Prisoner's Dilemma

Matthias Grill

March 2019

Inhaltsverzeichnis

1 Lizenz

2 Aufgabenstellung

Aufgabe dieses Projekts ist es ein netzwerk-basiertes iteratives Prisoner's Dilemma mit zentralem Server zu implementieren. Wobei jeder Client die Möglichkeit hat seine eigene Strategie zu implementieren und anschließend diese gegen den anderen anzuwenden.

2.1 Technologien

Prisoner's Dilemma wurde in C++17 mit der Hilfe von folgenden Technologien realisiert:

- Clipp
- Spdlog
- JSON
- Asio
- Google Protocol Buffers

3 Spielerklärung

Das Gefangenendilemma (Englisch: "Prisoner's Dilemma") ist ein mathematisches Spiel aus der Spieltheorie. Es modelliert die Situation zweier Gefangener, die beschuldigt werden, gemeinsam ein Verbrechen begangen zu haben. Die beiden Gefangenen werden einzeln verhört und können nicht miteinander kommunizieren. Leugnen beide das Verbrechen, erhalten beide eine niedrige Strafe von zwei Jahren, da ihnen nur eine weniger streng bestrafte Tat nachgewiesen werden kann. Gestehen beide, erhalten beide dafür eine hohe Strafe von vier Jahren, wegen ihres Geständnisses aber nicht die Höchststrafe. Gesteht jedoch nur einer der beiden Gefangenen, geht dieser als Kronzeuge straffrei aus, während der andere als überführter, aber nicht geständiger Täter die Höchststrafe von sechs Jahren bekommt.

	Spieler B kooperiert nicht	**Spieler B kooperiert**
Spieler A kooperiert nicht	A: 2 Jahre B: 2 Jahre	A: 6 Jahre B: 1 Jahre
Spieler A koopertiert	A: 1 Jahre B: 6 Jahre	A: 4 Jahre B: 4 Jahre

Tabelle 1: Visualisierung der Spielregeln

4 Implementierung

4.1 Programmablauf

1. Server wird gestartet
2. Server wartet auf zwei Clients
3. Clients werden gestartet
4. Clients schicken einen Join-Request an den Server
5. Wenn zwei Clients verbunden sind benachrichtigt der Server die Clients über den Spielstart
6. Ablauf für x Spielrunden:
   (a) Server wartet auf zwei Entscheidungen
   (b) Clients schicken ihre Entscheidungen
   (c) Server schickt Ergebnis an die Clients zurück
7. Server schickt das Endergebnis an die Clients
8. Server schließt die Verbindungen zu den Clients
9. Clients schließen die Verbindung zum Server
10. Server startet neu
11. Clients beenden sich

4.2 Kommandozeilen Benutzerschnittstelle

Die Kommandozeilen Benutzerschnittstelle wurde mit Hilfe von Clipp realisert. Folgende Schnittstellen werden zur Verfügung gestellt:

Prisoner's Dilemma Server

```
Usage: ./server [OPTIONS]

Options:
-p, --port          Port to connect to                (int)
-r, --rounds        Number of rounds you want to play (int)
-d, --debug         Debug mode                        (bool)
-h, --help          Print this help message and exit  (bool)
```

Prisoner's Dilemma Client

```
Usage: ./client [OPTIONS]

Options:
-st, --strategy     Your own strategy will be used    (bool)
-s,  --server       Ip adress of server               (string)
-p,  --port         Port to connect to                (int)
-d,  --debug        Debug mode                        (bool)
-h,  --help         Print this help message and exit  (bool)
```

Beispiel-Aufrufe

```
./server
./server -p 8080
./server --rounds 5
./server -p 8080 --debug

./client
./client -p 8080
./client -d --strategy
./client -st --server "192.168.0.110" --debug
```

4.3 Konfiguration

Die Standardkonfiguration des Servers und Clients erfolgt mittels zwei verschiedener JSON-Dateien (Welche sich in /src/static befinden). Die Einstellungen welche hier getroffen werden, können jedoch durch die Kommandozeilen Benutzerschnittstelle überschrieben werden.

Prisoner's Dilemma Server

```
{
    "port": 8081,
    "rounds": 3,
    "debug": false
}
```

- Port: Port auf dem der Server gestartet wird
- Rounds: Anzahl der Runden die gespielt werden
- Debug
  - True: Debug-Nachrichten werden ausgegeben
  - False: Debug-Nachrichten werden nicht ausgegeben

