiederum von mehreren Zufallszahlen abhängen, entsteht Typ2.
Die zweite Möglichkeit sieht zwei Inseln vom Typ1 vor und zusätzlich zwei Ufer in Form von Kreissehnen am Maprand.
Nun werden wie beim Festland die Runen- und Startpunkte auf den Inseln verteilt. Für weniger als vier Spieler wird „Möglichkeit 1“ kreiert, für mehr Spieler sind beide Möglichkeiten möglich.
Wenn „Möglichkeit 2“ eintritt, werden bei gerader Anzahl die Spieler auf beide Inseln zu gleichen Teilen aufgeteilt.
Bei ungerader Spieleranzahl wird automatisch eine der beiden Inseln größer und mit einem Spieler mehr auf dieser Insel erstellt.

Kleine Insel (Methode: creatGroundSmallIlands(..))

In der Mitte wird eine Insel vom Typ1 mit zufälligem Radius mit den Runen und um diese Insel herum werden gleichmäßig kleinere Inseln von Typ1 und Typ2, auf denen die Startpunkt der Spieler sind, verteilt.

Die Karte besteht entweder aus "Gras" oder "Meer", Start- und Runenpunkten; nun werden die Hindernisse verteilt.

Die Berge (Methode: setMountain(..))

Es wird ein Quadrat mit einer zufälligen Seitenlänge von bis zu vier Feldern kreiert. Nun werden in diesem Quadrat zufällig Felder als Berg belegt. Die Anzahl der Quadrate hängt sowohl von der Spieleranzahl als auch vom Zufall ab. Berge sind auf dem Wasser und auf dem Land Hindernisse. Es wird darauf geachtet, dass ein gewisser Abstand zu den Startobjekten (Runen, Startpunkte) vorhanden ist.

Die Flüsse (Methode: setRiver(..))

Zunächst werden auf der Karte "Quellpunkte" zufällig berechnet. Die Anzahl dieser Punkte ist abhängig von der Spieleranzahl. Der Fluss wird mit Funktionen von diesem Quellpunkt aus berechnet. Durch Parametrisierung der Kombinationen von Sinus-, Kosinus-, Quadrat- und Gebrochenrationalenfunktionen werden die Wege der Flüsse berechnet. Durch weitere Zufallszahlen wird der Fluss noch gedreht, so dass er von dem Quellpunkt theoretisch in alle Richtungen fließen kann. Eine Brücke wird an dem nächstgelegenen Punkt von dem Mittelpunkt der Flusslänge angelegt, an dem der Fluss in vertikaler oder horizontaler Richtung nur ein Feld breit ist.

Nachdem die Hindernisse verteilt wurden, werden weitere Bodentypen verlegt.

Eis (Methode: setIce(..))

Zunächst wird eine Zufallszahl "Z1" berechnet, die darüber entscheidet, wie viel Eis auf der Map sein wird.
Nun wird die Karte in eine zufällige, von "Z1" abhängige Anzahl gleichgroßer Teilfelder zerlegt. In jedem dieser Teilfelder werden zufällige Punkte verteilt, die Eckpunkte von Polygonen sind. Die Fläche, die von diesen Eckpunkten eingeschlossen wird und bisher Gras war, wird nun mit Eis belegt. Damit kein Spieler einen Nachteil erfährt, wird ein gewisser Abstand zwischen dem Eis und den "Startobjekten" eingehalten.

Lava (Methode: setLava(..))

Die Anzahl der Lavafelder auf der Map hängt von "Z1", der Spieleranzahl und vom Zufall ab. Auf der Karte wird solange zufällig ein Punkt kreiert, bis dieser entweder gewisse Eigenschaften erfüllt oder die "Grenze der Versuche" überschritten wird. Das hat den Vorteil, dass man weder weiß, ob überhaupt ein solcher Punkt gefunden wird, noch wo diese Felder liegen, was und eine Variabilität der Landschaft ermöglicht. Dieser Vorteil geht auf Kosten der Laufzeit. Um diese möglichst gering zu halten, wird der Vorgang durch die "Grenze der Versuche" entsprechend eingeschränkt.
Die gewissen Eigenschaften für einen solchen Punkt sind dadurch gekennzeichnet, dass sie jeweils einen bestimmten Abstand zu Startobjekten, Eis, Berge und Wasser aufweisen.

