DESIGNDOKUMENT

Version 1.0

1. Einführung und Ziele

Dieses Dokument beschreibt den softwaretechnischen Aufbau des Programms “Roomie Boomie”.
Die Ziele sind dem Pflichtenheft in der Version 1.0 unter dem Abschnitt 1: “Zielsetzung” zu entnehmen.

1.1 Aufgabenstellung

Dieses Spiel gibt dem Nutzer die Möglichkeiten, bereits vorgefertigte Level zu spielen in dem er gewisse Gegenstände bekommt und diese in einer bestimmten Zeit in dem Raum platzieren muss. Außerdem kann man sich mit seinen Freunden in spannenden Highscorebattles messen. Eine weitere Funktion des Spiels ist der Editier-Modus. In diesem Modus hat der Nutzer die Möglichkeit, seinen eigenen Raum zu erstellen und die dazugehörigen Objekte auszuwählen, mit denen ein Spieler den Raum füllen muss. Spielergebnisse sollen in Highscores festegehalten werden. Räume, Highscores und Benutzerprofile sollen persistent gespeichert werden können.

1.2 Qualitätsziele

Qualitätsziel	Szenario
Interaktives Erstellen, Bearbeiten und Löschen von Räumen / Grundrissen	User Startet Anwendung und zieht ein paar Wände im Editor, um sich einen Raum zu erstellen. Nachdem er ein paar Wände löscht und erneut platziert speichert er das Level ab.
User soll auf verschiedene Fehleingaben hingewiesen werden	User platziert Wände um eigenes Level zu erstellen und versucht abzuspeichern. Dies funktioniert leider nicht, da der Grundriss an manchen Stellen Lücken enthält. Nach dem Hinweis des Programms zieht User noch fehlende Wände und speichert erfolgreich ab. Anschließend spielt User eine Runde auf neuem Level. Da er zu lange zum platzieren eines Gegenstands braucht, bekommt er eine Punktestrafe. Nach dreimaliger Fehlplatzierung ist das Level schließlich vorbei und der Nutzer muss von vorne anfangen.
Alle Aktionen und Regeln sollen leicht und verständlich erklärt werden.	Vor dem ersten Spielen oder Erstellen eines Raumes wird dem User eine kurze Anleitung mit Erläuterung der Elemente angezeigt. Bei einer Fehlplatzierung oder Fehleingabe bekommt er eine visuelle und optional eine akustische Rückmeldung.
Oberfläche ist kindgerecht und motiviert zum Spielen.	Der User erkennt schnell, welche Buttons er wählen muss, um zum gewünschten Ziel zu gelangen. Durch Farbe und Rückmeldung der Elemente kann er spielen, ohne viel zu überlegen und verliert den Spaß nicht.

1.3 Stakeholder

Rolle	Kontakt	Erwartungshaltung
Nutzer/Kind	*spielt das Spiel*	*Erwartet vom System, dass seine Aktionen verstanden werden und das System einfach und schnell die Anwendung und Regeln erklärt*
Administrator/Entwickler	*wartet das Spiel*	*Erwartet, dass Benutzereingaben nicht fehlinterpretiert werden und Benutzer keine Aktionen durchführen kann, welche vom Entwicklerteam nicht vorgesehen sind.*

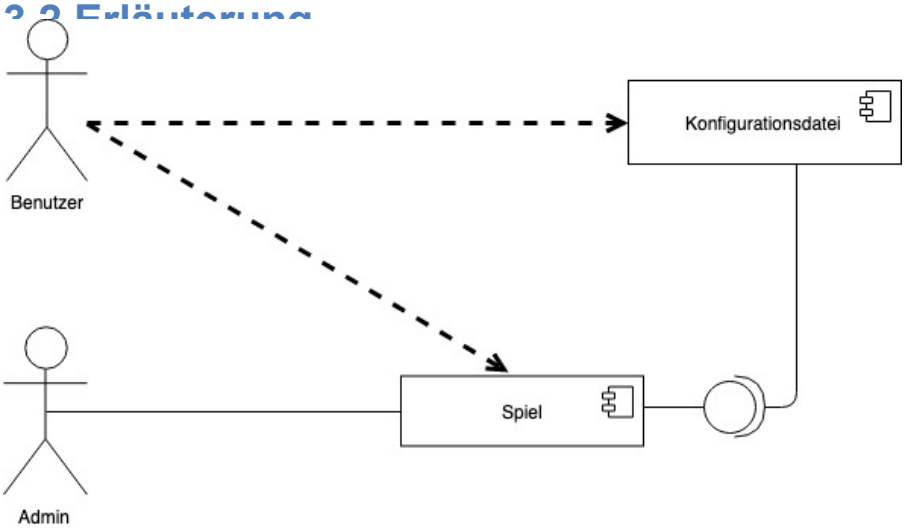
2. Randbedingungen

Randbedingung	**Erläuterung**
Räume welche im “Level-Modus” gespielt werden können, können lediglich von Entwicklern hinzugefügt/bearbeitet werden.	Hierfür muss Entwickler Programm über Konsole mit Username und Password aufrufen. Anschließend können Level-Modus Räume im Editor bearbeiten.
Highscore-Listen dürfen nicht manipulierbar sein.	Durch einen Vermerk in der Datei, die die Highscores beinhaltet, soll sichergestellt werden, dass die Datei aus dem Spiel heraus korrekt erstellt wurde. Verändert ein User über das Dateisystem einen Eintrag in der Datei, soll dies im Spiel erkannt werden.

