



Monster Trading Card Game

MTCG

FLORIAN WEISKIRCHNER – IF19B132

Inhalt

Vorbereitung/Benutzung	2
Datenbank	2
Umsetzung.....	2
Datenbank	2
Credentials.....	2
Cards.....	2
Player_Cards.....	2
Player.....	2
Packages	2
Trade.....	2
Player_deck	2
Battle	2
Battle_Log.....	3
Server	3
Player.....	3
Cards.....	3
Enum.....	3
Battlelogic.....	4
Klassendiagram.....	4
Tests	5
Probleme und Erkenntnisse („Lessons learned“)	5
Generelle Überlegungen	5
Server	5
Tests	5
Zeitaufwand.....	6
Insgesamter Zeitaufwand.....	6
GitHub	6

Vorbereitung/Benutzung

Datenbank

Bevor das Programm/ das MTCG benutzt werden kann muss die Datenbank vorbereitet werden. Dazu wurde die PostgreSQLPortable-10.1.1 verwendet mit der DATABASE mtcg. Um sie vorzubereiten kann die Datei „DBprepareAll.sql“ ausgeführt werden.

Umsetzung

Datenbank

Wie schon erwähnt läuft die Datenbank auf einer PostgreSQLPortable-10.1.1 Datenbank.

Credentials

Speichert die wichtigsten Userinformationen, wie Username, Password und Token. Wird bei der Registrierung befüllt und beim Login sowie der Token Überprüfung werden benötigte Informationen aus dieser Tabelle ausgelesen.

Cards

Speichert alle wichtigen Informationen einer Karte, wie ID, Name, Typ, Element, Damage. Wird bei BuyPackage verwendet.

Player_Cards

Wird gleichzeitig mit Cards bei BuyPackage befüllt.

In dieser Tabelle ist die ID einer Karte sowie die der Username eines Users abgespeichert, somit wird gespeichert welcher User welche Karte besitzt.

Player

Speichert die Userinformationen eines Players ab, wie Username, Name, Bio, Image, Elo. Wird gleichzeitig mit Credentials erzeugt und befüllt.

Packages

Diese Tabelle speichert den Json Text ab der bei CreatePackage übergeben wurde.

Trade

Wird nicht verwendet!

Speichert die Informationen eines Trades ab, wie Username, Card_Id, Required_Type, Required_Element, Required_Damage.

Player_deck

Beinhaltet alle wichtigen Informationen, Karten IDs von 4 Karten sowie den Usernamen eines Players.

Battle

Stellt bestehende Battle_Rooms dar, welche Informationen wie, ID, Player1 und Player2 beinhalten. Player2 kann NULL sein wodurch ein Battle_Room als verfügbar angezeigt

wird. Wenn sowohl Player1 als auch Player2 belegt ist, zeigt es das der Kampf schon durchgeführt wurde.

Battle_Log

Beinhaltet die Informationen eines bereits vergangenen Kampfes. Wird von einem ausgeführten Battle erstellt und beinhaltet Informationen wie der Kampftext, Gewinner, und eine ID, die auf ein Battle verweist in der Battle Tabelle.

Server

Der RestServer erstellt einen TCPListener und nimmt danach bis zu 2 Clients an und erstellt für diese einen Thread. Weiters ruft jeder dieser Threads ein Messgelnput Object, dieses Object nimmt die http message entgegen und sichtet diese weiter an das RequestContext Object. In diesem Object liest alle wichtigen Informationen aus dem http Header aus wie etwa das http Verb(Get, Post, Put, Delete) oder den Authorization Token sowie den Content. Diese Informationen werden von Messgelnput an das ReguestHandler Object weitergegeben. In diesem Object werden alle gesammelten Informationen verarbeitet und alle DB Zugriffe getätigt. Nachdem alle Daten verarbeitet wurden wird ein http Response zurück geschickt an den Client.

Player

In der Player Klasse werden die JsonPropertyts abgespeichert sowie das derzeitige Deck eines Players.

Cards

Meine Logic hinter den Karten besteht aus einer abstrakten Klasse „BaseCard“ von der die Klassen Monstercard und Spellcard erben. Weiters gibt es die Klasse JsonCard in der Json Anfragen verarbeitet wird und mittels RegularExpressions aus dem Namen ausgelesen wird um welche Category von Karte es sich handelt sowie welches Element und im Falle eines Monsters um welche Rasse/Klasse es sich bei diesem Monster handelt. Aufgrund dieser Informationen wird das benötigte Object erstellt.