Sand(Methode: setSand(..))

Die Verteilung des Sandes entspricht dem Prinzip der Eisverteilung. Die Anzahl und die Größe der Sandfelder hängen ebenfalls von “Z1” ab. Wenn “Z1” so berechnet wurde, dass viel Eis verteilt wird, werden automatisch weniger mögliche Sandflächen berechnet. Jeder Sandpunkt muss einen gewissen Abstand zum Eis haben. Damit an dieser Stelle die Laufzeit nicht “explodiert” wird nur in die vier Himmelsrichtungen geprüft, ob Eis vorhanden ist. Der Abstand zu Lava und Wasser ist bei der Sandmethode unerheblich.

Dreckiges Gras (Methode: setDirtyGrass())

Freie Grasflächen werden mit Hilfe von Polygonzügen zufällig mit dreckigem Gras belegt.

Nachdem die Lava verteilt wurde, werden zunächst die Ressourcen verteilt.

Ressourcen (Methode: setResources(..))

Zuerst werden in der Umgebung von jedem Startpunkt bestimmte Mengen aller Ressourcen verteilt (Nahrung, Stein, Erz, Holz). Anschließend werden diese auf der gesamten Karte verteilt, solange dort keine Lava, Wasser oder Hindernisse sind. Die Map wird wieder in Teilabschnitte zerlegt, in denen Ressourcen verteilt werden. Das Holz wird mit Hilfe von Polygonzügen, die einen Waldrand darstellen, verlegt. Stein und Erz werden nach dem selben Prinzip verteilt. Nahrung hingegen wird nur punktweise gesetzt. Alle Ressourcenmethoden ähneln sich zwar, unterscheiden sich aber in vielen Variablen. Dies ist der Grund dafür, dass jede Verteilung ihre eigene Methode hat, was wiederum der Übersichtlichkeit dient.

Zum Schluss wird die ganze Map mit all ihren Punkten und deren Inhalten zufällig gedreht und /oder gespiegelt.

All diese Berechnungen führen dazu, dass die Mapgenerierung auch mal eine halbe Minute dauern kann.

Der Server (Armin Gardyan)

Ki:

Die Ki ist in Form von zwei deterministischen Automaten programmiert. Einer regelt das Wirtschafts- und Aufbauverhalten, der andere die Kampfsteuerung (Verteidigen/ Angreifen) der Ki. Aufgaben der Einheiten, sowohl wirtschaftlicher (Rohstoffe holen) als auch militärischer Art (Rune/Basis verteidigen/ Angreifen) werden in HashMaps gespeichert, sodass bei Verlust einer Einheit direkt ein Rebuild des entsprechenden Status ausgeführt werden kann. Stirbt zum Beispiel ein Ritter, hat man direkten Zugriff auf die Information, welches Gebäude dieser bewacht hatte und direkt kann ein neuer Ritter ausgebildet werden, der diese Aufgabe übernimmt. Der zuvor letzte Status wird gemerkt und anschließend wieder in Kraft gesetzt. Damit Wirtschaft und Kampf gleichermaßen von abgebauten Rohstoffen profitieren, werden beide Automaten abwechselnd und in regelmäßigen Abständen pausiert. Muss die Ki auf Ereignisse reagieren, wird sofort der entsprechende Automat auf aktiv gesetzt.
Die Ki beginnt damit, Rohstoffe abzubauen, verteidigt dann die eigene Basis und eigene Rune, um dann Gegner anzugreifen. Mit Wahrscheinlichkeit von 30% wird ein Markt attackiert, mit 70% Wahrscheinlichkeit eine Rune. Dann wieder jeweils 30% zu 70% ein zufälliger, oder das am schwächsten bewachte Ziel.
Die Ki interagiert mit dem Server, indem sie Nachrichten in dessen Kommunikation fügt und bei Senden von Nachrichten von Serverseite von diesem per Methodenaufruf informiert wird.

