Meilenstein 6: Implementierungsentwurf

Bearbeitungsdauer: 16 Stunden, **Abgabetermin:** 09.02.2020

Tutor: Timo Zuccarello

Team: 003, **Mitglieder:** Akın Kula, Atakan Özsoy, Italgo Walter Pellegrino, Jonas Frei, Salih Şakar

1 Einleitung

Im NT2S ("No Time to Spy") System ist eine große Menge an Klassen vorhanden. Ein Diagramm für das ganze System wäre daher zu komplex und unübersichtlich. Deswegen wurde das System anhand ihrer Hauptkomponenten in 3 einzelne Abschnitte ausgelagert. Dabei folgt die Software Architektur des Systems dem MVC-Pattern, das die Aufteilung der Systemkomponenten in diesen Abschnitten begründet.

2 Modell

In diesem Abschnitt wird das Datenmodell des Benutzer-Clients dargestellt.
Im folgenden Diagramm sind Klassen und Enumerations dargestellt, die zur Realisierung des Spiels dienen sollen. Darüber hinaus definieren teilweise die Anwendungslogik. Diese wurden vom Standardisierungskomitee festgelegt. Daher wird an dieser Stelle auf die Standardisierungsdokumente im Team- Git Repo verwiesen. Detaillierte Erklärungen aller Klassen und Enums sind in diesem erklärt. Die bisherigen Ergebnisse des Standardisierungskomitees wurden in diese Arbeit miteinbezogen, da praktisch gesehen diese einheitlich alle bei der Implementierung des Spiels vorgesehen sind.