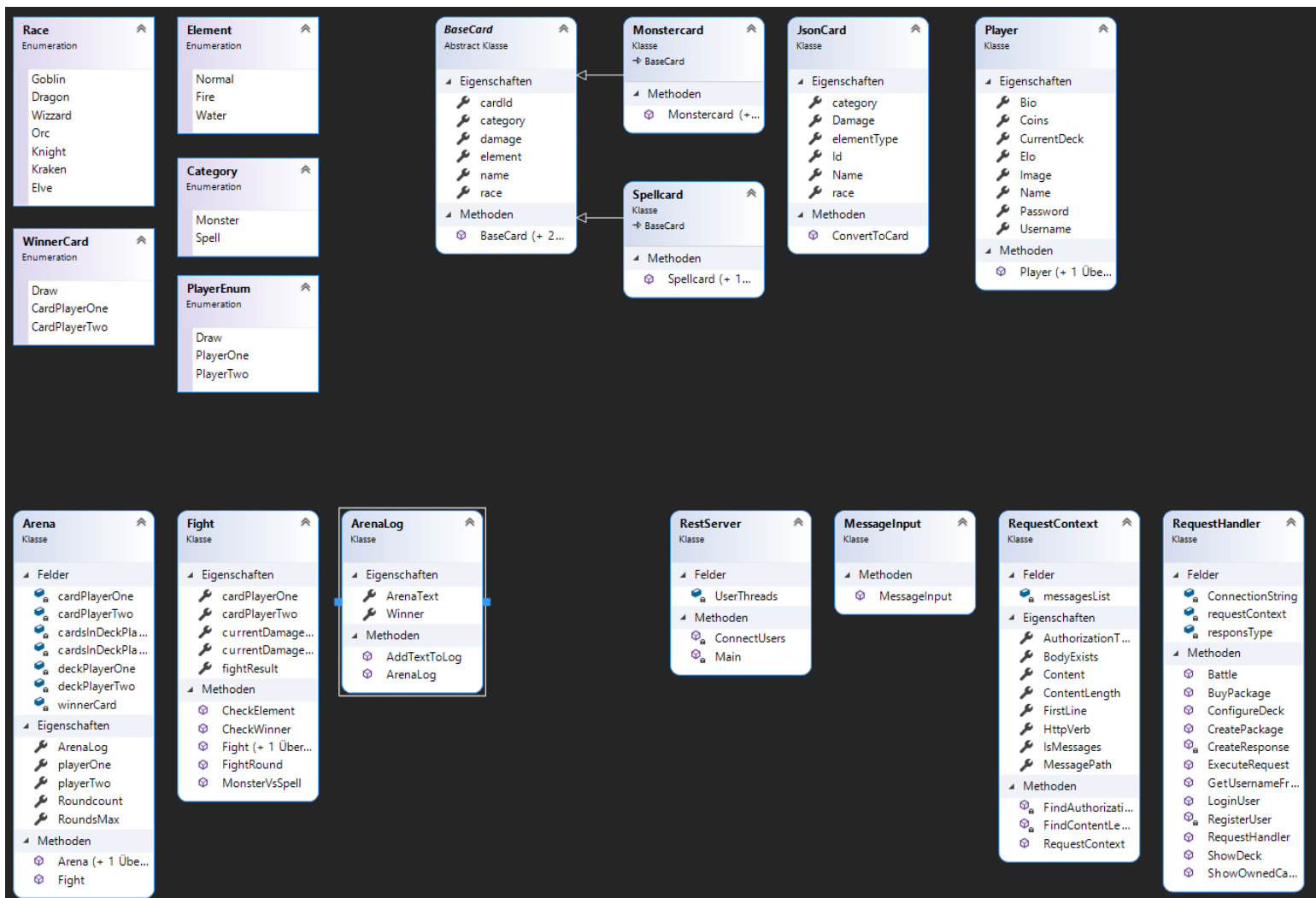
Enum

Im Laufe des Projektes habe ich 5 Enums erstellt, die die Lesbarkeit des Codes erleichtern sollen. Informationen wie die Category, Element oder Rasse einer Karte werden als Enum abgespeichert.

