Dokumentation des Projekts

im Studiengang

AIB

DiceWars

Referent : Prof. Dr. Gerd Unruh

Vorgelegt am : 09.06.2015

Vorgelegt von : E.Albach, M.Haas, J.Preuß, P.Polster
eduard.albach@hs-furtwangen.de
marco.haas@hs-furtwangen.de
jan-henrik.preuss@hs-furtwangen.de
phillip.polster@hs-furtwangen.de

Inhaltsverzeichnis

1 Einleitung

Für die Durchführung des Projekts haben wir uns entschlossen ein Spiel zu programmieren. Bei dem Spiel handelt es sich um das Spiel DiceWars. Dieses Spiel wurde also nicht neu erfunden sondern einem Bestehenden nachempfunden.
Das Spiel hat sehr viel Ähnlichkeit mit dem Brettspiel Risiko. Dort müssen taktisch Einheiten auf Feldern verteilt werden um so seine Feinde zu besiegen. So auch in unserem DiceWars. Den eigentlichen Spielablauf beschreiben wir in Kapitel 2 "Spielregeln".
Wichtiger Punkt bei der Implementierung war es verschiedene Technologien einzubinden. So greifen wir mit Hilfe der Random.org API auf einen Zufallsgenerator zu, der anhand des Weltraumrauschens Zufallszahlen berechnet.
Als weitere Technologie verwenden wir eine lokale Datenbank. In dieser Datenbank können die Spieler sich verewigen. Hier werden verschiedene Parameter abgespeichert und diese werden am Ende des Spiel grafisch angezeigt.
Damit alle Funktionen sinnvoll eingesetzt werden können ist die Verwendung verschiedener Design-Pattern Pflicht. Hier haben wir versucht an möglichst sinnvollen Stellen Design-Pattern für bekannte Probleme einzusetzen. Dazu jedoch mehr in dem Kapitel "Design-Pattern".

2 Spielregeln

Die Spielregeln des Spiels sind relativ simpel. Im Hauptmenü wählt man aus wie viele Spieler an einem Spiel teilnehmen wollen. Zusätzlich wählt man noch die Größe des Spielfeldes aus.

Jeder Spieler hat seine eigene Farbe. Diese Farbe behält der jeweilige Spieler für das ganze Spiel. Die Felder werden zufällig angeordnet, sodass bei jedem neuen Spiel neue Spielsituationen entstehen können.
Auf diese Felder werden Würfel verteilt. Diese stehen für die Größe der Streitmächte. Wenn also auf einem roten Feld die Zahl 5 steht heißt das, dass der Spieler mit der Farbe rot auf diesem Feld 5 Würfel hat. Also kann er mit diesen 5 Würfeln alle direkt anliegenden feindlichen Felder angreifen.

Bei einem Angriff auf ein feindliches Feld wird nun gewürfelt. Wir nehmen an der Feind hat ebenfalls 5 Würfel auf seinem Feld und wir greifen mit 5 Würfeln an. Jetzt wird angegriffen (gewürfelt).
Wir haben nun eine 4, eine 1, eine 3, eine 6, und eine 3 gewürfelt. Somit ergibt

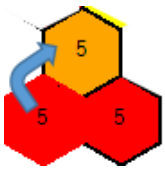

Abbildung 1: Spieler-Rot greift Spieler-Orange an

sich eine Summe von 17. Diese Summe ist ausschlaggebend. Wenn der Gegner mit seinen 5 Würfeln unsere Summe schlagen kann, haben wir verloren und unser Angriff war erfolglos. In diesem Fall kann unser Gegner und wir höchstens 6 + 6 + 6 + 6 + 6 Würfeln. Also eine 30. Unser Gegner würfelt jedoch nur eine 10. Somit haben wir eine höhere Zahl und dieses Feld gehört uns. Wenn wir das Feld eingenommen haben, bleibt ein Würfel auf dem Feld, von dem wir angegriffen haben. Somit bleibt dieses Feld in unserem Besitz. Da wir das feindliche Feld übernommen haben, wandern die anderen 4 Würfel auf das besiegte Feld. Natürlich wechselt dieses Feld auch die Farbe.

Wenn ein Angriff erfolglos war und wir eine kleinere Summe haben als der Geg-

Abbildung 2: Spieler-Rot übernimmt Spieler-Orange Feld

ner, dann verlieren wir alle Würfel bis auf einen. Dieser eine Würfel bleibt dann auf unserem Feld zurück.
In so einem Fall hat der Gegner leichtes Spiel und kann sobald dieser an der Reihe

Abbildung 3: Fehlgeschlagener Angriff

ist einen Konter starten und dieses "geschwächte" Feld angreifen.

Ein Feld auf dem nur ein Würfel liegt, kann nicht angreifen. Hier macht es keinen Sinn, da wir das eigene Feld verlassen müssen. Sprich nur von Feldern mit der Würfelanzahl > 1 kann angegriffen werden.

