# MTCG – Protocol

Repository Link: https://github.com/unravelet/MonsterCardTradingGame

## Design

### Battle

Als erstes wurde die Battle Logik programmiert, diese braucht 2 User mit vollständigen Decks aus 4 Karten. Da Karten einen Speedwert haben, wird bei jeder Runde überprüft welche Karte den höheren Speedwert hat, diese Karte greift dann zuerst an.

Bevor ich den Server implementiert hatte, war meine Idee, dass der User vor dem Battle sich 4 Karten aus seinem Stack aussuchen kann, doch später habe ich eine Datenbank Tabelle „decks“ erstellt, in der die ausgewählten Karten aus dem User-Stack von jedem User gespeichert werden, sollte man ein neues Deck erstellen wollen, wird das alte gelöscht und das neue einegfügt.

Zuerst war die Idee den User, der nur noch 2 Karten hat, zu fragen, ob ein Booster benutzt werden soll, das wurde aber geändert, nachdem ich die http-Requests implementiert habe. Jetzt wird der Booster automatisch bei der letzten Karte benutzt.

Wenn ein Battle startet, wird ein Timestamp in einem String gespeichert und als Teil des Filenamen für den Battlelog verwendet. Die 2. Hälfte des Namens sind die Usernames der Player.

Um dem Client das Ergebnis zu schicken, wird ein String mit dem Sieger oder einem Unentschieden als Rückgabewert der BattleStart() Funktionen genommen. Somit wird je nach Ergebnis der String gesetzt und kann an den Client geschickt werden

### Scoreboard

Nachdem im Battle ein Gewinner feststeht, wird das Scoreboard in der Datenbank upgedated, sodass immer der aktuelle Score zu sehen ist. Der Score wird berechnet, indem man 5 Punkte fürs Gewinnen dazubekommt, und 3 Punkte fürs Verlieren verliert. Die Win-Lose_ratio wird durch Wins/Losses berechnet.

### Package

Die Package Klasse generiert 5 zufällige Karten, die entweder eine Monsterkarte oder eine Spellkarte ist mit zufälligen Damage-, und Speedwerten. Weiters hat jedes Element und jedes Monster eine eigene Description, die etwas über die Spielewelt erzählt und gleichzeitig die Stärken und Schwächen der Karte andeutet.

## Cards

Die zufällig generierten Karten können einen von 7 Monstertypen haben oder ein Spell sein. Außerdem erhalten sie einen von 7 Elementtypen, die verschiedene Effektivität haben.

### Elementtypen

Jedes Element hat dieselbe Wahrscheinlichkeit. Die Effektivität der Elemente lässt sich so darstellen:

([Element] ist effektiv gegen -> **[Element]** ist effektiv gegen -> [Element])

Normal, Ice -> **Water** -> Fire, Ground

Ground, Water -> **Fire** -> Normal, Ice

Fire, Dark -> **Normal**-> Water, Light

Ice, Water -> **Ground** -> Fire, Light

Light, Fire -> **Ice** -> Water, Ground

Normal, Ground -> **Light** -> Dark, Ice

Dark, Light -> **Dark** -> Normal, Dark

## User

User erhalten beim Erstellen 20 Coins, die in der Datenbank gespeichert werden. Sollte ein User ein Package kaufen, wird zuerst in der Datenbank überprüft, wie viele Coins vorhanden sind. Nachdem Kauf werden die Coins in der Datenbank wieder aktualisiert.

# Lessons learned

### 1. Schreib alles klein in einer Datenbank

Beim Erstellen meiner Datenbank benannte ich meine Tables „Users“ und „Cards“ und beim Einfügen in die DB haben die SQL Statements fehlgeschlagen, weil es anscheinend häufig Probleme gibt, wenn man die Namen nicht richtig im Statement angibt, also wurde aus „Users“ „users“ und aus „Cards“ „cards“ und somit ist man auf der sicheren Seite

### 2. Zuerst Datenbank Models, dann Game Logic

Ich habe als aller erstes nur an die Battle Logic gedacht als ich zum Programmieren anfing. Nachdem diese fertig war, wollte ich die Datenbank erstellen und diese mit Usern und Cards füllen, doch musste dann einiges in meinem Programm umschreiben, da die Models nicht für Datenbankeinträge geeignet waren. So musste ich z.B. meine Child-Classes SpellCard und MonsterCard löschen und alles in der Card Class erstellen und zusätzlich auch einiges in meiner Package Class umschreiben.

### 3. Während des Programmierens an Unit Tests denken

Die Unit Tests wollte ich ganz zum Schluss machen, musste aber feststellen, dass mein programm dann Unit Test-ungeeignet war und konnte dann nicht sehr sinnvolle Unit Tests schreiben.