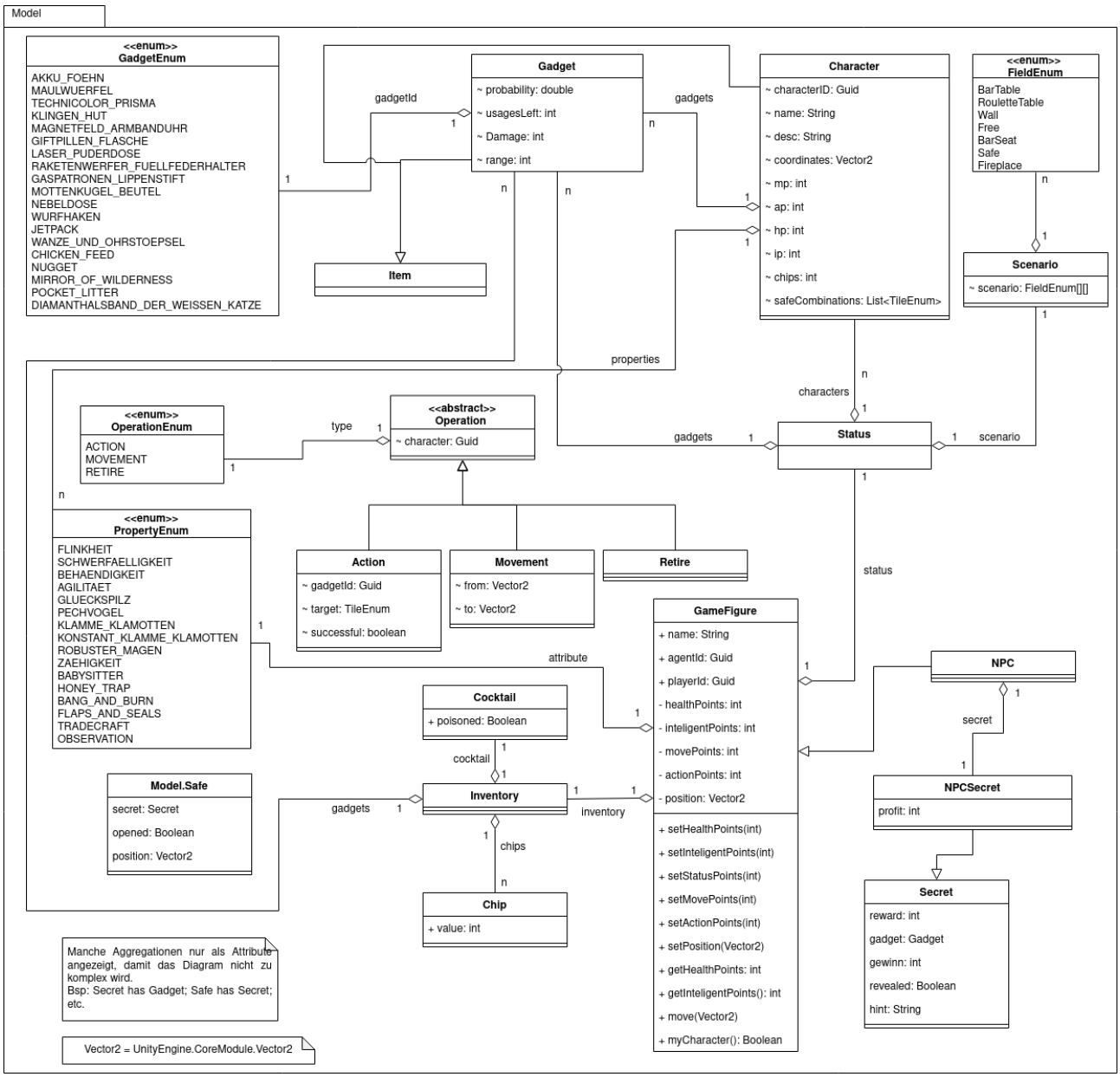

Abbildung 1: Klassendiagramm für das Datenmodell (engl. model) der Benutzer-Client Komponente

Das Modell für die Netzwerkschnittstelle enthält weitere Klassen, die auch vom Standardisierungskomitee veröffentlicht worden sind. Die Mehrheit dieser Klassen sind solche für unterschiedliche Nachricht Typen im Rahmen der Client-Server Architektur. Es ist möglich, diese Klassen getrennt vom lokalen Modell wiederzugeben.
Der folgende Abschnitt zeigt das Datenmodell dieser Klassen getrennt vom lokalen Datenmodell.