Servergame:

Die Klasse Servergame verarbeitet die Nachrichten vom Client. Die Methode handleWrapper empfängt die Nachrichten und stellt dann entsprechende Events in den EventTimer, da so gut wie alle Aktionen nicht direkt ausgeführt werden, oder auch in der Zukunft stattfinden sollen. Außerdem initialisiert die StartGame Methode das Spiel und schickt Karte sowie Informationen über Teams und Sichtbarkeit an alle Clients. Die Nachrichtenverarbeitung macht einen großen Teil des Servers aus, da auf viele verschiedene Anfragen/Aufträge reagiert werden muss. Die Methode getDamage berechnet bei Kämpfen die neuen Lebenspunkte und löscht gegebenenfalls eine vernichtete Einheit. Die Methode delClient löscht bei Niederlage, oder Verbindungsabbruch alle Einheiten eines Spielers.
Die Methode AStar berechnet natürlich die Wege. Sie gibt für jede Anfrage den Punkt zurück, der dem Ziel am nächsten liegt und speichert die Wegpunkte in der WayHash, sodass diese für die weitere Bewegung (senden der einzelnen Wegpunkte vom EventTimer) zur Verfügung stehen.

EventTimer:

Der Eventtimer, der Events in chronologischer Reihenfolge abarbeitet ist maßgeblich für die Organisation des Spieles. Erhält er einmal ein Event vom Server, verarbeitet er dieses (abgesehen von getDamage) autonom und wirft falls nötig neue Events in die Zukunft.
Die Events zu jeder Einheit werden anhand der Id der Einheit in einer Hashmap abgelegt, sodass ausstehende Aufgaben dort gesammelt (oder auch gelöscht) werden können.
Besonders wichtig sind zwei „Gruppen“ von Methoden. Als erstes die „Start“ Methoden, die immer dann aufgerufen werden, wenn eine Einheit nach Bewegung am Zielpunkt ankommt. Die Start-Methoden prüfen dann, welche Aufgabe als nächstes ausstand und prüfen zunächst, ob Zielpunkt (Rohstoff, Ressource, Gebäude, Gegner) in Reichweite. Ist dies nicht der Fall, werden die Try-Methoden aufgerufen. Diese versuchen zunächst näher an das Ziel

heranzukommen. Gelingt dies nicht, ermitteln sie einen neuen Punkt (neue Ressource, neuen „Verbindungspunkt“ zum Gebäude, neuen Punkt von dem aus angegriffen werden kann) und bewegen die Einheit dorthin. Anschließend wird wieder die die entsprechende Start-Methode aufgerufen. So laufen alle Vorgänge (Angreifen, Bauen, Abbauen) nach dem gleichen Prinzip ab. Außerdem regelt der EventTimer auch den Ablauf des Countdowns zu Sieg oder Niederlage. Nebenbei enthält der EventTimer einige Methoden, die dazu dienen, zum Beispiel Gruppen von Einheiten zu formieren (initFormation) oder neue Ressourcen, Verbindungspunkte etc. zu finden (Search-Methoden), also Hilfmethoden, die sich in der Spielwelt „orientieren“.
Soll eine Einheit laufen, zählt der EventTimer die Schritte mit und berechnet automatisch nach fünf Schritten, oder falls der in der WayHash vom Server hinterlegte Weg nicht mehr frei ist (WayIsFree-Methode), den Weg neu.

