ulm university universität
uulm

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Softwaretechnik und Programmiersprachen

Softwaregrundprojekt Meilenstein 6

Softwaregrundprojekt an der Universität Ulm

Vorgelegt von:
Gruppe 10

Dozent:
Florian Ege

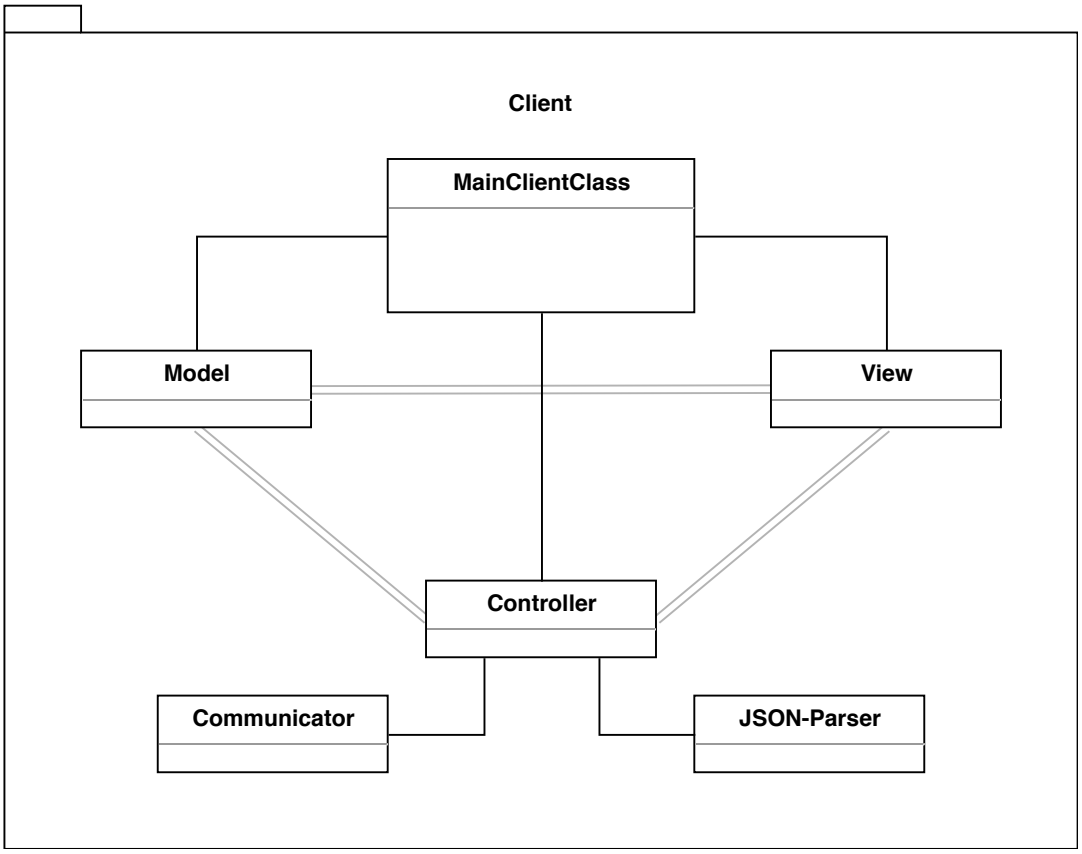
Betreuer:
Stefanos Mytilineos

2019

Inhaltsverzeichnis

1 Klassen Übersicht

1.1 Klassen-Diagramm

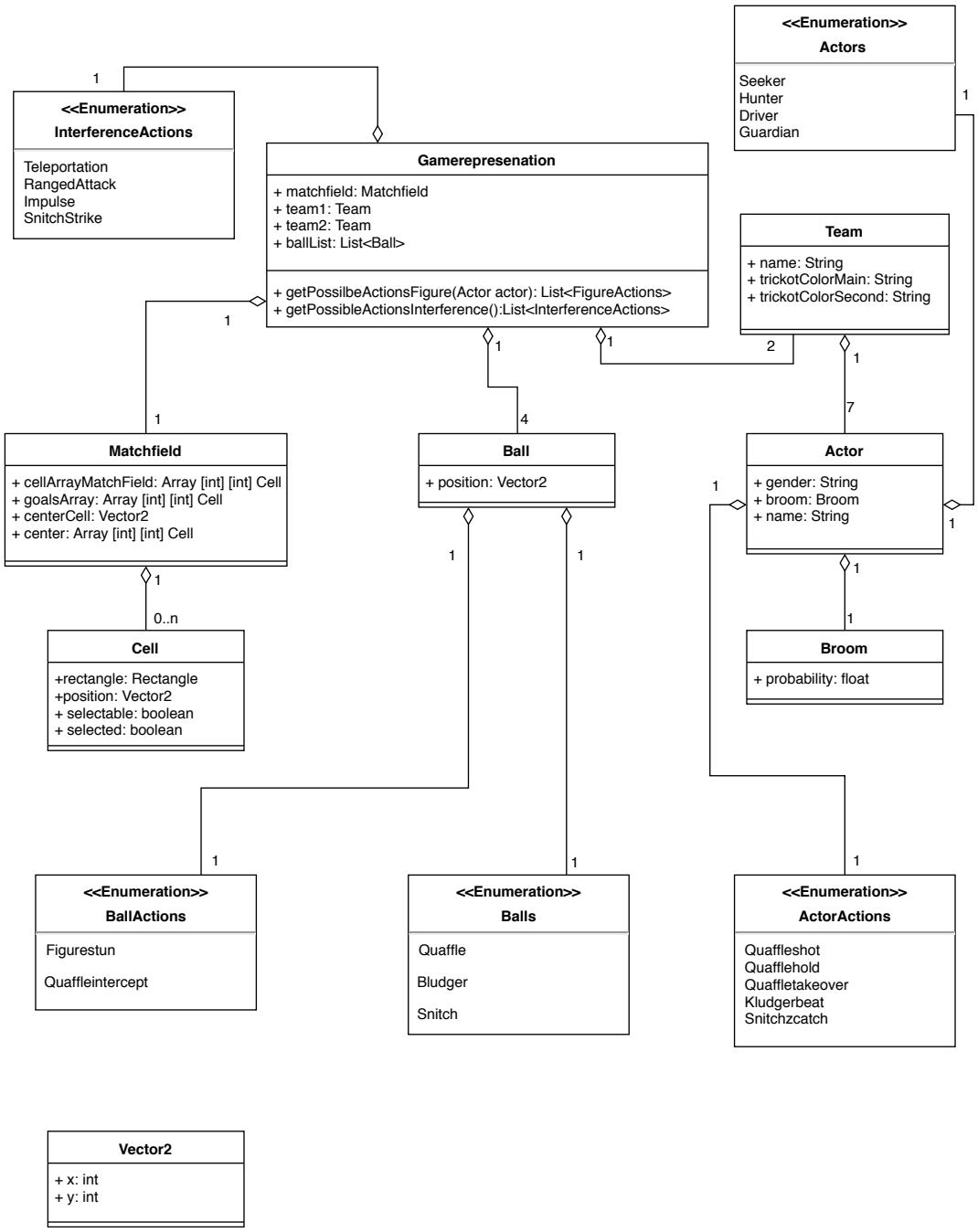


1.2 Beschreibung

Die Grundlegende Struktur der Client Anwendung basiert auf dem Model-View-Controller Modell (MVC Modell). Dabei werden alle Aufgaben weitestgehend auf drei unabhängige Klasse verteilt. In der Model Klasse sind alle für die Funktion der Anwendung benötigten Datenstrukturen enthalten. Die View Klasse ist dafür verantwortlich alle Daten in einer geeigneten Form darzustellen. Die Controller Klasse dient dazu, basierend auf verschiedenen Ereignissen, wie z.B. Benutzereingaben, die Daten im Model anzupassen, sodass in diesem Fall z.B. eine neue Spielsituation von der View abgebildet werden kann. Die Klassen Communicator und JSON-Parser stelle Hilfsklassen der Controller Klasse dar.

2 Model

2.1 Klassen-Diagramm



2.2 Beschreibung

Die Model ist für die Repräsentation für die Daten verantwortlich. Hier sind alle Daten hinterlegt, die für das Spiel relevant sind und angezeigt werden in der Benutzeroberfläche. Die Daten wertet der Controller aus und gibt diese dann der View weiter, welche diese dann auf der Benutzeroberfläche an.