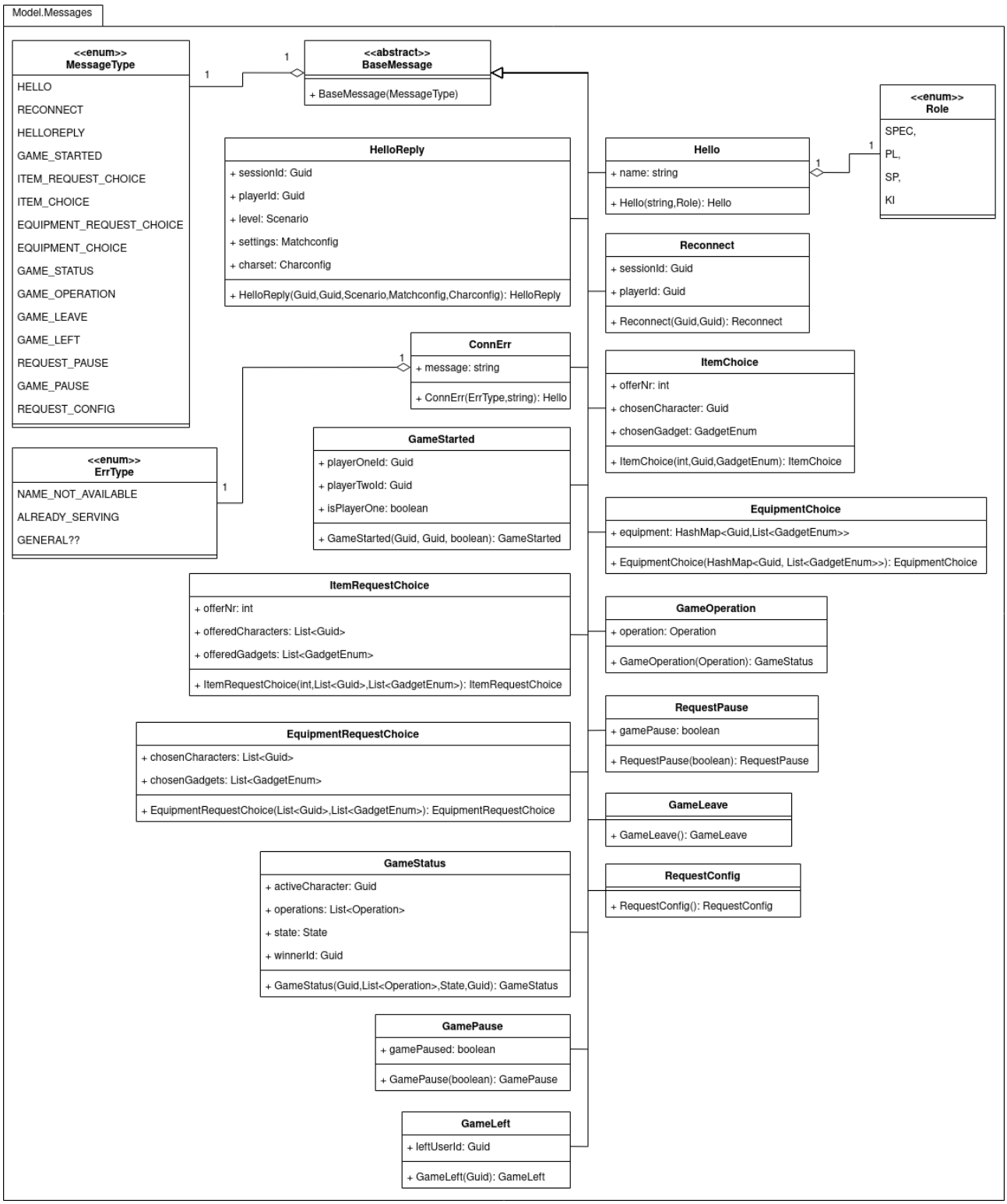

Abbildung 2: Klassendiagramm für das Datenmodell (engl. model) der Netzwerkschnittstelle der Benutzer-Clients

2.1 Zuordnung der funktionalen Anforderungen

Diese Konstanten und Klassen werden im Controller verwendet und werden auch im Nachrichtenverkehr eingesetzt. Die folgende Tabelle zeigt, welche funktionalen Anforderungen durch die Enums und Klassen abgedeckt werden. Die funktionalen Anforderungen sind als Abkürzungen in unserem Pflichtenheft zu finden:

- **GadgetEnum:** FA38-57
- **Gadget:** FA38-57
- **Character:** FA13-37
- **FieldEnum:** FA4-11
- **Scenario:** FA4-11,FA78-79
- **Status:** FA90
- **OperationEnum:** FA58-67
- **Operation:** FA58-67
- **PropertyEnum:** FA17-37
- **GameFigure:** FA13, FA58, FA59, FA38-57, FA60-67
- **Inventory:** FA18
- **NPC:** FA17ff
- **NPCSecret:** FA17ff
- **Secret:** FA68
- **Safe:** FA12
- **MessageType:** FA84-93
- **BaseMessage:** FA84-93
- **ErrType:** FA86,FA97
- **Role:** FA93

3 View

In diesem Abschnitt wird die Repräsentation der Benutzer-Client Komponente dargestellt. Im folgenden Diagramm sind die Screens (Scenes in Unity) dargestellt, die dem Benutzer das Spielmodell zugänglich macht. Bei der Erstellung dieses Diagramms wurde versucht, die Struktur der Unity Game Engine zu integrieren. (z.B. da die Scenes graphisch erzeugt und von dem UnityEngine.CoreModule automatisch verwaltet sind, sind Methoden wie render() aus LibGDX nicht eingetragen, sondern die Klassen enthalten nur die Spiel-bezogenen, von Entwickler zu implementierenden Funktionen - wie Button Events -).

