Schiffe Versenken

Raphael Futschik
5BHIF
Katalognummer: 6
Aufgabe: 23

Contents

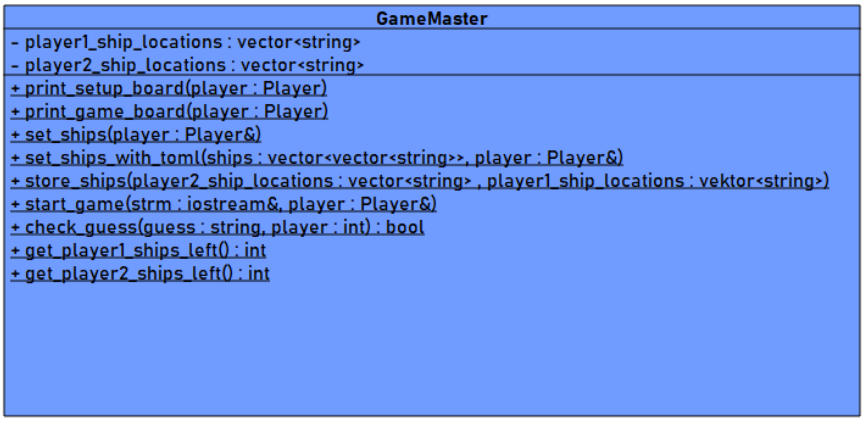
1 Schiffe Versenken

1.1 Das Spiel

Schiffe Versenken ist ein beliebtes das meistens mit Stift und Zettel gespielt wird. Es geht darum, die Schiffe des Gegners zuerst zu versenken, bevor die eigenen Schiffe zerstört werden. Beide Spieler zeichnen am Anfang ihre Schiffe auf Papier, danach versuchen beide Spieler abwechselnd die Position der Schiffe zu erraten. Die Anzahl und Größe der Schiffe variiert dabei je nach Version.

1.2 Realisierung in C++

Zuerst einmal muss man sich die Situation ohne Client-Server Kommunikation vorstellen. Welche Daten sind nötig um Schiffe versenken zu spielen? Hierzu können wir uns die UML Diagramme anschauen:



GameMaster Klassen Diagramm

Player Klassen Diagramm

Die allermeisten Variablen und Funktionen werden nur für die visuelle Ausgabe des Spieles benötigt. Im Grunde wird von der Klasse Player nur ship_locations und add_ship benötigt. In ship_locations werden außerdem alle Position der Schiffe einzeln als string gespeichert, da es keinen Sinn macht jedes Schiff als eigenes Objekt zu betrachten. Wenn ein Schiff oder Schiffteil getroffen wird, wird es daraus gelöscht. Es wird allerdings nicht einmal aus den ship_locations des Players gelöscht, sondern aus dem GameMaster. Der GameMaster übersieht den Spielstatus, während die Player Klasse sich um die Client-seitige Ausgabe kümmert.

2 Projektumsetzung

2.1 Projektbeschreibung

Es soll das Spiel Schiffe Versenken soll mit einem Server und 2 Clients umgesetzt werden. Die Clients sollen sich für ein Spiel anmelden können, dann spielen und auch ihren Punktestand abfragen können. Die Daten der Schiffe sollen mit TOML eingelesen werden können.

2.2 Client-Server Kommunikation

Die Client-Server Kommunikation wurde mithilfe von ProtoBuf realisiert. Es gibt mehrere Phasen die, der Server und der Client durchlaufen muss, um ins Spiel zu gelangen.

Phase 1 - Verbindung aufbauen

Der Server wird gestartet und wartet bis sich 2 Clients zu ihm verbunden haben. Er speichert die Verbindungen und schickt den Clients eine Nachricht, dass das Spiel beginnt.

Phase 2 - Schiffe aufsetzen

Dieser Schritt passiert automatisch, falls beide Clients ihre Schiffe mit TOML aufsetzen. Falls dies nicht der Fall ist, wird solange gewartet bis der/die Client/s ihre Schiffe fertig aufgesetzt haben. Wenn ein Client fertig ist, schickt er seine Schiffe dem Server, der sich die Schiffe speichert und wartet bis jeder der 2 Clients die Schiffe geschickt hat. Sobald das der Fall ist schickt der Server den Clients die Nachricht um das Spiel zu starten.

Phase 3 - Das Spiel

Beide Clients betreten den Spiel Zyklus mit unterschiedlichen Rollen. Der eine Client beginnt mit seinem ersten Zug während der andere auf den nächsten Zug wartet. Derjenige der am Zug ist schickt dem Server seine Vermutung. Der Server überprüft ob es sich um einen Treffer handelt. Danach schickt der Server die Nachricht für den nächsten Zug. Durch das visuelle Feedback sieht jeder Client, welche Positionen er und sein Gegner schon geraten haben und welche davon Treffer waren und welche nicht.