3. Kontextabgrenzung

3.1 Fachlicher Kontext

3.2 Erläuterung

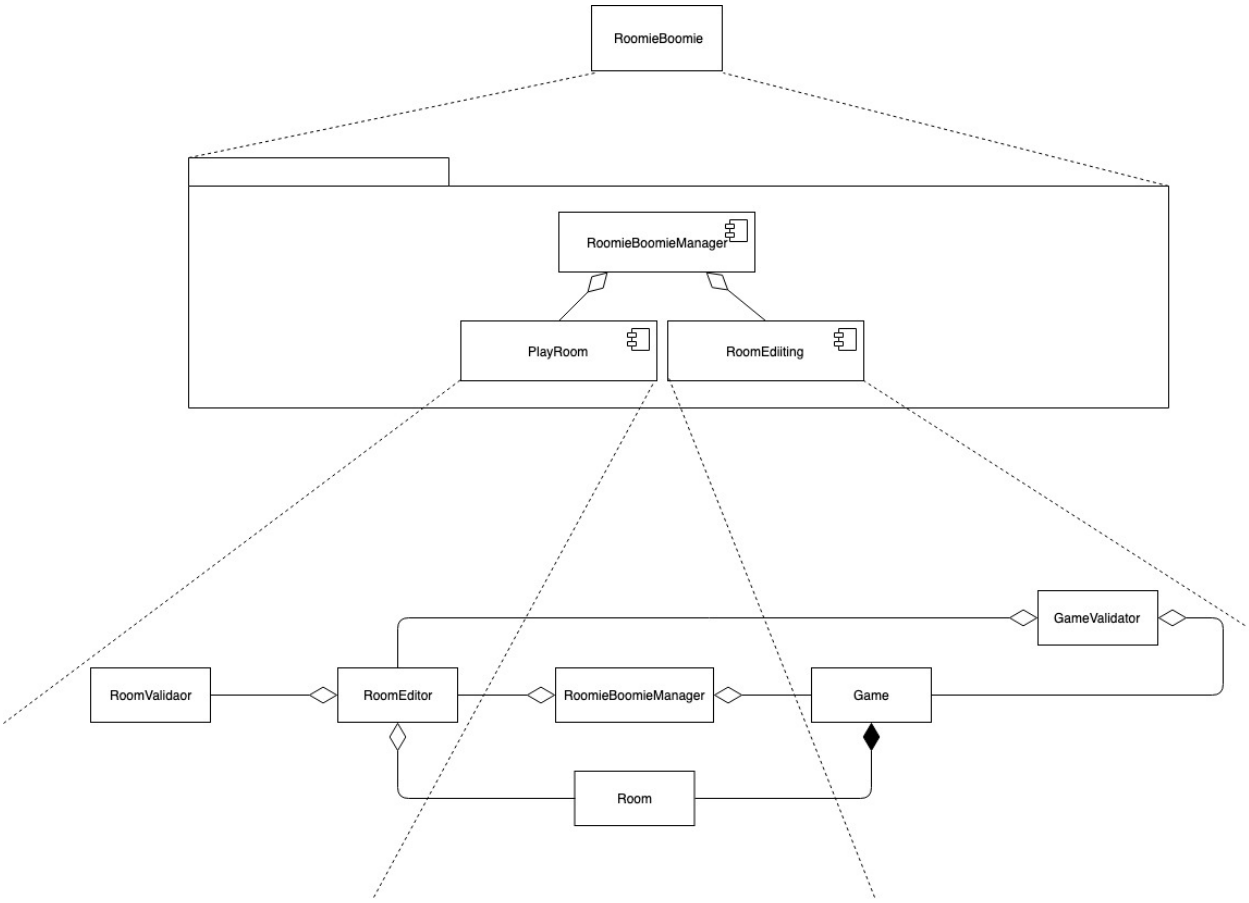


Der Administrator verfügt über alle Konfigurationsdateien, in welchen alle existierenden Gegenstände, alle erstellten Level sowie alle Highscore-Listen enthalten sind. Um diese zu ändern benötigt man einen Texteditor sowie Administrator Zugriffsrechte in der Programmkonsole.

