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.

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.

3. Nächste Schritte

3.1 Kampf-Mechanismus weiterentwickeln

- Implementierung einer detaillierteren Logik für den Kampf, die den Schaden basierend auf den Kartenelementen und deren Werten berechnet.

3.2 Integration der Erweiterungen für das MTCG

- **Karten-Elemente**:
  - Jede Karte wird einem bestimmten Element (z.B. Wasser, Feuer, Erde, Normal) zugeordnet. Diese Elemente beeinflussen, wie Karten im Kampf miteinander interagieren.
  - Eine Enumeration `ElementType` beschreibt die verschiedenen Kartenelemente: `Water`, `Fire`, `Earth`, `Normal`. Jede Karte erhält ein Attribut, das ihren Elementtyp festlegt.
- **Kampfmechanik**:
  - Es wird eine Logik implementiert, die den Kampf zwischen Karten regelt. Beispielsweise könnte `Fire` Wasser kontern, `Water` gegen `Fire` verlieren, `Earth` gegen `Water` gewinnen, und so weiter.
  - Ein `ElementInteractionHandler` wird hinzugefügt, um die Interaktionen zwischen verschiedenen Kartenelementen zu verwalten.
- **Coins-Logik**:
  - Jede Transaktion oder jedes Spielereignis kann Coins generieren oder verbrauchen. Ein System zur Verwaltung von **Coins** wird eingeführt, das den Spielern ermöglicht, Coins zu verdienen, auszugeben und zu handeln.
  - Eine Klasse `CoinHandler` wird erstellt, die Methoden wie `EarnCoins`, `SpendCoins`, und `GetBalance` zur Verwaltung des Coinsystems bereitstellt.
- **Erweiterte Kampf-Logik**:
  - Ein `BattleLogic` wird erstellt, um den gesamten Kampfprozess zu verwalten, mit Methoden wie `DetermineWinner`, `CalculateDamage`, und `ApplyEffects`.
- **Benutzerverwaltung und Kartenverwaltung**:
  - Ein `UserHandler` verwaltet die Benutzer und ihre Karten, ermöglicht es Spielern, Karten zu kaufen, zu tauschen und in Kämpfen einzusetzen.
  - Ein `CardHandler` verwaltet die Kartensammlung, einschließlich Erstellen, Bearbeiten und Löschen von Karten.

3.3 Integration von Benutzeraktionen

- Spieler sollen die Möglichkeit erhalten, Karten zu tauschen, zu kaufen oder zu verkaufen. Dazu wird eine Schnittstelle für die Benutzerverwaltung entwickelt.

3.4 Erweiterung der Coins-Logik

- Weitere Mechanismen für das Sammeln und Ausgeben von Coins einbauen, z.B. durch den Erwerb neuer Karten oder das Freischalten von zusätzlichen Spielinhalten.

3.5 Sicherheit und Optimierung

- Sicherheitsmaßnahmen einführen, um sicherzustellen, dass keine unbefugten Änderungen an den Coins oder Karten vorgenommen werden können.
- Optimierung der Performance der Datenbankabfragen, um eine schnelle und skalierbare Benutzererfahrung zu ermöglichen.

4. GitHub Repository

Link zum Repository