Die Datenbank (Martin Kühn)

Die Datenbankverbindung des Servers wird über 3 Klassen realisiert. Hierbei baut die Klasse DatabaseConnect, wenn möglich, eine Verbindung zur Datenbank auf und gibt per Methode entsprechende Daten wieder. Die Methoden getEntries() und getHighscorelist() arbeiten hier völlig analog und rufen die Datenbankeinträge ab, wobei die erstere nach dem Primärschlüssel ID und die zweite nach den Punkten absteigend sortiert. Die Methode setNewUser() legt einen neuen Benutzer in der Datenbank an und die Methode updateStats() aktualisiert die Statistiken zu einem Namen.
Die Klasse Database ruft die Verbindungsklasse einmal auf und verwendet deren Methoden um die Informationen zwischenzuspeichern und damit mehrfachen Datenbankaufruf zu vermeiden. Hier rufen die Methoden getDbentries() und getHighscorelist() die zugehörigen Informationen aus der DatabaseConnect ab.
Darüber hinaus wird durch die Methode checkNick() überprüft, ob ein Nickname bereits existiert und authorizeUser() überprüft über getPWtoNick(), ob ein Passwort zum zugehörigen Namen passt. createUser() fordert die DatabaseConnect Instanz auf, einen neuen Benutzer zu erstellen.
Die Klasse DatabaseEntry stellt nur die Struktur eines Datenbankeintrags zur Verfügung, sowie deren Getter und Setter.

Die Kommunikation (Jennifer Bergsch)

Die Kommunikation arbeitet in Threads. Dies garantiert eine ‚parallele' Bearbeitung der Nachrichten.
Die Kommunikation zwischen Server und Client geschieht über die von Java bereitgestellte Klasse Socket, über die Nachrichten mithilfe von ObjectOutput-/-InputStreams versendet bzw. empfangen werden können.

Server-Kommunikation

Die wichtigsten Klassen der Server-Kommunikation sind zum einen die Klasse ServerNetwork, zum anderen die Klassen zum Empfangen (ServerReceiver) und zum Versenden (ServerSender) der Nachrichten.
Beim Erstellen einer neuen Kommunikation mithilfe des ServerNetworks wird permanent auf eingehende Verbindungsanfragen der Clients gewartet, dies geschieht in der Methode waitForConnection(), welche beim Starten des Threads des ServerNetworks direkt aufgerufen wird. Verbindet sich ein Client auf den Server, so wird der Client in einem Vector abgespeichert, ein ServerSender erstellt und über diesen ein ServerReceiver für diesen Client erzeugt. Diese werden wiederum in Threads realisiert. Zudem werden schon ObjectInput-/-OutputStream im Receiver bzw. Sender geöffnet, welche zum Empfangen/Senden benötigt werden.

Client-Kommunikation

Analog zur Server-Kommunikation gibt es hier auch die Klassen ClientNetwork, ClientServer und ClientReceiver.
Will sich der Client mit dem Server verbinden, so wird die Methode connect() im ClientNetwork aufgerufen. Hier wird über einen Socket die Verbindung hergestellt , ein ClientSender erzeugt und über diesen ein ClientReceiver erstellt, welche ebenso durch Threads realisiert werden. Zudem werden schon ObjectInput-/-OutputStream im Receiver bzw. Sender geöffnet, welche zum Empfangen/Senden benötigt werden.

Versenden und Empfangen von Nachrichten

Nachrichten können vom Server, wie auch von den Clients, durch den Aufruf der Methode sendMessage() im jeweiligen Network versendet werden, wobei der Server die Möglichkeit hat nur an einen (sendMessageToID()) oder an alle Clients (sendMessageToAll()) zu senden.
Das Senden bzw. Empfangen läuft bei Server und Client analog:
Wird die Methode sendMessage() aufgerufen, so wird die zu versendende Nachricht in der SendeSchlange (sendQueue) des jeweiligen Senders abgelegt. Die Threads ClientSender und ServerSender überprüfen permanent (bis zum Zeitpunkt stopped=true), ob noch Nachrichten in dieser Schlange vorhanden sind. Falls ja, wird die nächste Nachricht aus der Schlange entfernt und mithilfe des ObjectOutputStreams an den Server bzw. Client gesendet.
Die Threads ClientReceiver und ServerReceiver überprüfen ebenso bis zum Zeitpunkt stopped=true, ob Nachrichten über den ObjectInputStream eingehen. Eingehende Nachrichten werden in einer Empfangs-Schlange (rcvQ) abgelegt und können dann vom Client bzw. Server weiterbearbeitet werden.