2.2.1 Gamerepresentation (Klasse)

Konstruktor Der Konstruktor erstellt eine Instance des Objektes Gamerepresentation. In diesem Objekt sind alle Objekte die für das Spiel notwendig sind und auf dem Spielfeld agieren enthalten und jeweils eine Instanz erstellt.

getPossibleActionsFigure (Methode) Diese public Methode hat den Zweck eine Liste zu erstellen, in der alle möglichen Aktionen aufgelistet werden, die eine übergebene Spielfigur machen kann. Dazu überprüft die Methode welchen Typ Figur die übergeben Spielfigur hat, und welche AKtionen sie dadurch durchführen darf und gibt diese als Liste wieder.

getPossibleActionsInterference (Methode) Diese public Methode gibt die in der Enumeration definierten Einmischungen als Liste zurück. Dazu liest sie die Enumeration ein und spiechert diese in einer Liste, welche dann zurückgegeben wird.

2.2.2 Matchfield (Klasse)

Konstruktor In dem Konstruktor für das Matchfield, wird das eigentliche Spielfeld erstellt. Dazu erstellt diese Klasse ein Array von Zellen, welche das Spielfeld repräsentieren. Außerdem gibt es noch Bereiche, die besonders sind, aber auch Zellen auf dem Spielfeld sind. So sind die Torringe ebenfalls in einem Array hinterlegt, das Zentrum ist als Vektor hinterlegt und das Mittelfeld ist ebenfalls als Array hinterlegt.

2.2.3 Team (Klasse)

Konstruktor Das Team ist ebenfalls Bestandteil der Klasse Gamerepresentation. In dem Team werden die 7 Spieler instantiiert, so wie der Name, die Trikotfarbe und die Ersatzfarbe für die Trikots.

2.2.4 Ball (Klasse)

Konstruktor Durch die Klasse Ball wird ebenfalls in der Gamerepresentation ein Bestandteil des Spieles instantiiert. Die Bälle bestehen aus einer Enumeration, welche Bälle es gibt, so wie aus einer Position, die als Vektor repräsentiert wird. Außerdem besitzt die Klasse Ball noch eine Enumeration, welche Aktionen ein Ball machen kann.

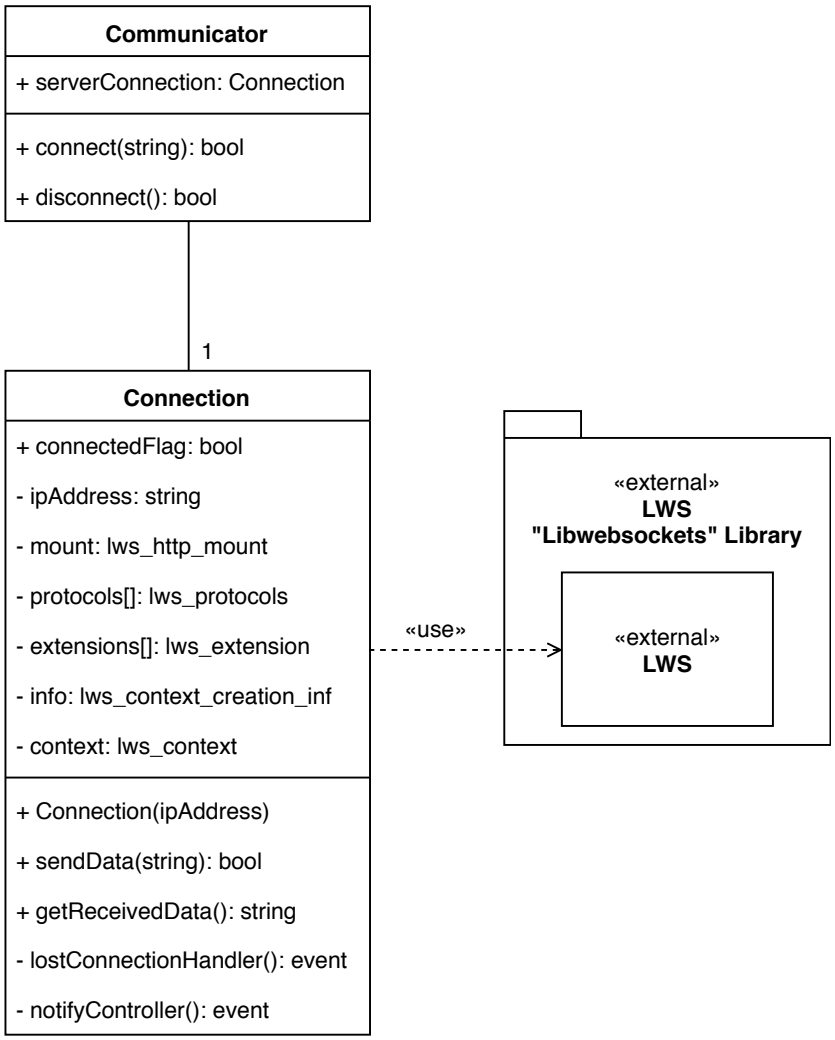
2.3 Zuordnung der Funktionalen Anforderungen

