Protokoll zur Implementierung des Klassendiagramms

@Author - Akyol Emre

1. Zielsetzung

Ziel dieser Implementierung war die Entwicklung eines Systems für einen HTTP-Server, der Anfragen verarbeitet und Daten über HTTP-Methoden wie POST abruft. Zudem wurde die Grundlage für ein Monster Trading Card Game (MTCG) gelegt, das Elemente wie Kartentypen (Wasser, Feuer, etc.), Coins und die Kampfmechanik umfasst. Darüber hinaus wurden fortlaufend neue Funktionen wie Benutzerverwaltung, Kartensystem und erweiterte Kampfmechaniken integriert.

2. Architektur

2.1 HTTP-Server

- **HttpSvr**:
  - Implementiert die Serverfunktionalität. Die Methoden `Run` und `Stop` steuern den Start und das Stoppen des Servers.
  - Die Methode `Incoming` verarbeitet eingehende HTTP-Anfragen und leitet diese an die entsprechenden Handler weiter.
- **HttpHeader und HttpStatusCode**:
  - **HttpHeader** verwaltet die HTTP-Header, die bei der Verarbeitung der Anfragen benötigt werden.
  - **HttpStatusCode** dient zur Verwaltung und Interpretation der Statuscodes, die als Antwort an den Client gesendet werden.

2.2 Datenabfrage

- **Datenhandler**:
  - Diese Klasse verarbeitet Anfragen zur Datenabfrage. Sie empfängt POST-Anfragen, analysiert die angeforderten Daten und führt entsprechende Aktionen aus.
  - Beispielmethoden könnten `GetData` oder `ProcessData` sein, die auf eingehende Anfragen reagieren.
- **Verarbeitung der Anfragen**:
  - Die Methode `Incoming` im **HttpSvr** leitet die eingehenden HTTP-Anfragen an den **Datenhandler** weiter. Dieser sorgt dafür, dass die angeforderten Daten verarbeitet und dem Client zurückgegeben werden.

2.3 Design-Entscheidungen

- **Modularität**:

    - Das Klassendiagramm teilt die Verantwortung klar auf: Der HTTP-Server (`HttpSvr`) kümmert sich um die Kommunikation, während der **Datenhandler** für die spezifische Logik der Datenabfrage verantwortlich ist.
- **Fehlerbehandlung**:
    - Um die Stabilität des Systems zu gewährleisten, sind alle relevanten Klassen mit einer robusten Fehlerbehandlung ausgestattet. Diese stellt sicher, dass fehlerhafte Anfragen oder Serverprobleme korrekt abgewickelt werden.

2.4 Erweiterung der Kampf- und Benutzerlogik

- **Kampfmechanik**:
    - Es wurde eine erweiterte Kampf-Logik implementiert, die den Schaden basierend auf den Kartenelementen und deren Werten berechnet.
    - Ein `ElementInteractionHandler` wurde hinzugefügt, um die Interaktionen zwischen verschiedenen Kartenelementen zu verwalten. Hierbei beeinflussen die Elemente wie `Water` und `Fire` den Ausgang eines Kampfes.
- **Benutzerverwaltung und Kartenverwaltung**:
    - Die `User`-Klasse verwaltet Benutzerinformationen wie Benutzername, Coins, Kartensammlung und Session-Token.
    - Ein `UserHandler` wurde entwickelt, der Methoden zur Verwaltung der Benutzer, wie das Erstellen neuer Benutzer und das Authentifizieren über Tokens, bereitstellt.
- **Coins-Logik**:
    - Eine `CoinHandler`-Klasse wurde eingeführt, die Methoden wie `EarnCoins`, `SpendCoins`, und `GetBalance` zur Verwaltung des Coinsystems bereitstellt.
    - Jede Transaktion oder jedes Spielereignis kann Coins generieren oder verbrauchen.

2.5 Kartensystem und Elemente

- **Karten-Elemente**:
    - Jede Karte wurde einem bestimmten Element zugeordnet (z.B. Wasser, Feuer, Erde, Normal). Diese Elemente beeinflussen, wie Karten im Kampf miteinander interagieren.
    - Eine Enumeration `ElementType` beschreibt die verschiedenen Kartenelemente: `Water`, `Fire`, `Earth`, `Normal`.
- **Kartenmanagement**:
    - Das Kartenmanagement umfasst die Erstellung, Bearbeitung und Löschung von Karten. Jede Karte hat einen Namen, einen Basiswert für den Schaden und einen Elementtyp, der im Kampf verwendet wird.

3. Nächste Schritte

3.1 Erweiterung der Kampf-Logik

- Weitere Mechanismen zur Schadensermittlung und zum Anwendung von Effekten auf Karten im Kampf implementieren.
- Integration eines `BattleLogic`-Handlers zur Verwaltung von Kämpfen und zur Berechnung des Schadens in Abhängigkeit von den Kartenelementen.

3.2 Erweiterung der Benutzerfunktionen

- Implementierung von Funktionen zum Tauschen, Kaufen und Verkaufen von Karten innerhalb der Benutzeroberfläche.
- Sicherstellen, dass Benutzer mit einer einfachen Schnittstelle ihre Karten und Coins verwalten können.

3.3 Erweiterung der Coins-Logik

- Erweiterung der Mechanismen für das Sammeln und Ausgeben von Coins, z.B. durch das Freischalten neuer Karten oder das Gewinnen von Duellen.
- Einführung von Sicherheitsmaßnahmen, um die Coins und Karten vor unbefugten Änderungen zu schützen.

3.4 Optimierungen und Performance

- Weitere Optimierungen der Server- und Datenbankstruktur, um die Performance bei größeren Datenmengen zu verbessern und die Skalierbarkeit zu gewährleisten.
- Implementierung zusätzlicher Sicherheitsprotokolle für die Kommunikation zwischen Server und Client.

4. GitHub Repository

Link zum Repository