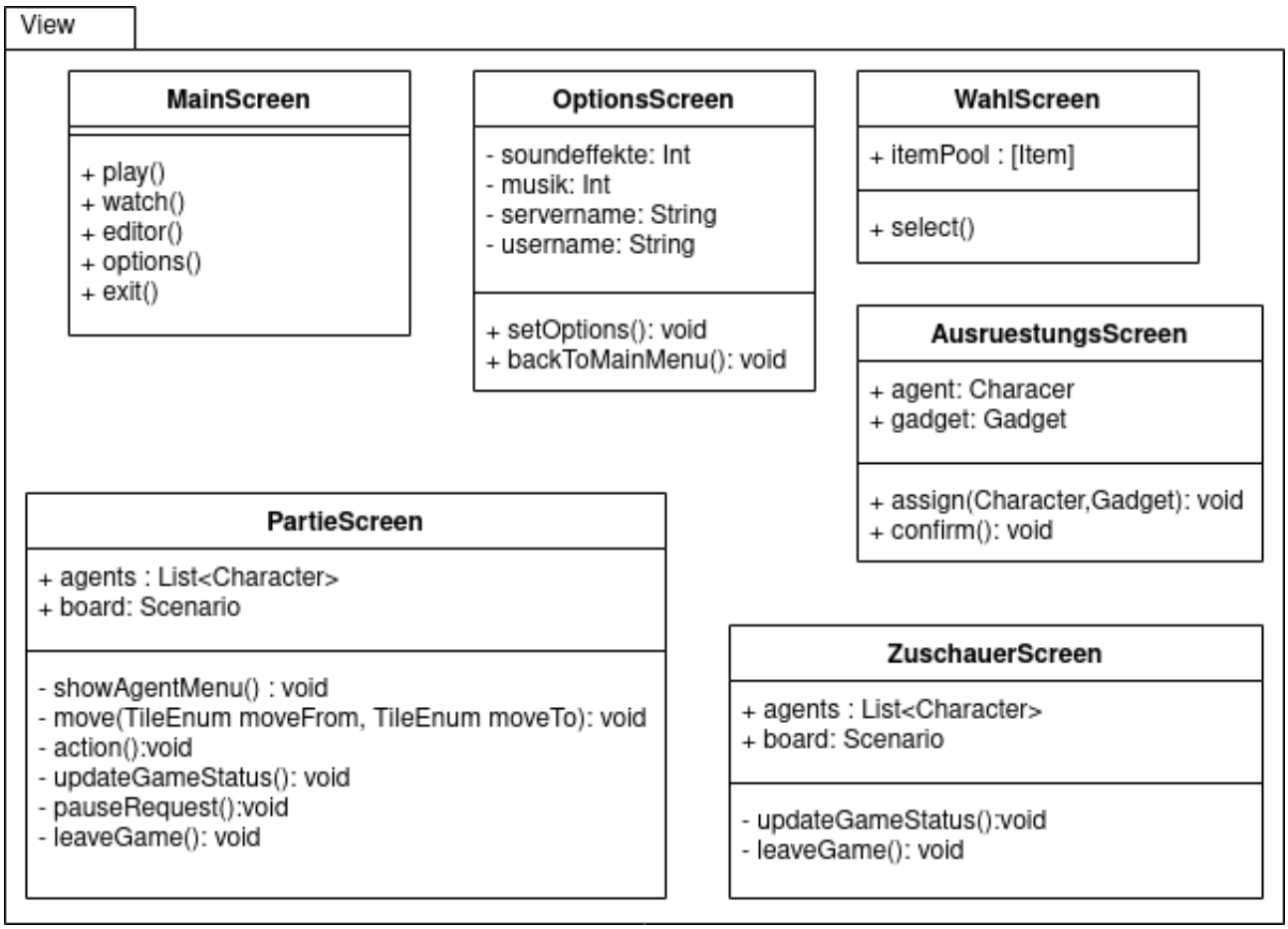

Abbildung 3: Klassendiagramm für die Repräsentation (engl. view) der Benutzer-Client Komponente

3.1 Methodenbeschreibungen

MainScreen

- **play()** Wenn eine IP Adresse gespeichert ist (s. OptionsScreen/setOptions()), versucht der Client sich bei dem Server als Spieler zu registrieren. Falls dies erfolgreich war, erscheint ein "Warten"Label bis ein Spiel bereitsteht. Danach wird das WahlScreen angezeigt.
- **watch()** Wenn eine IP Adresse gespeichert ist (s. OptionsScreen/setOptions()), versucht der Client sich bei dem Server als Zuschauer zu registrieren. Falls dies erfolgreich war, erscheint ein "Warten"Label bis ein Spiel bereitsteht. Danach öffnet sich das Zuschauerfenster.
- **editor()** Editor Komponente wird geöffnet.
- **options()** `Unity.SceneManagement.SceneManager.LoadScene(OptionsScreen);` wird aufgerufen. Diese Methode wird aufgerufen, falls Editor Button gedrückt wird.

OptionScreen

- **setOptions()** Liest die aktuellen Werte der Sliders (sound effect volume & music volume) aus und aktualisiert die Attribute danach. Falls beide Textfelder gefüllt sind (IP Feld passend zum Regex), wird die Adresse für weitere Aktionen gespeichert. Falls ein Problem auftaucht, wird der Benutzer durch ein Popup informiert. Diese Methode wird aufgerufen, wenn Übernehmen Button gedrückt wird.

- **backToMainMenu()** [basicstyle=,language=[Sharp]C]| `Unity.SceneManagement.SceneManager.LoadScene(MainScreen);| wird aufgerufen. Diese Methode wird aufgerufen, wenn Back Button gedrückt wird.`

WahlScreen

- **select()** Diese Methode wird aufgerufen, wenn ein Item Button gedrückt wird. Das Item, das der Button repräsentiert, wird über EquipmentHandler ausgewählt.