Die Funktionalen Anforderungen werden den Methoden folgendermaßen zugeteilt:

Methode	**Funktionelle Anforderungen**
getPossibleActionsFigure	FA21, FA24, FA26, FA28, FA30
getPossibleActionsInterference	FA32 - FA35

3 Kommunikator

3.1 Klassen-Diagramm



3.2 Beschreibung

3.2.1 Connector (Klasse)

Im Allgemeinen dient die Connector Klasse dazu, die Kommunikation zwischen der Client Anwendung und der Server Anwendung über das Netzwerk zu verwalten.

connect (Methode) Die connect Methode dient dazu, eine neue Verbindung mit einem Server aufzubauen. Die IP-Adresse des Servers wird dabei als Parameter übergeben. Ist die Aktion erfolgreich, so wird true zurück gegeben, anderenfalls false.

disconnect (Methode) Die disconnect Methode dient dazu, eine bestehende Verbindung mit einem Server zu trennen. Ist die Aktion erfolgreich, so wird true zurück gegeben, anderenfalls false.

3.2.2 Connection (Klasse)

sendData (Methode) Die öffentliche sendData Methode dient dazu, einen im JSON-Format formatierten String an den Server zu übertragen. Ist die Aktion erfolgreich, so wird true zurück gegeben, anderenfalls false. Zur Umsetzung dieser Methode werden Funktionen und Strukturen der Drittanbieter-Bibliothek LWS benötigt.

getReceivedData (Methode) Mit Hilfe der öffentlichen Methode getReceivedData kann der letzte vom Server gesendete JSON Datensatz ausgelesen werden. Zur Umsetzung dieser Methode werden ebenfalls Funktionen und Strukturen der Drittanbieter-Bibliothek LWS benötigt.

lostConnectionHandler (Methode) Die private Methode lostConnectionHandler kümmert sich im Falle eines außerplanmäßigen Verbindungsverlust darum, dass alle notwendigen Schritte eingeleitet werden, indem der Controller über den Verbindungsabbruch informiert wird. Auch hier werden zur Umsetzung Funktionen und Strukturen der Drittanbieter- Bibliothek LWS benötigt.

notifyController (Methode) Die private notifyController Methode ist dafür zuständig den Controller über das Eingehen von neuen Nachrichten des Servers zu informieren.

3.2.3 LWS (Externe Bibliothek)

Bei dieser Klasse Handelt es sich um eine externe Bibliothek, die verschiedenste Methoden und Strukturen zur Verfügung stellt um in einem C/C++ Projekt einfache Server Client Verbindungen mittels Web-Sockets zu implementieren. Genauere Informationen zu dieser Bibliothek sind unter folgendem Link zu finden:
`https://libwebsockets.org/`