Abgebrochene Verbindungen / Abmelden

Wenn Verbindungen abbrechen, wird auf Server- und Client-Seite analog die Methode connectionLost() im jeweiligen Network aufgerufen. Diese setzt die vorhin schon erwähnten Variablen stopped auf true, sodass kein Versuch mehr unternommen wird, Nachrichten zu

senden oder zu empfangen, damit keine neuen Fehler auftreten. Zudem werden die jeweiligen Sender- und Receiver-Threads geschlossen. Im ServerNetwork wird zudem der jeweilige ‚verlorene' Client aus dem Vector gelöscht.
Abmelden ist durch das Aufrufen der close()-Methoden möglich. In diesen wird die oben genannte Methode connectionLost() aufgerufen.

‚Ich bin noch da'-Nachrichten / Ping

Periodisches Versenden von ‚KeepAlive'-Nachrichten wird durch die zwei Threads PingTimer in der Client-Kommunikation und ConnectionChecker in der Server-Kommunikation realisiert. Diese werden in dem jeweiligen Network erstellt.
Der PingTimer versendet periodisch in einem Abstand von einer Minute eine KeepAliveMessage an den Server, damit der Server erkennt, dass die Verbindung des Clients noch besteht. Zudem wird eine Variable lastSend auf die aktuelle Zeit gesetzt. Im ServerReceiver wird beim Empfangen einer Nachricht zunächst immer überprüft, ob die empfangene Nachricht eine KeepAliveMessage ist. Falls ja, so wird die Nachricht zurückgesendet und eine Variable lastReceive auf die aktuelle Zeit gesetzt. Sobald die Nachricht im ClientReceiver empfangen wurde so kann mithilfe der Variablen lastSend und der aktuellen Zeit der Ping ausgerechnet und ausgegeben werden.
Der ConnectionChecker überprüft (anhand der Variablen lastReceive und der aktuellen Zeit) in einem Abstand von 2 Minuten, ob der Client sich durch eine KeepAliveMessage in den letzten 2 Minuten ‚gemeldet' hat. Falls nicht, wird die Verbindung zu diesem Client mithilfe der close()-Methode beendet.

Der Client (Martin Bartmann & Denis Röthig)

Die Hauptaufgabe des Clients ist das korrekte Analysieren und Weiterleiten von Eingaben des menschlichen Spielers, sowie Änderungen vom Server richtig zu verarbeiten.
Der Client besteht im Wesentlichen nur aus der Klasse ClientGame.java, die viele Methoden zur genaueren Analyse der Messages und Mausklicks enthält.
Die Eingaben des Spielers fängt die GUI ab und leitet diese an den Client weiter, dies betrifft vor allem Mausklicks, bei dem die Art, zum Beispiel linker Mausklick, und die Position übertragen wird. Es folgt eine der wichtigsten Methoden des Clients: die MouseClickAnalyse. Dabei werden sämtliche Zustände analysiert, so dass ein eindeutiger Gesamtzustand eingegrenzt werden kann.
Beispielsweise: Ich erhalte einen Rechtsklick auf ein leeres Feld der Map. Können meine Einheiten laufen, wird nun der Gehprozess eingeleitet.
Im nächsten Schritt wird dem Zustand entsprechend eine Message erstellt, die alle notwendigen Informationen enthält und via Kommunikation zum Server gesendet. Im Beispiel wie oben ist dies eine Walkmessage, die die IDs der aktivierten Einheiten und den Zielpunkt enthält.
Nach kurzer Verarbeitung durch den Server erhält nun der Client die Veränderung über die getMessages-Methode. Dabei werden ständig eingehende Messages empfangen und dem Typ entsprechend verarbeitet (messageAnalyse()).
Im Beispiel schickt der Server eine Walkmessage an die Clients zurück, die jeweils auf ihrer Map die zu bewegendende Einheit umsetzt.