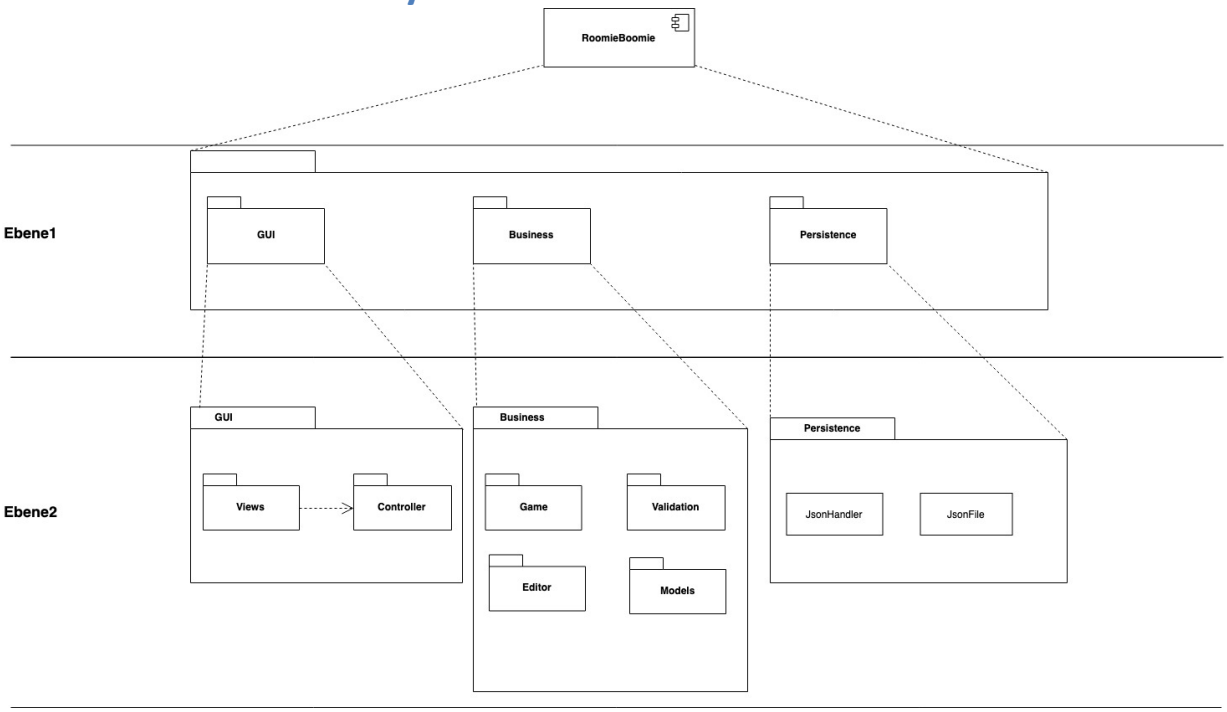
4. Lösungsstrategie

Die Software soll in Java 8 mit JavaFX umgesetzt werden. Persistente Speicherung von Daten soll mithilfe von JSON-Dateien auf dem lokalen Rechner umgesetzt werden. Für das Parsen von JSON soll JSON-P verwendet werden.

5. Bausteinsicht


RoomieBoomie
RoomieBoomieManager
PlayRoom
RoomEdiiting
GameValidator
RoomValidaor
RoomEditor
RoomieBoomieManager
Game
Room

5.1 Whitebox Gesamtsystem



Begründung
Die Software RoomieBoomie ist gemäß der Drei-Schichten-Architektur in GUI-, Business- und Persistence-Schicht eingeteilt.

Enthaltene Bausteine
GUI, Business, Persistence

5.1.1 Ebene 1

GUI
Der Baustein GUI enthält View und Controller, um die Eingaben des Anwenders an die Geschäftslogik weiterzureichen. Hier erfolgt die Trennung gemäß MVC. Der Anwender stößt hier durch Eingaben an der Oberfläche die Programmlogik an.

Business
Business enthält die Geschäftslogik. Welche Methoden durch welche Benutzerangaben ausgeführt werden, entscheidet der Controller im GUI-Baustein.

Persistence
Persistence bildet die Schnittstelle zum Dateisystem und dient der persistenten Speicherung. Genutzt wird hier das JSON-Format

5.1.2 Ebene 2

GUI

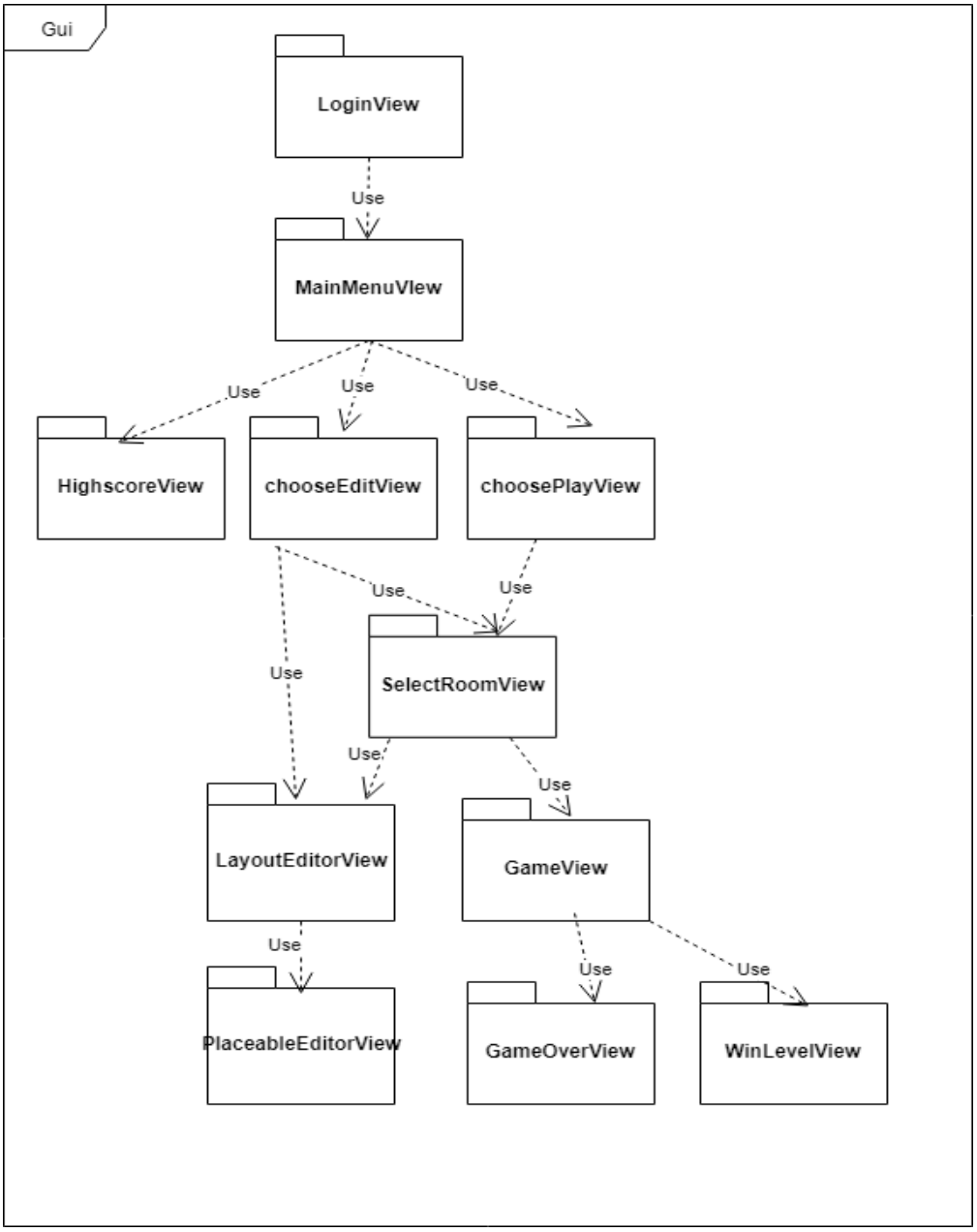
Die View enthält die Benutzungsoberfläche ohne Logik. Im Controller wird den Elementen der GUI Aktionen zugeordnet. Die Controller bilden die Schnittstelle in den Business-Baustein.