### 4. Mehr Klassen und striktere Trennungen

Nach einigen Stunden musste ich feststellen, dass mein Programm nicht so sauber ist, wie ich es gerne hätte, und würde beim nächsten mal mehr Funktionalitäten trennen und mehr Klassen machen, die kleinere und mehr Funktionen haben, um bessere Übersicht zu haben.

## Unit Test Decisions

Die gewählten Unit Tests testen z.B. ob es möglich ist negative Zahlen zu Scores oder Coins hinzuzufügen und ob die einzelnen Funktionen den gewünschten Wert abziehen oder hinzufügen.

Mein Programm ist leider nicht auf Unit Tests ausgelegt, da ich während des Programmierens nicht daran gedacht habe, also habe ich auch im Nachhinein Funktionen verändern und hinzufügen müssen. Jedoch haben sie mir sehr geholfen ein paar Bugs und Sicherheitslücken zu finden.

## Unique Features

### Booster

Wenn ein Player in einem Battle nur noch eine Karten hat, kann er einmalig im Battle einen Booster einsetzen. Dieser Booster erhöht den Damage der Karte, die gerade im Battle ist, um entweder 20%, 30%, 40% oder 50%. Um wie viel der Damage erhöht wird ist zufällig.

### Speed

Jede Karte hat zusätzlich zu einem Damagewert auch einen Speedwert (1-10). Dieser wird zufällig beim Erstellen der Karte generiert, jedoch haben Karten mit niedrigerem Damage eine höhere Chance einen Speedwert über 4 zu haben. Z.B.: Karte mit 10 Damage hat eine 80% Chance einen Speedwert zwischen 5 - 10, und eine 20% Chance einen Speedwert zwischen 1 – 4 zu erhalten, welcher es wird ist auch zufällig. Jedoch hat eine Karte mit 90 Damage eine Wahrscheinlichkeit von 2% einen Speedwert zwischen 5 – 10 und eine 98%ige Wahrscheinlichkeit einen Speedwert zwischen 1 – 4 zu erhalten.

In einem Battle werden dann die Karten zufällig aus dem Deck gezogen und die Karte mit dem höheren Speedwert greift zuerst an. Sollten beide Karten in einer runde denselben Speedwert haben, wird gewürfelt welche Karte zuerst angreift.

# Tracked Time

| | Rebekka Tscheppen | | | | |
|---|---|---|---|---|---|
| | | *sum - hours of work:* | | 58:20:00 | |
| # | date | start | end | duration | content of work |
| 1 | 29.09.2021 | 12:30:00 | 12:50:00 | 00:20:00 | UML Diagramm angefangen |
| 2 | 09.11.2021 | 12:00:00 | 12:50:00 | 00:50:00 | simple server/client demo |
| 3 | 29.12.2021 | 14:40:00 | 15:38:00 | 00:58:00 | Projekt erstellt, Klassen erstellt |
| 4 | 30.12.2021 | 15:20:00 | 17:22:00 | 02:02:00 | Server und Client erstellt |
| 5 | 31.12.2021 | 13:30:00 | 16:50:00 | 03:20:00 | postgresql installieren, npgsql versuchen zu installieren, client.bat datei erstellt für mehr terminals |
| 6 | 01.01.2022 | 14:00:00 | 17:10:00 | 03:10:00 | package implementiert, user cards list implementiert, card descriptions, mehr elemente hinzugefügt |
| 7 | 01.01.2022 | 20:40:00 | 21:47:00 | 01:07:00 | elemental types effektivität |
| 8 | 03.01.2022 | 12:45:00 | 19:00:00 | 06:15:00 | battle class |
| 9 | 04.01.2022 | 15:00:00 | 20:40:00 | 05:40:00 | debugging, Battle Log, card descriptions, unique feature booster |
| 10 | 05.01.2022 | 11:30:00 | 11:57:00 | 00:27:00 | speed algorithm |
| 11 | 05.01.2022 | 13:37:00 | 19:38:00 | 06:01:00 | speed in battle hinzugefügt, debugging, datenbank verbindung, data in datenbank einfügen |
| 12 | 06.01.2022 | 13:10:00 | 16:57:00 | 03:47:00 | monstercard und spellcard klasse gelöscht, card klasse umgeschrieben, daten aus db auslesen und löschen |
| 13 | 06.01.2022 | 17:40:00 | 18:20:00 | 00:40:00 | data aus db holen und startBattle() probieren, timestamp zu battlelog hinzugefügt |
| 14 | 08.01.2022 | 14:30:00 | 20:55:00 | 06:25:00 | user über http processor in db eingefügt |
| 15 | 09.01.2022 | 13:15:00 | 19:00:00 | 05:45:00 | session, package, card deck, battle controller |
| 16 | 09.01.2022 | 22:00:00 | 23:10:00 | 01:10:00 | debugging, deck controller, battle controller, protokoll |
| 17 | 10.01.2022 | 10:59:00 | 21:22:00 | 10:23:00 | unittests, scoreboard datenbank, stats controller, score controller, card descriptions, protokoll, debugging |