AusrüstungsScreen

- **assign(Character,Gadget)** Ruft giveGadget-Methode im EquipmentHandler auf.
- **confirm()** Methodenaufruf, wenn Spieler mit Ausrüstungsphase fertig ist. Sendet die Loadout zu dem Server.

GameScreen

- **showAgentMenu()** Diese Methode wird aufgerufen, wenn ein Agent Button in HUD geklickt wird. Die Details über diesen Agenten werden angezeigt.
- **action()** Benutzeraktion um eine GameOperation Message (Operation vom Typ Action) zu senden.
- **move(Character,Gadget)** Benutzeraktion um eine GameOperation Message (Operation vom Typ Movement) zu senden.
- **updateGameStatus()** Diese Methode wird automatisch vom GameHandler aufgerufen, damit der Spielzustand auf dem Fenster aktualisiert wird.
- **pauseRequest()** Mit dieser Methode wird eine Pause Anfrage über den GameHandler gesendet.
- **leaveGame()** Benutzeraktion um eine GameLeave Message zu senden.

3.2 Zuordnung zur funktionalen Anforderungen

- **MainScreen:** FA1,FA2,FA95
- **OptionScreen:** FA95
- **WahlScreen:** FA69-70,FA95
- **AusruestungsScreen:** FA71,FA95
- **GameScreen:** FA72-77,FA95-96,FA98
- **ViewerScreen:** FA72-77,FA95

4 Controller

In diesem Abschnitt wird die Steuerung der Benutzer-Client Komponente dargestellt. Im folgenden Diagramm sind die einzelnen Controller dargestellt, die den Zustand des Modells und des Views während des Spiels verwalten. Bei der Erstellung dieses Diagramms wurde versucht, die vorimplementierten Klassen aus der Unity Game Engine zu integrieren. Da Unity eine unterschiedlichere Struktur und Funktionsweise als die vorherigen Java Projekte hat, kann es sein, dass die Implementierung des Controllers nicht zu diesem Modell passt.