Business
Business bildet mit den Elementen Game, Validation, Editor und Models die Geschäftslogik. Die Models managen die Zusammenarbeit zwischen den Elementen. Die zwei Komponenten, mit denen der User arbeitet, sind Game, wo Levels gespielt werden können und Highscores angezeigt werden können, sowie Editor, wo neue Levels kreiert werden können. Beide Komponenten werden im Hintergrund durch Validation überprüft.

Persistence
Der JsonHandler ist für das persistente Speichern und Laden von Objekten und Konfiguration-Attributen zuständig und ist damit die Schnittstelle zwischen Programm und Speichersystem. Die Klasse übernimmt auch die komplette Umwandlung von Klassen in Json-Objekte, die dann im Dateisystem als. json-Dateien abgelegt werden und umgekehrt. Auch Konfigurationsdaten für das Programm werden über diese Schnittstelle eingelesen.

6. GUI Beschreibung

6.1 Views



LoginView
Dies ist die erste Ansicht die der User bekommt wenn er die Anwendung startet. Sie bietet ihm die Möglichkeit mit einem Namen auszuwählen und sich dann einzuloggen.

MainMenuView
Diese View repräsentiert das Hauptmenü und besitzt jeweils die Buttons Spielen, Spiel Erstellen, Tutorial und Highscore.

HighscoreView
Diese Ansicht zeigt dem User in einer sortierten Tabelle die höchsten 5 Highscore Einträge an wie auch die Namen der User die dies geschafft haben.

ChooseEditView
Diese Ansicht gibt dem User die Möglichkeit zwischen der Auswahl ‘’neuen Raum erstellen’’ oder ‘’Raum laden’’ sich zu entscheiden was er machen will.

ChoosePlayView
Diese View zeigt dem User die Buttons ‘’Level Modus’’ und ‘’Kreativ Modus’’ an, in denen der User die Möglichkeit hat im Level Modus schwierigere vorgefertige Räume freizuschalten oder im Kreativ Modus die selbsterstellten Räume zu spielen.

SelectRoomView
Diese Ansicht zeigt dem User die Auswahl der Räume an die er zum spielen auswählen kann.

LayoutEditorView
Diese View repräsentiert den ersten Teil des Raum Editors und gibt dem User die Möglichkeit aus der Auswahl LayoutObjekte einen groben Grundriss des Raums zu erstellen.

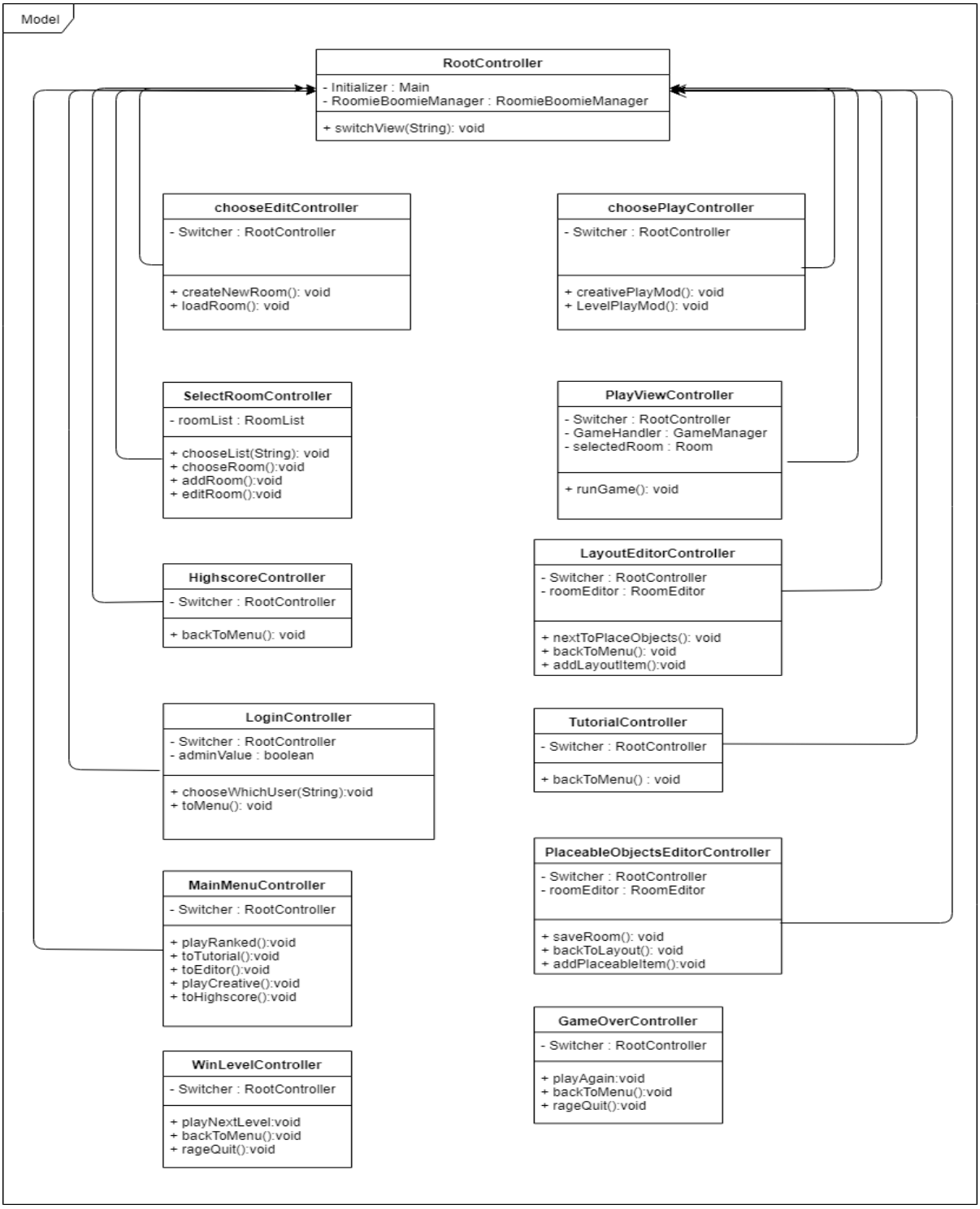
GameView
Diese View zeigt da Spiel selbst an.