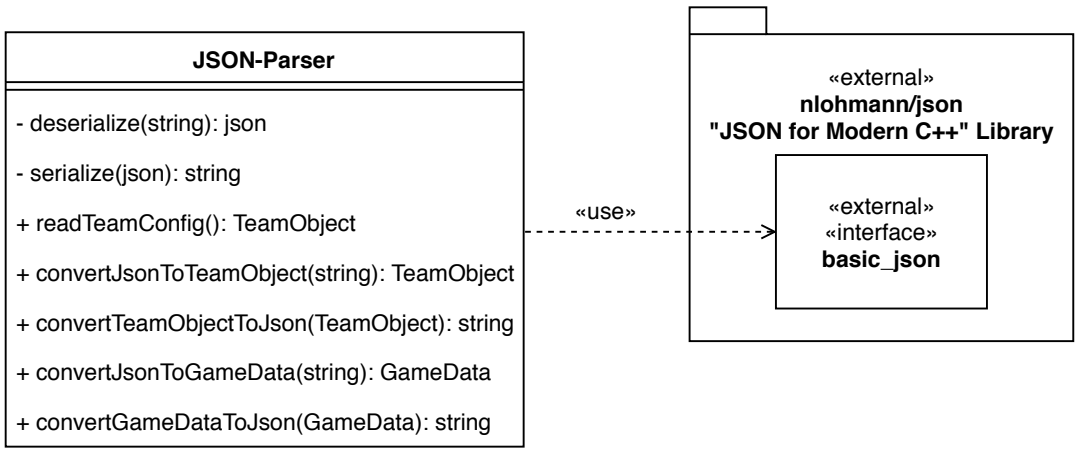
3.3 Zuordnung der Funktionalen Anforderungen

Die Funktionalen Anforderungen werden den Methoden folgendermaßen zugeteilt:

Funktionale Anforderungen	Methoden
FA55	Connector::connect
FA55	Connector::disconnect
FA55	Connection::sendData
FA55	Connection::getReceivedData
FA55	Connection::lostConnectionHandler
FA55	Connection::notifyController

4 JSON-Parser

4.1 Klassen-Diagramm



4.2 Beschreibung

4.2.1 JSON-Parser (Klasse)

Der JSON-Parser übernimmt das Serialisieren und Deserialisieren von Objekten in einen String im JSON-Format. Diese Funktionalität wird benötigt, da alle Daten, die über das Netzwerk versendet werden oder aus einer Datei eingelesen werden diesem Format entsprechen müssen.

deserialize (Methode) Die private deserialize Methode dient dazu, einen dem JSON-Format konformen String in ein JSON-Object umzuwandeln. Dabei wird bei der Umsetzung dieser Funktionalität auf Methoden und Strukturen aus einer Softwarebibliothek eines Drittanbieters zurückgegriffen.

serialize (Methode) Die private serialize Methode dient dazu, aus einem JSON-Object ein dem JSON-Format genügenden String zu erzeugen. Wie auch schon bei der deserialize Methode wird dabei auf Methoden und Strukturen aus einer Softwarebibliothek eines Drittanbieters zurückgegriffen.

readTeamConfig (Methode) Mit Hilfe der readTeamConfig Methode kann die Team-Konfiguration aus der Team-Konfigurationsdatei eingelesen werden und in ein internes TeamObject umgewandelt werden. Diese wird dann im Model hinterlegt. Im Hintergrund greift diese Methode auf die deserialize Methode zurück.

convertJsonToTeamObject (Methode) Die convertJsonToTeamObject Methode dient dazu, die vom Server empfangenen Daten über das gegnerische Team in ein internes TeamObject umzuwandeln. Diese wird dann im Model hinterlegt. Im Hintergrund greift diese Methode auf die deserialize Methode zurück.

convertTeamObjectToJson (Methode) Die convertGameDataToJson Methode bildet das Gegenstück zur convertJsonToTeamObject Methode. Es kann also ein internes GameTeam in einen JSON konformen String konvertiert werden, welcher dann an den Server versendet werden kann. Im Hintergrund greift diese Methode auf die serialize Methode zurück.

convertJsonToGameData (Methode) Die convertJsonToGameData Methode dient dazu, die vom Server empfangenen Daten über das aktuelle Spielgeschehen aus dem JSON-Format in ein Internes GameObject zu konvertieren. Diese wird dann im Model hinterlegt. Im Hintergrund greift diese Methode auf die deserialize Methode zurück.

convertGameDataToJson (Methode) Die convertGameDataToJson Methode bildet das Gegenstück zur convertJsonToGameData Methode. Es kann also ein internes GameObject in einen JSON konformen String konvertiert werden, welcher dann an den Server versendet werden kann. Im Hintergrund greift diese Methode auf die serialize Methode zurück.