Battlelogic

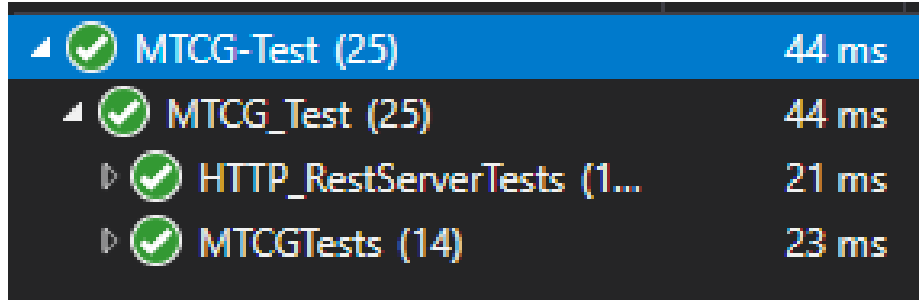
Meine Battlelogic besteht aus einer Arena Klasse, einer Fight Klasse und einer ArenaLog Klasse. Zu Beginn eines Kampfes wird ein Objekt der Arena Klasse aufgerufen, welches ein Objekt von Fight und ArenaLog erstellt. In Arena wird ein kompletter Kampf überwacht und solange durchgeführt bis 100 Runden erreicht sind oder ein Player keine Karten mehr in seinem Deck hat. Jede Runde wird das Objekt Fight aufgerufen, in dem die Logic einer Runde durchgeführt wird. Am Ende einer Runde werden die Ereignisse dieser im ArenaLog abgespeichert. Wenn ein Kampf vorbei ist wird der ArenaLog an den Client zurückgegeben.

Klassendiagramm



Tests

Ich habe insgesamt 25 Tests, die meine Funktionalität meines MTCGs testen.



▲ ✓ MTCG-Test (25)	44 ms
▲ ✓ MTCG_Test (25)	44 ms
▶ ✓ HTTP_RestServerTests (1...	21 ms
▶ ✓ MTCGTests (14)	23 ms

Ich habe meine Tests in zwei Teile aufgeteilt, ein Teil testen den Rest Server und der zweite Teil die Battlelogic. Bei dem Test vom Rest Server handelt es sich um Test die ich schon beim Erstellen des Rest Servers hatte die leicht abgewandelt wurden bzw einige wurden raus genommen da ich die Funktionalität etwas geändert habe.

Der zweite Teil sind Tests zur Battlelogic und Karten ob diese richtig erstellt werden und die richtigen Werte haben und die Logic in einem Kampf wird getestet.

Probleme und Erkenntnisse („Lessons learned“)

Generelle Überlegungen

Ich hatte das Problem am Anfang des Projektes wie dieses ausschauen wird am Ende, wenn es fertig ist, somit viel es mir schwer zu beginnen. Beginnen habe ich dann mit der Battlelogic da ich diese an Einfachsten zu beginnen gefunden habe.

Server

Die meisten Probleme hatte ich damit meine Überlegungen zu dem Projekt mit dem Rest Server zu verbinden. Da ich die Battlelogic schon fertig hatte bevor ich meine Arbeit mit dem Server verbunden habe hatte ich schon eine Überlegung wie der Server darauf reagieren muss.

Tests

Testdriven Development war ein neues Thema für mich und eine neue Art ein Projekt zu starten bzw durchzuführen. Leider hatte ich meine Probleme mit diesem Ansatz da ich von den letzten beiden Semestern eine andere Herangehensweise gewohnt war. Mocking habe ich leider garnicht verwendet da ich nicht genau weiß wie ich dieses richtig verwenden kann.

Was mir sehr geholfen hat war ein Klassendiagramm ganz am Anfang zu erstellen, da ich mir schon sehr früh Gedanken über vieles machen musste was ich umsetzen möchte.

Auch wenn das Diagramm sich über die Zeit sehr verändert hat war es trotzdem Hilfreich zu erkennen was gemacht werden muss.

Zeitaufwand

Leider habe ich keine genaue Zeitmessung von der Arbeit an dem ersten Projekt, dem Rest Server dadurch ist der insgesamte Zeitaufwand nicht ganz genau, aber ich schätze es waren zwischen 7-8 Stunden, die ich dafür aufgewendet habe.

Insgesamter Zeitaufwand

Ich habe 64 Stunden an dem Projekt gearbeitet.

GitHub

Link zu meinem Repository auf GitHub:

<https://github.com/FlewXD/MTCG>