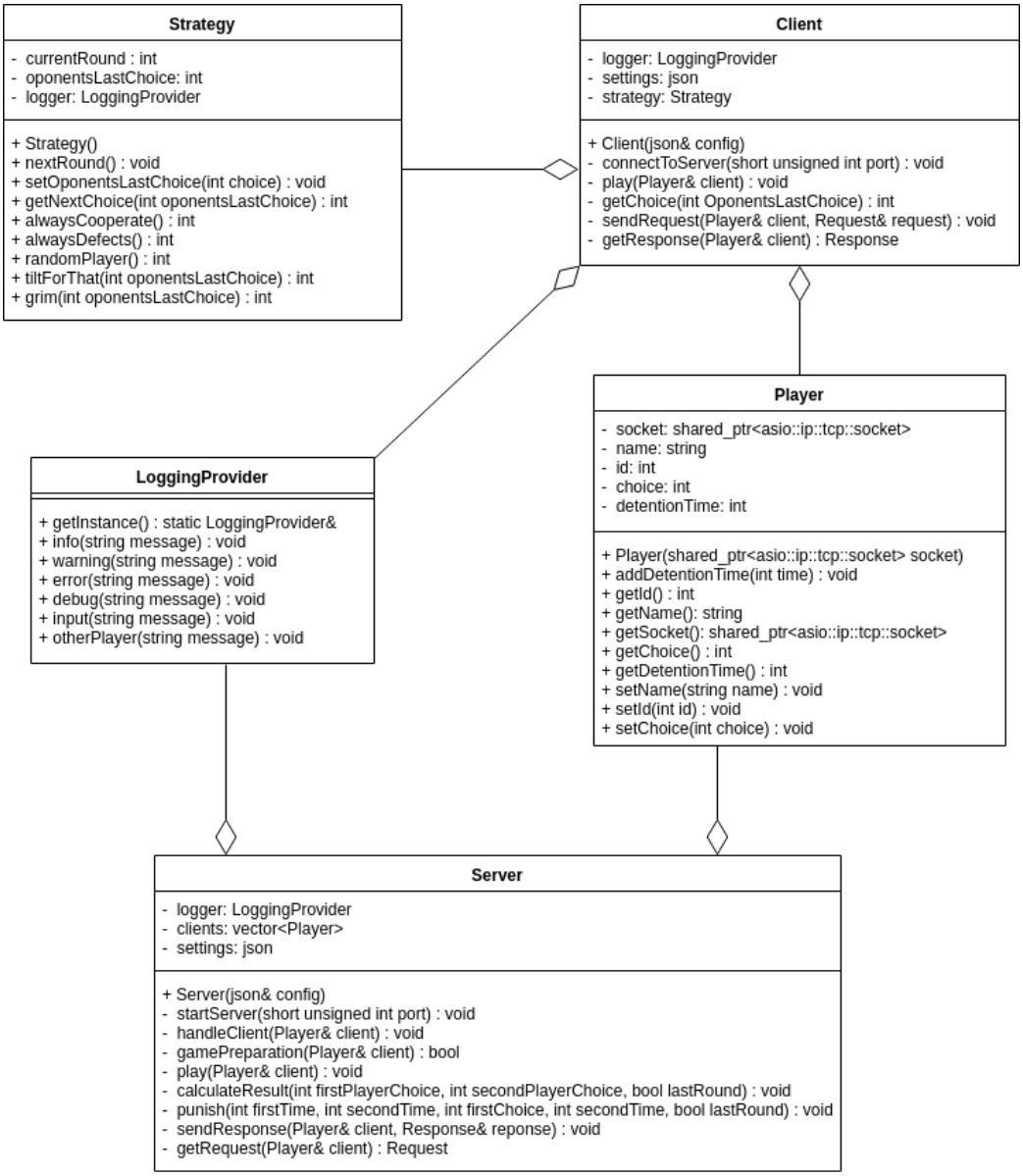
Prisoner's Dilemma Client

```
{
    "playOnCommandLine": true,
    "serverIp": "localhost",
    "port": 8081,
    "debug": false
}
```

- PlayOnCommandLine
  - True: Client kann die Entscheidungen auf der Kommandozeile treffen
  - False: Es wird automatisch die eigens implementierte Strategie verwendet

- ServerIp: IP-Adresse des Servers
- Port: Port des Servers
- Debug
  - True: Debug-Nachrichten werden ausgegeben
  - False: Debug-Nachrichten werden nicht ausgegeben

4.4 UML Klassendiagramm



5 Strategie

Für den Client gibt es zwei verschiedene Möglichkeiten Prisoner's Dilemma zu spielen. Einerseits kann er es über die Kommandozeile spielen und anderseits gibt es auch eine eigene Schnittstelle, bei welcher es die Möglichkeit gibt, die eigene Strategie zu implementieren. Des Weiteren muss nicht jeder Client den gleichen Spielstil wählen, also ist es auch möglich, dass ein Spieler über die Kommandozeile spielt und der andere verwendet seine eigene Strategie welche er im vorhinein implementiert hat.

5.1 Schnittstelle

In der Strategy-Klasse gibt es für den Spieler die Möglichkeit seine eigene Strategie zu implementieren. Des Weiteren gibt es schon ein paar vordefinierte Strategien welche natürlich auch verwendet werden könnnen. Folgende Strategien sind bereits implementiert:

- AlwaysCooperate: Kooperiert immer
- AlwaysDefects: Kooperiert nie
- RandomPlayer: Zufällige Entscheidung
- TiltForThat: Kooperiert beim ersten mal und ab dann entscheidet er sich immer für das selbe wie der andere Spieler in der Vorrunde
- Grim: Kooperiert so lange bis der andere Spiele kooperiert ab dann koopiert er nicht mehr

Will man einer dieser Strategien andwenden muss man sie in der getNextChoice() Methode aufrufe. Will der Spieler seine komplett eigene Strategie implementieren muss er diese ebenfalls in der getNextChoice() Methode implementieren. Folgende Variablen stehen ihm in dieser Funktion bereits zur Verfügung:

- OpponentsLastChoice: Die Entscheidung des Gegner in der letzten Runde. In der ersten Runde beträgt der Wert -1, da es ja noch keine Entscheidung gab
- CurrentRound: Derzeitige Runde (Beginnend mit 1)