PlaceableEditorView
Diese View repräsentiert den zweiten Teil des Raum Editors und gibt dem User die Möglichkeit aus der Auswahl der platzierbaren Möbel/Objekte den Raum zu füllen und zu speichern.

GameOverView
Diese Ansicht wird aufgerufen falls der User bei einem Spiel scheitert und gibt ihm die Möglichkeit das gleiche Level nochmal zu starten oder wieder zurück ins Hauptmenü zu gelangen.

WinLevelView
Diese Ansicht wird aufgerufen falls der User erfolgreich alle Objekte gerecht platzieren konnte und gibt ihm die Möglichkeit ein neues Level zu starten oder wieder zurück ins Hauptmenü zu gelangen.

6.2 Controller



RootController
Dieser Controller beinhaltet die Instanz RoomieBoomieManager, die wiederum alle notwendigen Objekte des Spiels beinhaltet und initialisiert. Außerdem ist er mit der Methode

switchView() für zuständig, zwischen den einzelnen Views zu wechseln und gibt dabei jeweils beim initialisieren der aufgerufenen Controller sich selbst und dessen notwendigen Objekte mit.

LoginController
Dieser Controller kontrolliert ob der Name der eingegeben wurde als User schon existiert und erstellt einen neuen User falls dies nicht der Fall ist. Außerdem überprüft dieser Controller ob der User der sich einloggt ein Admin ist oder ein normaler Spieler.

MainMenuController
Dieser Controller ist für die Hauptmenü des Spiels zuständig und hat jeweils die Funktionen ein Spiel zu erstellen, Ein Spiel zu spielen, das Tutorial sich anzugucken und sich die Highscore Tabelle anzuschauen.

TutorialController
Dieser Controller bietet ist jeweils nur für die Tutorial Ansicht die Funktionen an diese weiter anzugucken oder wieder zurück zum Hauptmenü zu gelangen.

HighscoreController
Dieser Controller ist jeweils für die Highscore Ansicht zuständig und zeigt einem die besten Ranking Einträge an die in diesem Spiel erreicht wurden oder wieder zurück zum Hauptmenü zu gelangen.

ChooseEditController
Dieser Controller gibt einem die Optionen einen neuen Raum zu erstellen oder einen bereits existierenden Raum zu laden und zu bearbeiten.

ChoosePlayController
Dieser Controller gibt einem die Möglichkeit die Räume unseres statischen LevelModus zu spielen oder einer der selbsterstellten Räume zu spielen welche im KreativModus sind.

LayoutEditorController
Dieser Controller ist der erste Teil des RaumEditors und gibt dem User die Möglichkeit aus den LayoutObjekten wie Wände, Türen und Fenster einen groben Grundriss zu zeichnen und validiert diese ob diese nach den Regeln gerecht platziert wurden.

PlaceableObejectsEditorController
Dieser Controller repräsentiert den zweiten Teil des RaumEditors und gibt dem User die Möglichkeit den Raum mit Objekten/Möbel zu befüllen, validiert diese ob diese nach den Regeln gerecht platziert wurden und speichert das Level/ den Raum nach positiver Rückgabe des Validators ab.

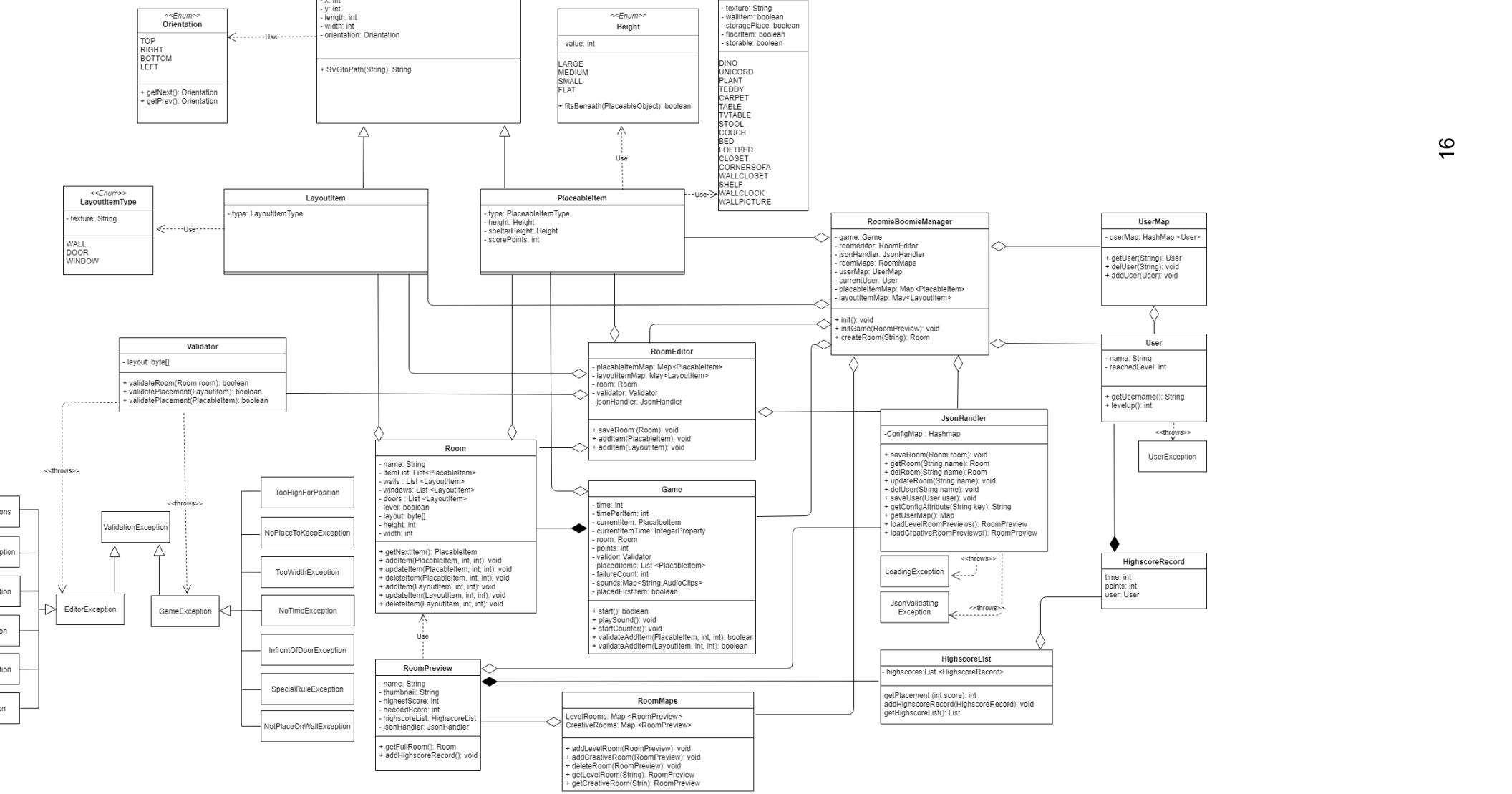
PlayViewController
Dieser Controller ist dafür zuständig die Aktionen des Users während des Spiel richtig zu interpretieren und ruft jeweils ob der User das Spiel verloren hat die looseGame() Methode auf oder wenn der User erfolgreich alle Objekte fehlerfrei platzieren konnte die Methode winLevel() auf.