4.2.2 nlohmann::json (Externe Bibliothek)

Bei dieser Klasse Handelt es sich um eine externe Bibliothek, die verschiedenste Methoden und Strukturen zur Verfügung stellt um in einem C++ Projekt mit Objekten mit JSON-Struktur einfach zu arbeiten. Vermutlich wird die von *Niels Lohmann* entwickelte

freie JSON Bibliothek namens *JSON for Modern C++* verwendet werden. Genauere Informationen zu dieser Bibliothek sind unter folgendem Link zu finden:
`https://github.com/nlohmann/json`

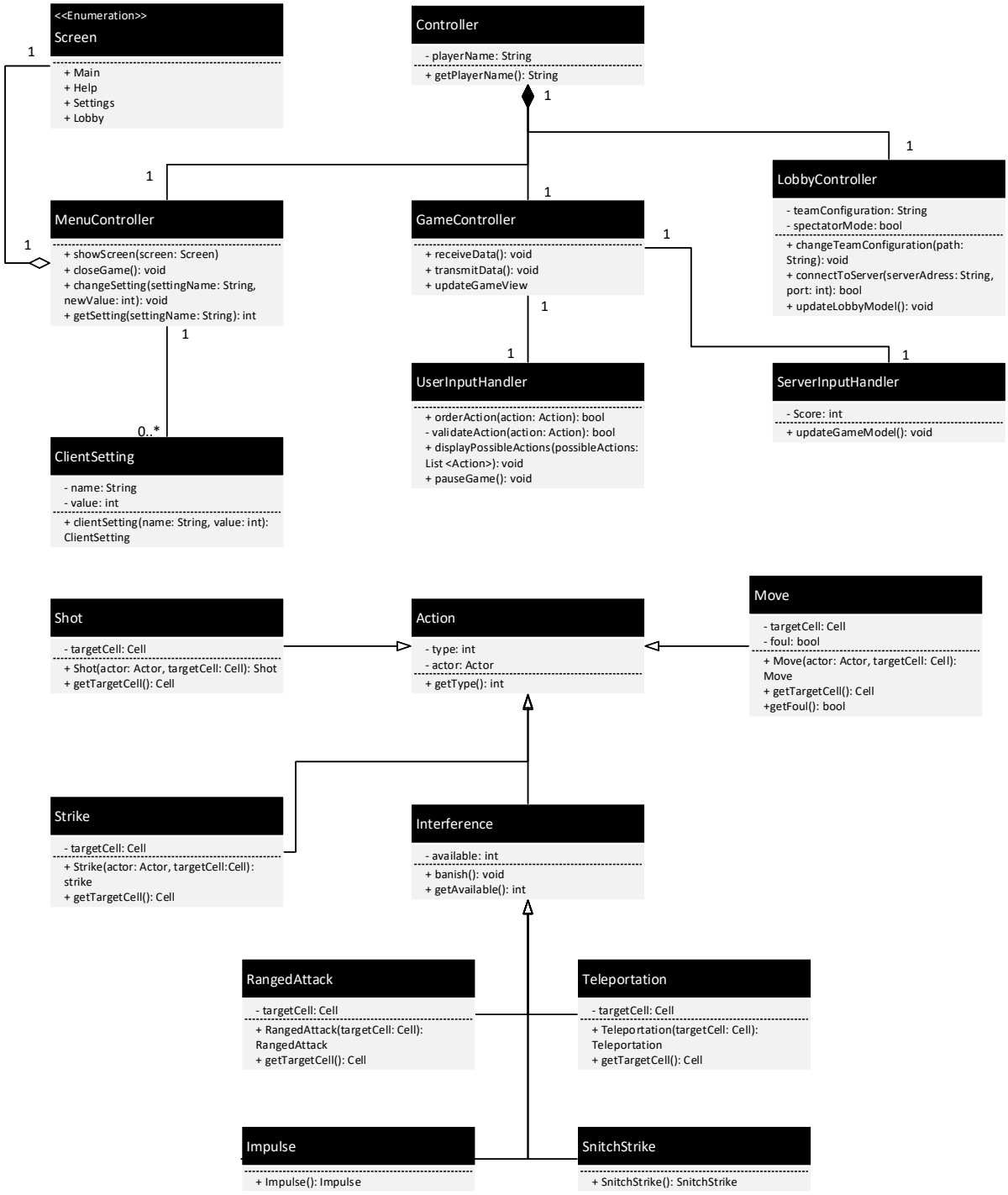
4.3 Zuordnung der Funktionalen Anforderungen