Neben manuellen Eingriffen über die GUI, existieren auch Automatismen wie AutoAttack und Breitensuche. Auch die Verwaltung des Sichtbarkeitsbereichs ist Aufgabe der Client-Engine.

Autoattack

Damit der Spieler nicht alle seine Einheiten ständig unter Beobachtung halten muss, kann er durch Betätigen des Schild-Buttons je Einheit einen Autoattack-Modus aktivieren. Dabei wird kontrolliert, ob bei Umsetzen einer Einheit auf der Map, gegnerische Einheiten in den Sichtbarkeitsbereich derjenigen Einheiten gelangen, die sich im Autoattack-Modus befinden. Der Tower befindet sich dauerhaft im Autoattack-Modus. Mit Hilfe der Autoattack-Methode wird auch für eine Kampfeinheit ein neuer Gegner in Sichtweite gesucht, falls der ursprüngliche Gegner getötet wird.

Zu Breitensuche (SearchForNewRessources)

Auch bei dieser Methode wird dem menschlichen Spieler Arbeit abgenommen. Baut ein Workman eine Ressource komplett ab, sucht sich dieser automatisch mittels Breitensuche das nächste Feld mit dem vorher abgebauten Ressourcentyp, begibt sich dort hin und baut weiter ab. Findet er im Umkreis von 9 Feldern keine neue Ressource, bleibt der Workman stehen und wartet auf neue Kommandos vom Spieler.

Fog of War

Um den Sichtbarkeitsbereich zu verwalten, verwenden wir zwei zweidimensionale Arrays, die „über der Map liegen“. Das erste Array viewed beinhaltet booleans, die angeben, ob jemals eine Einheit des Spielers das Feld erkundet hat. Dafür wird bei jedem Umsetzen einer eigenen Einheit, der Sichtbarkeitsbereich der Einheit auf „true“ gesetzt, er ist somit erkundet und wird angezeigt.

Im zweiten Array view werden Integer-Werte gespeichert, die pro Feld angeben, wie viele eigene Einheiten aktuell dieses Feld in ihrem Sichtbarkeitsbereich haben. Bewegt sich eine Einheit, werden die Einträge der vorher sichtbaren Felder um 1 verringert und die neu sichtbaren um 1 erhöht.
So kann die GUI beim Zeichnen abfragen, ob mindestens eine Einheit diesen Kartenbereich aktuell sieht und ob jemals eine Einheit diesen Bereich erkundet hatte. Dementsprechend zeichnet diese das Feld komplett, verschleiert (Fog of War) oder schwarz.

Zusammenfassend kann man sagen, dass der Client die analysierende Schnittstelle graphischer Oberfläche und der Schaltzentrale (Server) darstellt.

Die grafische Benutzeroberfläche – GUI (Martin Kühn)

Die Hauptklasse: GUIWindow

Die grafische Benutzeroberfläche verwendet allumfassend die Klasse GUIWindow, die ein JFrame erweitert und ein JPanel beinhaltet, welches den aktuellen Schirm darstellt. In der Initialisierung werden Listener implementiert, Größe und Stil des Hauptfensters festgelegt, sowie ein Bild zum Zeichnen im Hintergrund angelegt. Die Hauptzeichenmethode paint() wird definiert und verwendet zum Zeichen den angelegten Zeichenpuffer.
Die Klasse besitzt einige Methoden, die vom Client aufgerufen werden und je nach aktuellem Schirm auf die enthaltenen Bildschirmklassen weitergeleitet werden.
Die Methode createIngameScreen() initialisiert hierbei vor Aufruf des Spielschirms alle Elemente desselbigen, wie die Minimap, Toolbar oder Anzeigekarte. Per changeScreen() Methode wird nach Überprüfung vom Client jeweils der Schirm gewechselt.
Die MouseListener Methoden leiten Klicks auf den Ingame Komponenten per Neuberechnung der Klickkoordinaten an die entsprechenden Komponenten weiter.