GameOverController

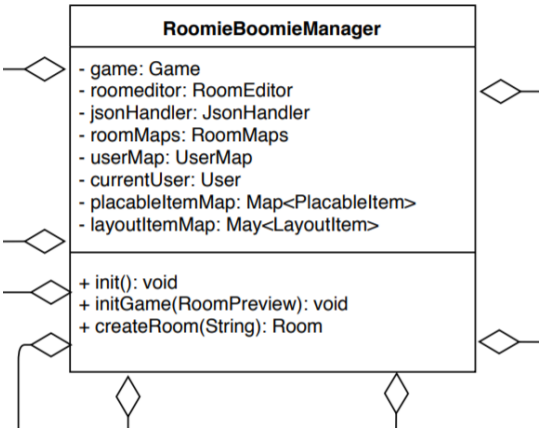
Dieser Controller gibt den User nach einem gescheiterten Spiel die Möglichkeit das Level nochmal neu zu starten oder wieder zurück ins Hauptmenü zu gelangen.

WinLevelController
Dieser Controller gibt dem User nach einem erfolgreichen abschließen des Levels die Möglichkeit das nächste Level zu spielen, von dem gespielten Level die besten Highscores zu sehen und wieder zurück ins Hauptmenü zu gelangen.

7. Business und Persistence

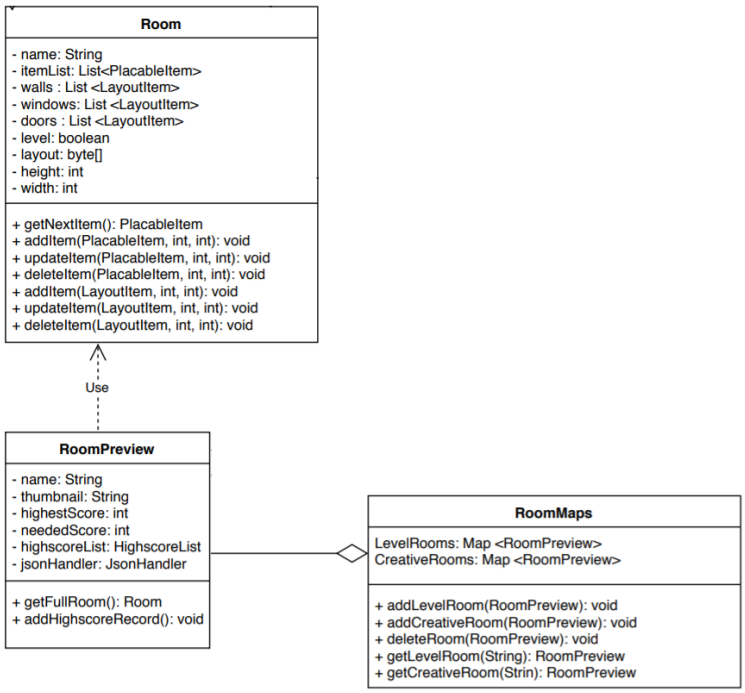

«Enum»
Orientation
TOP
RIGHT
BOTTOM
LEFT
+ getNext(): Orientation
+ getPrev(): Orientation
Use
- x: int
- y: int
- length: int
- width: int
- orientation: Orientation
+ SVGtoPath(String): String
«Enum»
Height
- value: int
LARGE
MEDIUM
SMALL
FLAT
+ fitsBeneath(PlaceableObject): boolean
- texture: String
- wallItem: boolean
- storagePlace: boolean
- floorItem: boolean
- storable: boolean
DINO
UNICORD
PLANT
TEDDY
CARPET
TABLE
TVTABLE
STOOL
COUCH
BED
LOFTBED
CLOSET
CORNERSOFA
WALLCLOSET
SHELF
WALLCLOCK
WALLPICTURE
«Enum»
LayoutItemType
- texture: String
WALL
DOOR
WINDOW
LayoutItem
- type: LayoutItemType
PlaceableItem
- type: PlaceableItemType
- height: Height
- shelterHeight: Height
- scorePoints: int
RoomieBoomieManager
- game: Game
- roomeditor: RoomEditor
- jsonHandler: JsonHandler
- roomMaps: RoomMaps
- userMap: UserMap
- currentUser: User
- placableItemMap: Map<PlacableItem>
- layoutItemMap: May<LayoutItem>
+ init(): void
+ initGame(RoomPreview): void
+ createRoom(String): Room
UserMap
- userMap: HashMap <User>
+ getUser(String): User
+ delUser(String): void
+ addUser(User): void
User
- name: String
- reachedLevel: int
+ getUsername(): String
+ levelup(): int
Validator
- layout: byte[]
+ validateRoom(Room room): boolean
+ validatePlacement(LayoutItem): boolean
+ validatePlacement(PlacableItem): boolean
RoomEditor
- placableItemMap: Map<PlacableItem>
- layoutItemMap: May<LayoutItem>
- room: Room
- validator: Validator
- jsonHandler: JsonHandler
+ saveRoom (Room): void
+ addItem(PlacableItem): void
+ addItem(LayoutItem): void
JsonHandler
-ConfigMap : Hashmap
+ saveRoom(Room room): void
+ getRoom(String name): Room
+ delRoom(String name):Room
+ updateRoom(String name): void
+ delUser(String name): void
+ saveUser(User user): void
+ getConfigAttribute(String key): String
+ getUserMap(): Map
+ loadLevelRoomPreviews(): RoomPreview
+ loadCreativeRoomPreviews(): RoomPreview