Die Funktionalen Anforderungen werden den Methoden folgendermaßen zugeteilt:

Funktionale Anforderungen	**Methoden**
FA54	readTeamConfig
FA54	convertJsonToTeamObject
FA54	convertTeamObjectToJson
FA53	convertGameDataToJson
FA53	convertJsonToGameData

5 Controller

5.1 Klassendiagramm



5.2 Beschreibung

Im Folgenden werden die eingetragenen Methoden erklärt.

Name	Vorbedingungen	Nachbedingungen	Erklärung
MenuController			
closeGame	-	Positive Antwort in einem Popup-Fenster	Beendet die Anwendung
changeSetting	ClientSetting mit dem angegebenen settingName existiert	Der Wert von newValue wird akzeptiert	Ändert das Attribut value des ClientSetting mit dem Attribut name, der dem Parameter settingName entspricht, auf den Wert des Parameters newValue.
getSetting	ClientSetting mit dem angegebenen settingName existiert	-	Liefert den derzeitigen Wert des Attributs value des ClientSetting, dessen Attribut name mit dem Parameter settingName übereinstimmt.
GameController			
receiveData	-	-	Empfängt Daten vom JSON-Parser und übergibt sie dem ServerInputHandler
transmitData	-	-	Empfängt Daten vom UserInputHandler und sendet sie über den JSON-Parser und den Kommunikator zum Server
LobbyController			
changeTeamConfiguration	-	Parameter path ist ein gültiger Pfad zu einer Team-Konfigurations-Datei	Ändert den Wert von teamConfiguration, in dem der Pfad zu der zu verwendenden Team-Konfigurations-Datei gespeichert ist.

connectToServer	Parameter server-Adress und port sind ungleich Null	-	Veranlasst über den JSON-Parser den Kommunikator dazu, eine Verbindung mit dem in den Parametern spezifiziertem Server aufzubauen. Gibt true zurück, wenn der Verbindungsaufbau erfolgreich war, ansonsten false.
UserInputHandler			
validateAction	-	-	Fordert vom Partie-Model Daten an und entscheidet danach, ob die geforderte Action durchgeführt werden kann. Gibt true zurück wenn die als Parameter übergebene Action gültig ist, ansonsten false.
orderAction	validateAction gibt für die als Parameter übergebene Action true zurück.	-	Übergibt die auszuführende Aktion an den GameController. Ruft die Methode validateAction auf und gibt deren Rückgabewert zurück.
displayPossibleActions	-	-	Fordert vom Partie-Model Daten an und berechnet alle möglichen Aktionen und gibt sie an die Partie-View weiter, um sie dem Spieler anzuzeigen.
Interference			
banish	-	-	Verringert das Attribut available um eins.

5.3 Zuordnung der Funktionalen Anforderungen

Die Funktionalen Anforderungen werden den Methoden folgendermaßen zugeteilt:

Funktionale Anforderungen	**Methoden**
FA7	Shot::shot Shot::getTargetCell
FA21	Shot::shot
FA26	Move::move
FA27	Shot::shot
FA28	Strike::strike
FA30	Move::move
FA31	RangedAttack::rangedAttack Teleportation::teleportation Impulse::impulse SnitchStrike::snitchStrike Inteference::banish
FA32	Teleportation::teleportation
FA33	RangedAttack::rangedAttack
FA34	Impulse::impulse
FA35	SnitchStrike::snitchStrike
FA36	Inteference::banish Actor::setBanished
FA37	Actor::setBanished Move::move
FA38 - FA42	Move::move
FA48	Actor::setBanished
FA54	LobbyController::changeTeamConfiguration LobbyController::updateLobbyModel
FA55	GameController::receiveData GameController::transmitData LobbyController::connectToServer
FA56	GameController::receiveData GameController::transmitData
FA60-FA61	MenuController::showScreen

FA67	UserInputHandler::orderActions UserInputHandler::validateActions MenuController::showScreen MenuController::changeSetting LobbyController::changeTeamConfiguration LobbyController::connectToServer
FA69	UserInputHandler::pauseGame