Ein Spieler muss aber nicht zwangsweiße angreifen. Er kann auch theoretisch vorsichtig spielen und eine Verteidigungslinie aufbauen. Da man wie schon erwähnt nur direkt anliegende Felder angreifen kann, kann man auch Würfel an direkt anliegende Felder verschieben. So kommt auch etwas Taktik in das Spiel. Auf diese Art lassen sich Verteidigungs- bzw. Angriffslinien aufbauen. Beim Verschieben werden zwei drittel der Würfel auf das gewünschte Feld verschoben. (Natürlich nur benachbarte Felder)
Wenn ein Spieler alle Entscheidungen getroffen hat, kann er seine Runde abschließen und der nächste Spieler ist an der Reihe.
So hat Spieler für Spieler die Chance seine Einheiten zu verschieben bzw. gegnerische Felder anzugreifen.
Nachdem alle Spieler am Zug waren ist eine Runde beendet. Nach jedem Zug bekommt der jeweilige Spieler neue Würfel auf seine Felder verteilt. So bekommt jeder Spieler "Nachschub" für seine Felder und kann seine Einheiten wieder neu verteilen bzw. andere Felder angreifen.

Ziel ist es die gesamte Spielfläche mit seinen Einheiten einzunehmen.
Am leichtesten lernt man die Spielabläufe jedoch beim Spielen.

3 Verwendete Technologien

Random.org API

Um beim Würfeln richtige und keine Pseudo-Zufallszahlen zu bekommen, greifen wir auf die Random.org API zu. Diese liefert Zufallszahlen in Abhängigkeit vom Weltraumrauschen und wird deshalb von Lotterien, Casinos und Online-Spielen benutzt, da es sich um wahre Zufallszahlen handelt und nicht um Pseudo-Zufallszahlen, die ein Computer generiert.
Die Implementierung findet sich in der Klasse RandomGenerator. Diese Klasse enthält die öffentliche Methode "public int rollTheDices(int dices)". Diese Methode ruft random.org auf mit passenden Übergabeparametern (kleinst mögliche Zahl, größt mögliche Zahl, Anzahl Zahlen) auf. Das Ergebnis (bzw. der HTML Code der Ergebnis-Seite) wird mit der Methode "DownloadData" heruntergeladen. Dieser muss dann erst noch geparst werden, also die generierten Zahlen aus dem HTML-Code herausgefilert und in einem Integer-Array gespeichert werden. Da es bei diesem Spiel nur um die Summe aller Augenwürfel geht, werden alle Zahlen des Integer-Arrays anschließend addiert und die Summe als Rückgabewert zurückgegeben.
Sollte dabei ein Fehler auftreten (z.B. weil der Rechner gerade keine Internetverbindung hat), wird die rekursive Methode "rollTheDiceOffline" aufgerufen, die Pseudo-Zufallszahlen generiert und die Summe davon zurückgibt. Somit ist man nicht auf eine Internetverbindung beim Spielen angewiesen.

SQlite

Des weiteren haben wir eine lokale SQLite Datenbank erstellt. Diese soll die Statistik abspeichern. Diese umfasst den Spieler-Namen, die benötigte Spielzeit, die Gegneranzahl und die Spielfeldgröße. Zur Installation bzw. Nutzung von SQLite gehen wir hier nicht weiter ein. Im Netz finden sich diverse Anleitungen für alle verfügbaren Visual Studio Versionen.
Auch viele Dokumentationen mit anschaulichen Beispielen lassen sich leicht finden.

4 Verwendete Design-Pattern

RandomGenerator (Singleton)

Den Zufallsgenerator, der wahre Zufallszahlen mit Hilfe von der Random.org API generiert, haben wir als Singleton implementiert. Von dem RandomGenerator soll es nur eine Instanz geben. Man ruft von außen nur die Methode "rollTheDice" auf (diese generiert dann, je nachdem ob man eine Internetverbindung hat oder nicht, richtige oder Pseudo-Zufallszahlen und gibt deren Summe zurück). Mehrere Instanzen eines RandomGenerators anzulegen wäre daher nicht sinnvoll, da die Methode "rollTheDice" nie mehrmals parallel aufgerufen werden muss.

Builder-Pattern

Das Builder-Pattern wird bei uns verwendet um das Spielfeld zu erzeugen. Wir verwenden zweimal das Pattern. Einmal wie erwähnt für das "Board" und einmal für das "BoardState". Das Builder-Pattern ist dafür da, um die Erstellung komplexer Objekte zu vereinfachen.

DTO-Pattern

Das DTO-Pattern verwenden wir um Daten zu sammeln und diese dann in die Datenbank zu speichern. So sammeln wir an verschiedenen Stellen im Code Daten und in der letzten Form speichern wir diese in die Datenbank.
Das DTO-Pattern ist dafür da, primitive Datentypen abzuspeichern bzw. einzusammeln.

Observer-Pattern

Da wir auf dem Spielfeld die Würfelanzal anzeigen und diese sich ständig ändert, haben wir hier das Observer-Pattern implementiert. Nachdem jeder Spieler seine Züge gemacht hat, werden Würfel neu vergeben bzw. auf die Vorhandenen addiert. Jetzt muss die GUI und alle dazugehörigen Objekte aktualisiert werden.
Durch die Observer-Funktionalität lassen sich solche Probleme leicht lösen.

Somit haben wir vier Design-Pattern implementiert. (Gefordert waren hier drei Pattern)