Die Bildschirmklassen

Die bereits erwähnten Bildschirmklassen sind die Klassen PaintLogin, PaintLobby, PaintCreate, PaintIngame und PaintStats, die JPanel erweitern, und die Hintergründe der jeweiligen Schirme zeichnen, sowie deren Komponenten wie Eingabefenster und Auswahlmenüs beinhalten. Da diese Klassen völlig analog arbeiten, erfolgt nur die Bertrachtung der Klasse PaintLogin.
Die Bildschirmklassen erhalten in der Regel Instanzen des ImageLoaders, der im Package Shared liegt und lediglich die Bilder läd, sowie Instanzen des GUI Hauptfensters GUIWindow und des ClientGame, der den zugehörigen Client repräsentiert. Für die Buttons, die den Bildschirmen hinzugefügt werden, werden die Buttons der ImageButton Klasse verwendet.
Die Abfragen und Setzmethoden für die Wahrheitsvariablen error oder wait lassen den Client Fehler- oder Wartezustände abfragen oder anzeigen.
Je nach Klick auf einen Button, der per buttonClicked() abgefangen wird, werden diverse Client Methoden aufgerufen, die überprüfen ob die Aktion des Buttons durchgeführt werden kann und daraufhin entsprechendes zurückliefern.

Die Komponentenklassen

Zu den sogenannten Komponentenklassen gehören alle Klassen, die bestimmte Komponenten des Ingame Schirms zeichnen, wie ShowMinimap, ShowMap, ShowRessources und Toolbar.

Die Klasse ShowMinimap

Die Klasse ShowMinimap zeichnet eine verkleinerte Ansicht eines maximal 175x175 großen Teilausschnitts der ganzen Karte. So wird zuerst berechnet, ob die Karte diese Größe überhaupt besitzt und falls nicht, ob diese sogar vergrößert (Faktor: 2, 3, ..) angezeigt werden kann. Hierbei werden grundsätzlich einige Farben definiert und je nach Rohstoff, Spielelement (Gebäude, Einheit), Bodentyp (in dieser Reihenfolge) alle Felder des Teilausschnitts abgefragt und in 1 Pixel Größe (ggf. um Faktor multipliziert) angezeigt. Hier wird ebenfalls das Modell des Zeichenpuffers verwendet, sodass der große Kartenteilausschnitt nur 2x pro Sekunde abgefragt wird und auf ein Pufferbild gezeichnet wird, welches dann wiederum bei jedem Zeichenvorgang gezeichnet wird, sodass es nicht zum Blinken der Minikarte der kommt.
Je nach Klick auf die Karte wird das gelb angezeigte Rechteck verschoben, welches den Ausschnitt des aktuell angezeigten Ausschnitts charakterisiert und per setPosX0() bzw.

…Y0() wird die aktuelle Anzeigeposition der großen Anzeigekarte versetzt. Die Abfragen verhindern hier ein mögliches ‚über die Karte hinaus'-Anzeigen. Ähnlich funktioniert die Scrollmethode, welche lediglich die Anzeige der Minikarte verschiebt.

