FH University of Applied Sciences TECHNIKUM WIEN

Protokoll MonsterTradingCardGame

FH Technikum
Bachelor of Science Informatik

Von Dorian Fetty

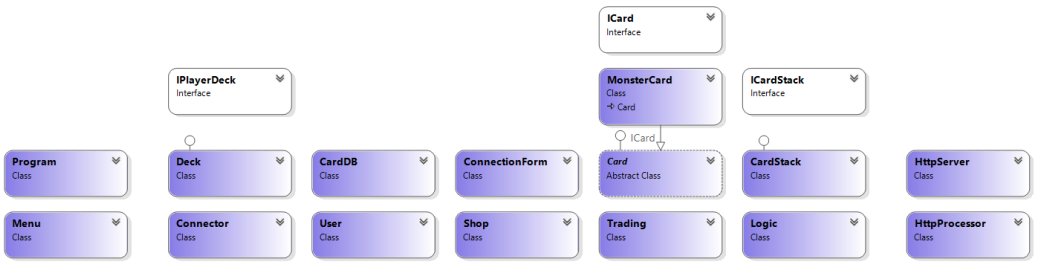
Inhaltsverzeichnis

Dorian Fetty
if20b082
Semester 3

Software Design

Alle Funktionen sind über die Klasse “Menu” für den Nutzer interaktiv ansprechbar. Menu kann zwischen Registrierten und nicht registrierten Nutzern unterscheiden. Wobei die gesamte Spielmechanik nur eingeloggten Usern zur Verfügung steht. Um ein wenig mehr sicherheit in das Konzept für die Datenverwaltung zu bringen, wird für jeden Nutzer ein zufälliger Token als ID bei der Registrierung erstellt. In der Klasse User existiert eine statische Variable die diese ID entgegennimmt und bei jeder Nutzer spezifischen Kommunikation mit der Datenbank diese Variable abfragt. Die ID ist dem Nutzer selber nicht bekannt.

UML-Diagram



Datenbankdesign

Unique Feature

Usern steht es frei sich einer Gilde anzuschließen. Zurzeit sind zwei Gilden Implementiert Goldrusher und Sand Nomade.

Goldrusher	Waren schon immer begabt darin ihren Gewinn zu maximieren. Jede gewonnene Runde ergibt für sie +1 Coin.
Sand Nomade	Sand Nomaden sind geborene Händler und bekommen einen Rabatt von 1 Coin bei käufen im Shop.

User können unter dem Menüpunkt “Profil” die Gilde wechseln.

Lessons Learned

Datenbank integrierung in ein C# Programm

Sowohl PostgreSQL als auch in Verbindung mit einem C# Programm waren eine neuheit für mich. Durch einige Projekte im Webbereich waren mir die Datenbanken von MariaDB und die SQL Sprache schon bekannt. PostgreSQL Dokumentation war eine gute Quelle für PostgreSQL spezifische Befehle. zB: “SERIAL” ist eine Nützliche erweiterung für Primary Keys unter PostgreSQL. Schlechte online Recherche bereitete mir bei der konfigurierung meines ConnectionStrings. FH-Kollegen konnten mir hier die richtige Syntax zeigen und die Verbindung zwischen Datenbank und Programm machte seither keine Probleme mehr.

online Recherche	UserID=root;Password=******;Host=localhost;Port=5432;Database=monstercardgame;Connection Lifetime=0;
FH-Kollegen	Host=localhost;Username=postgres;Password=******;Database=monstercardgame

UnitTesting

Bisherige Erfahrungen mit Unit-Testing waren auf C++ mit Catch2 begrenzt. NUnit Tests sind einfacher zu beginnen, da es anders als Catch2 schon in die IDE eingebaut war. Es mussten keine extra Header-Files heruntergeladen werden. Durch die einfache Syntax und mit der Konvention einen Test in ARRANGE, ACT und ASSERT zu unterteilen war es mir meist möglich effektiv Tests zu schreiben ohne viel Dokumentation heranzuziehen zu müssen. Lediglich die minimal Anzahl der Tests die notwendig waren stellte für mich eine gewisse herausforderung dar. Der Großteil meiner Funktionen beinhalten Datenbankverbindungen. Diese waren nicht von den Tests aufrufbar, ein Fehler den ich bislang nicht beheben konnte. Ich bin mir auch nicht sicher, ob es überhaupt

funktionieren soll.

Des weiteren waren alle Funktionen die User-Input benötigen nicht gut geeignet für Tests. Normaler string Input war mir in den Tests möglich, aber jegliche Funktion die auf funktionalität eines ConsoleKey baute konnte mit den üblichen Methoden nicht bedient werden.

Die Tests die mir Möglich waren fokussieren sich auf die Kampflogik des Spiels. Werden Kämpfe mit Spells richtig erkannt. Wird der Schaden einer Karte richtig berechnet. Funktionieren Grundelemente wie Deck mischen, Karten in sein Deck einfügen. Wird der richtige Spieler als Gewinner markiert. Bei der weiteren Implementierung von Spielelementen konnte mir so schnell gezeigt werden, dass meine Grundregeln eingehalten oder verletzt wurden und eine Notwendige Korrektur konnte so schnell durchgeführt werden.

User-Input und SQL-Injection

User Input ist immer ein Risikofaktor für ein Programm. Besonders, wenn dieser Input auch noch an eine Datenbank gesendet wird. Einige meiner Kollegen versuchen SQL-Injection vorzubeugen indem sie einen Inputhandler (Primär für die Stabilität des Programms) geschrieben haben und SQL-String handling (dynamisches einfügen von Werten in eine vorgeschriebenes SQL-Statement) betreiben. Zweiteres ist für einen besseren Schutz auch in meinem Programm eingebaut. Jedoch um direkten User-Input größtenteils zu vermeiden, sind fast alle Funktionen darauf ausgelegt ihre Inhalte dynamisch anzupassen und die User-Auswahl durch die Pfeiltasten und dem Enter-Knopf zu eruieren.

Zeitaufwand

Zeit in Minuten	Feature
112	Database 1.0
42	Database 2.0
44	SQL Injection
37	Login
42	Register
23	Shop - Buy Pack
27	Shop - Sell Card
12	Shop - Navigation
43	Shop - Design
86	Configure Deck 1.0
55	Configure Deck 2.0 (Arrow Navigation)

34	User Profile
36	Edit User Data
22	ScoreBoard - Logic
19	ScoreBoard - Design
92	Trading - Create Offer
54	Trading - Accept Offer
66	Trading Offer - Navigation
174	Battle Logic
40	Battle Logic Outcomes (Win, Lose, Draw, Elo)
37	Battle Log
33	User Token Implementation
167	Unit Test
194	HTTP Server Prototyp + Testing
69	Protokoll
142	Monster Card Class
62	User Class
117	Database Connection
42	Menu Class 1.0
107	Menu Class 2.0
32	Unique Feature
In Summe 34 Stunden	

Gitlink

https://github.com/Warrensy/MonsterCardGame.git