2.3 Client-Server Kommunikation - ProtoBuf

Wie oben erwähnt, wurde ProtoBuf in diesem Projekt benutzt. In a Nutshell kann man sagen, dass mit ProtoBuf immer Nachrichten gesendet werden und auf diese Nachrichten gewartet wird. Im der .proto Datei wird definiert, was in den Nachrichten mitgeschickt wird. Es werden nur primitive Datentypen verwendet, allerdings werden nested types wie repeated, required oder optional unterstüzt. Die letzten 2 erklären sich von selbst, mit repeated können statt zum Beispiel nur einem Integer, mehrere gesendet werden.

```
if(strm) {
    cout << "Waiting for a Opponent to join..." << endl;
    strm << name << endl;

    string data;

    getline(strm, data);

    OpponentMsg msg;
    msg.ParseFromString(Base64::from_base64(data));
    cout << msg.name() << endl;
```

Client wartet auf Nachricht des Servers um das Spiel zu starten

```
if(strm) {
    cout << "Waiting for a Opponent to join..." << endl;
    strm << name << endl;

    string data;

    getline(strm, data);

    OpponentMsg msg;
    msg.ParseFromString(Base64::from_base64(data));
    cout << msg.name() << endl;
```

Server sended beiden Clients die Nachricht um das Spiel zu starten

2.4 Dateneingabe - TOML

TOML ist definitiv ein zentraler Teil dieses Projektes, man kann zwar seine Schiffe auch manuell erstellen, dies dauert allerdings deutlich länger. Die einzige Aufgabe des Benutzers ist es, ein korrektes TOML file zur Verfügung zu stellen. Falls der User einen Fehler in der Datei hat, wird der genaue Grund dafür geloggt und kann einfach behoben werden. In C++ benötigt man nicht mehr als 2 Zeilen um die Daten aus der Datei zu bekommen. Das einzige worauf geachtet werden muss, sind die Datentypen. In TOML sind quasi alle Datentypen unterstützt, ein Array wird aber zum Beispiel in einen Vektor in C++ umgewandelt. Deshalb muss man bei komplexen Datentypen, zumindest bei C++ aufpassen.

```cpp
const auto data  = toml::parse(toml_path);
const auto ships = toml::find<vector<vector<string>>>(data, "ships");
```

Aufnahme der Daten des TOML files in C++

```toml
ships = [["A2", "A4"],
         ["C2", "C5"],
         ["F2", "G2"],
         ["I2", "I2"],
         ["H5", "I5"],
         ["B9", "B9"],
         ["F7", "F8"],
         ["J7", "J7"],
         ["I9", "I9"],
         ["D10", "F10"],
        ]
```

Beispiel eines TOML files

2.5 Datenspeicherung - JSON

Neben Logging, werden in dem Projekt durch JSON Daten gespeichert. Dabei wird die Bibliothek JSON for Modern C++ von Niels Lohmann verwendet. Es muss lediglich json.hpp inkludiert werden um zu funktionieren. Wie im unteren Beispiel zu sehen ist, ist es einfach in json files zu schreiben, man definiert ein json Objekt, in dem man alle Werte speichert die man möchte. Dann muss man nur noch den Pfad festlegen und man kann es dort speichern.

```
jsonf jsonfile;

if (won == player.number){
    jsonfile["game"] = "You won vs " + player.opponent_name;
} else {
    jsonfile["game"] = "You lost vs " + player.opponent_name;
}


if (player.number == 1){
    ofstream file("../../stats/player1/stats.json");
    file << jsonfile;
} else {
    ofstream file("../../stats/player2/stats.json");
    file << jsonfile;
}
```

Es wird gespeichert gegen wen man gewonnen/verloren hat

2.6 Projekt Bedienung

Damit das Spiel gestartet werden kann, muss immer zuerst der Server gestartet werden. Am Server muss nicht konfiguriert werden, er wartet einfach nur bis sich 2 Clients verbinden. Der Server könnte hierbei auch von einer 3. Partei bereitgestellt werden. Als Client kann man ein paar Option auswählen die hier beschrieben sind.

- `name` → Der Username des Spielers(erforderlich
- `port` → Ändert den Port zu dem sich der Client verbindet
- `toml_path` → Setzt die Schiffe mit einem TOML file auf
- `save` → Speichert die Siege/Niederlagen in ein JSON file.
- `save_game` → Speichert das gesamte Spiel in einem JSON file
- `save_game_path` → Spezifiziert den Speicherort für die Datei mit dem Spiel