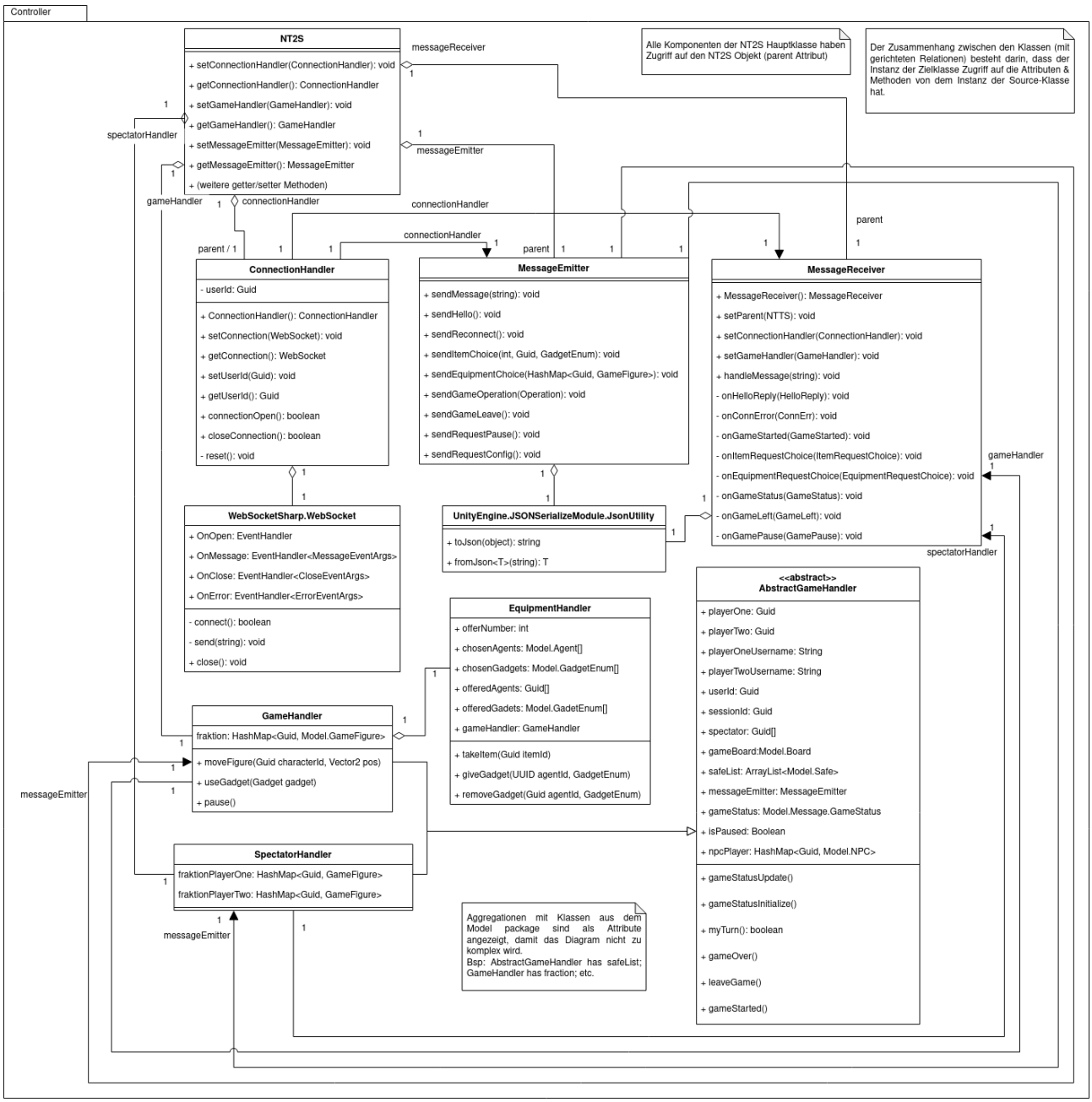

Abbildung 4: Klassendiagramm für die Programmsteuerung (engl. controller) der Benutzer-Client Komponente

4.1 Methodenbeschreibungen

AbstractGameHandler

- **gameStatusUpdate()** Wenn Spiel Status initialisiert worden ist, wird es nach der empfangenen Nachricht aktualisiert.
- **gameStatusInitialize()** Spielstatusinitialisierung nachdem die erste GameStatus Nachricht empfangen wurde.
- **myTurn()** Interpretiert gemäß Spieler IDs und activePlayer ID, ob der Spieler am Zug ist.
- **gameOver()** Wird aufgerufen, wenn das Spiel zu Ende ist. Gewinner wird angezeigt und Spieler wird in das MainMenuScreen weitergeleitet.
- **leaveGame()** Ruft sendGameLeave-Methode vom MessageEmitter auf.

GameHandler

- **moveFigure(Guid,Vector2)** Zum Senden eines GameOpearations vom Typ Movement.
- **useGadget(Gadget)** Zum Senden eines GameOpearations vom Typ Action.
- **pause()** Zum Senden einer Pause Anfrage.

EquipmentHandler

- **takeItem(Guid)** Zum Auswählen eines Items in der Wahlphase.
- **giveGadget(Guid,GadgetEnum)** Gadget wird einem Charakteren zugeordnet (in sein Inventar eingelegt).
- **removeGadget(Guid,GadgetEnum)** Gadget wird aus dem Inventar des Charakters entfernt.

MessageEmitter

- **sendMessage(string)** Methode zum Senden eines bereits erstellten JSON Strings an den Spielserver.
- **sendX(*)** Erzeugt ein Message Objekt vom Typ X mit übergebenen Parametern (falls vorhanden) und wandelt es in einen JSON-String um. Dieser String wird der Methode sendMessage übergeben.

MessageReceiver

- **handleMessage(string)** Wandelt einen empfangenen JSON String in ein Message Objekt um und leitet dieses Objekt je nach Messagetyp an die entsprechende onX Methode weiter.
- **onX(X)** Enthält die Logik dafür, wie das Spiel auf eine Nachricht (vom Typ X) reagiert.

4.2 Zuordnung zur funktionalen Anforderungen

- **NT2S:** FA*
- **AbstractGameHandler:** FA4-37, FA72-74, FA76, FA87, FA90, FA91, F75, FA77, FA89
- **GameHandler:** FA58, FA59, FA88, FA3, FA80, FA81,
- **SpectatorHandler:** FA80, F94
- **EquipmentHandler:** FA70, FA71, FA69
- **ConnectionHandler:** FA84-85
- **MessageEmitter:** FA84-93
- **MessageReceiver:** FA84-93
- **WebSocket:** QA9
- **JsonUtility:** QA9-10