«throws»
UserException
Room
- name: String
- itemList: List<PlacableItem>
- walls : List <LayoutItem>
- windows: List <LayoutItem>
- doors : List <LayoutItem>
- level: boolean
- layout: byte[]
- height: int
- width: int
+ getNextItem(): PlacableItem
+ addItem(PlacableItem, int, int): void
+ updateItem(PlacableItem, int, int): void
+ deleteItem(PlacableItem, int, int): void
+ addItem(LayoutItem, int, int): void
+ updateItem(LayoutItem, int, int): void
+ deleteItem(LayoutItem, int, int): void
Game
- time: int
- timePerItem: int
- currentItem: PlacalbeItem
- currentItemTime: IntegerProperty
- room: Room
- points: int
- validor: Validator
- placedItems: List <PlacableItem>
- failureCount: int
- sounds:Map<String,AudioClips>
- placedFirstItem: boolean
+ start(): boolean
+ playSound(): void
+ startCounter(): void
+ validateAddItem(PlacableItem, int, int): boolean
+ validateAddItem(LayoutItem, int, int): boolean
LoadingException
JsonValidating Exception
HighscoreRecord
time: int
points: int
user: User
HighscoreList
- highscores:List <HighscoreRecord>
getPlacement (int score): int
addHighscoreRecord(HighscoreRecord): void
getHighscoreList(): List
«throws»
ValidationException
EditorException
GameException
TooHighForPosition
NoPlaceToKeepException
TooWidthException
NoTimeException
InfrontOfDoorException
SpecialRuleException
NotPlaceOnWallException
Use
RoomPreview
- name: String
- thumbnail: String
- highestScore: int
- neededScore: int
- highscoreList: HighscoreList
- jsonHandler: JsonHandler
+ getFullRoom(): Room
+ addHighscoreRecord(): void
RoomMaps
LevelRooms: Map <RoomPreview>
CreativeRooms: Map <RoomPreview>
+ addLevelRoom(RoomPreview): void
+ addCreativeRoom(RoomPreview): void
+ deleteRoom(RoomPreview): void
+ getLevelRoom(String): RoomPreview
+ getCreativeRoom(Strin): RoomPreview

Business



RoomieBoomieManager

Diese Klasse ist die zentrale Verwaltung für alle Klassen im Model. Hier werden die Komponenten erstellt und zentrale Elemente wie die Maps verwaltet und referenziert. Der Manager kann Räume aus den RoomPreviews erstellen und damit Spiele oder den Editor starten. Der aktuelle User wird auch hier gehalten.

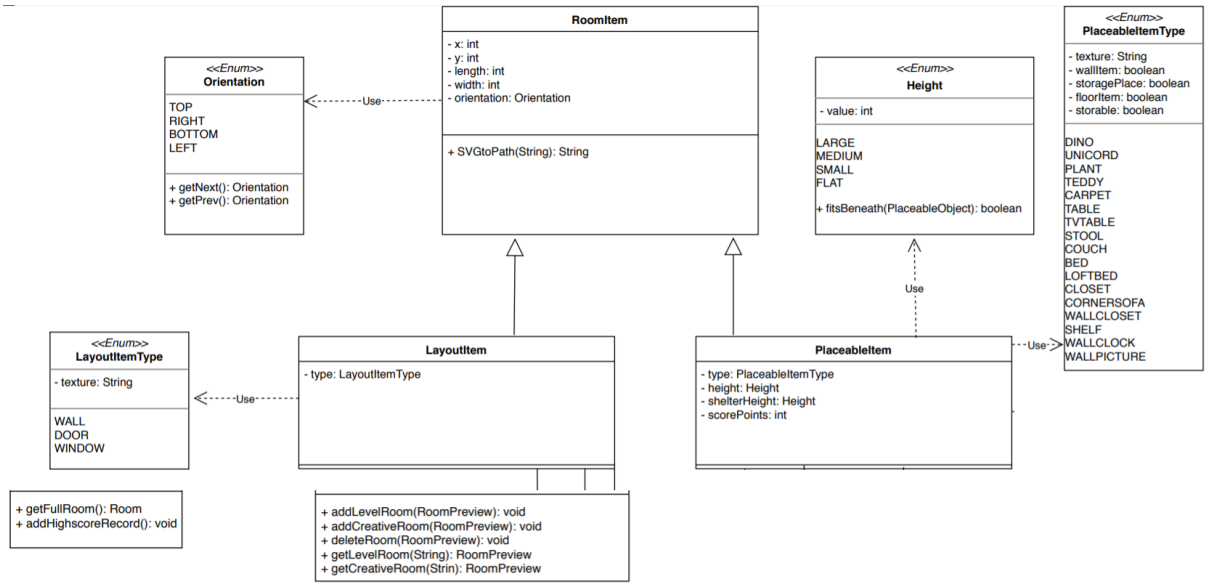


Room

Room-Objekte stellen sowohl für den Editor als auch für ein Spiel den entsprechenden Raum bereit. Alle Daten zum Grundriss sowie eine Liste von Items, die beim Spielen in diesem Raum platziert werden sollen, können gelesen und geschrieben werden.