Die Klasse ShowMap

Die Klasse ShowMap bildet den eigentlichen Bestandteil der Spielanzeige, da Sie das Spielgeschehen abbildet. Hierbei wird die Startanzeigeposition des Clients, das Auswahlrechteck, die Nachrichtenliste, die angezeigt wird, die TechTree Anzeige, die Berechnungs-Hashmaps für flüssige Bewegungen, sowie die Grafiken für die Tag- und Nachtanzeige initialisiert.
Die Methode addMsg(), die auch in der Lobby verwendet wird, lässt den Client Nachrichten auf der Karte anzeigen.
Über die Methode paintContinous() werden zusammenhängende Bodentypen wie Meer, Fluss oder Sand gezeichnet. Hier wird durch Eingabe der zusammenhängenden Typen jeweils abgefragt ob die 4 benachbarten Felder auch Felder desselben Bodentyps sind und je nach Verzweigung wird die entsprechende Grafik angezeigt. Auf den Boden wird über die Methode paintRessources() eine, von bis zu 5, zufällig ausgewählte Grafik für die Ressource angezeigt.
Die Methode setMovementImage() zeichnet je nach Art der Beschäftigung einer Einheit den Bewegungsablauf und ruft im Status ‚Walk' die Methoden setWalkingInitial(), die Parameter für die flüssige Bewegung berechnet und die Methode setWalking auf. Diese wird solange aufgerufen wird, wie sich das aktuelle Feld, auf dem die Einheit sich bewegt, nicht ändert.
Fernwaffen werden über die Methoden paintArcherAttack() bzw. …TowerAttack() realisiert.
Die Methoden checkBuildingPaint(), paintBuildingToBuild() und paintBuilding() zeichnen fertig gebaute Gebäude oder überprüfen ob Bauen an gewünschter Position möglich ist.
setDay() und paintDayNNight() realisieren den Wechsel von Tag und Nacht und dessen Anzeige.
Die Zeichenmethode paint() fragt jeweils ein 20x20 Teilarray der Karte ab und zeichnet in der Reihenfolge: Bodentyp (nur bei Bewegung Neuzeichnung), Ressourcen, Gebäude und Einheiten, sowie Nachrichten oder Nachtzustand und ruft dabei die oben erwähnten Methoden auf.
Mouse- und MouseMotionListener ermöglichen Scrollen analog zur Minikarte.

Die Klassen Toolbar und ShowRessources

Die Toolbar Klasse zeichnet alle Eigenschaften der zurzeit ausgewählten Einheiten, Gebäuden oder Ressourcen. Unter der Toolbar werden über die Klasse ShowRessources die aktuell verfügbaren Ressourcen des Spielers angezeigt.
In der Toolbar wird unterschieden ob nichts, d.h. keine Aktion, Einheiten, Gebäude oder Rohstoffe angezeigt werden. Bei Einheiten erfolgt eine Betrachtung der Menge der aktiven Einheiten. Wird eine Einheit ausgewählt, so werden dessen komplette Eigenschaften angezeigt. Werden mehrere Einheiten ausgewählt, so ist der Button für Verteidigen verfügbar, sowie die Anzeige aller ausgewählter Einheiten.

Weitere Klassen und Interfaces

ImageButton, GUIButtonAnaylze, GUIMessage, ImageLoader

Ein ImageButton erweitert den AWT Button, erhält zwei Bilder für die Zustände ‚gedrückt' und ‚nicht gedrückt', die Startposition und das JPanel, dem er hinzugefügt wird. Er verfügt

über eine Methode, die überprüft, ob ein Klick auf ihn erfolgte, sowie welches Bild je nach Zustand angezeigt wird.
Das Interface GUIButtonAnalyze wird von den ImageButton implementiert und bringt die Methode buttonClicked() ein.
Die GUIMessage beinhaltet die Datenstruktur einer Nachricht mit Sendername, Nachricht, Nachrichtyp und Sendezeit, sowie deren Getter und Setter.
Der ImageLoader des Packages Shared läd alle Bilder, die im Spiel verwendet werden in BufferedImages, die direkt abgerufen werden können um bei der Masse der Bilder die vielen Getter zu vermeiden.