RoomPreview
RoomPreviews sind Vorstufen von Room. Ein derartiges Objekt beinhaltet alle Informationen, die zum Anzeigen des Raumes in der Raumauswahl-Ansicht benötigt werden. Über die Methode getFullRoom() kann ein vollständiges Room-Objekt erstellt werden.
Dabei werden mit Hilfe eines JsonHandlers die Informationen über den Grundriss und die beinhalteten Objekte ausgelesen und damit ein Room-Objekt konstruiert. Da dieses Vorgehen voraussichtlich sehr ressourcenlastig sein wird und nur beim konkreten Spielen oder Bearbeiten eines Raumes benötigt wird, wurde dieses zweistufige Modell gewählt.
Außerdem beinhaltet die Klasse schon die HighscoreList. Einträge in dieser Liste werden in RoomPreview von Game angestoßen.

RoomMaps
Die Klasse RoomMaps verwaltet unabhängig voneinander alle RoomPreviews des Level-Modus und des Kreativ-Modus. Die RoomPreviews können per String geladen werden. Zum Speichern wird nur das Objekt übergeben und dieses unter dem Namen im Attribut "name" in der Map abgelegt.



RoomItem
Dies ist die Oberklasse von PlacableItem und LayoutItem. Sie stellt Informationen über Position, Größe und Richtung (Enum Orientation) eines Raumobjektes bereit. Über SVGtoPath() kann das Bild für die Items abgefragt werden.

Orientation
Das Enum Orientation beinhaltet die Drehrichtungen TOP, RIGHT, BOTTOM und LEFT.

Über getNext() und getPrev() kann die nächste oder vorherige Richtung zurückgegeben werden, wenn ein Item gedreht werden soll.

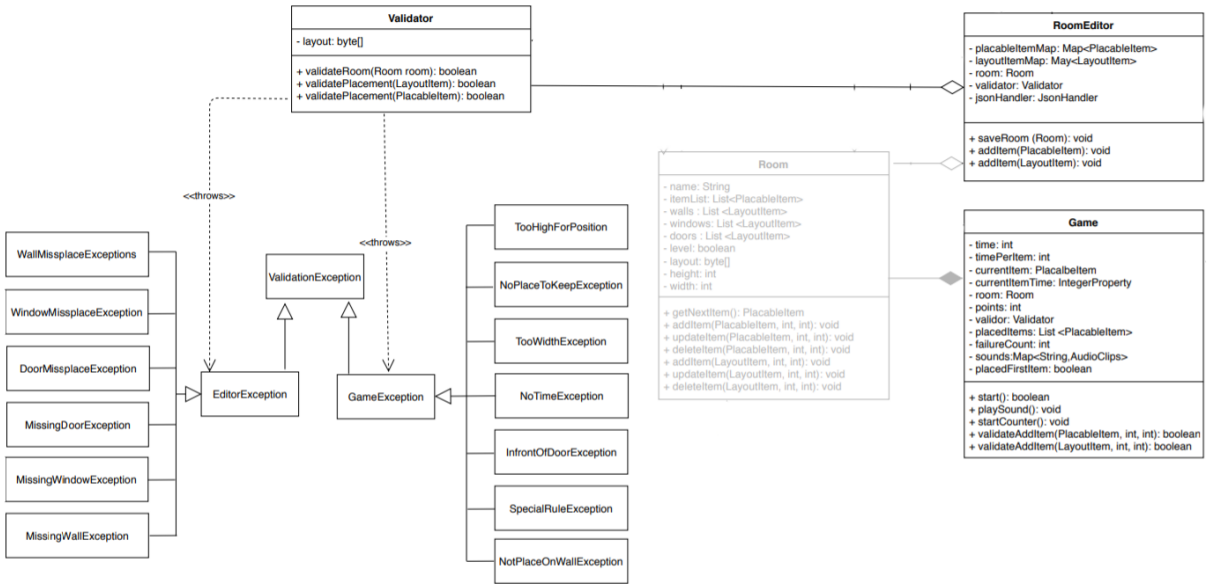
PlacalbeItem
Diese Items stellen Gegenstände dar, die in einem Raum beim Spielen platziert werden können. Ihr Typ wird über das Enum PlacableItemType beschrieben. Anders als LayoutItems haben diese Objekte noch vom Enum-Typ Height eine Höhe (height) und Unterstellhöhe (shelterHeight) sowie Spielpunkte (scorePoints).

Height
Dieses Enum verwaltet die Höhen und Unterstellhöhen von Items. Über die Methode fitsBeneath() kann abgefragt werden, ob die Höhe eines Items unter die Unterstellhöhe eines anderen passt.

PlacableItemType
In diesem Enum sind alle Typen von PlavableItems aufgelistet. Es kann die Textur zurückgeben. Außerdem sind Boolea-Werte zu folgenden Eigenschaften abfragbar:
wallItem (Item nur an Wand platzierbar),
floorItem (Item nur auf Boden/Teppich platzierbar),
storagePlace (Item bietet eine Ablage),
storable (Item kann auf anderem abgelegt werden).

LayoutItem
LayoutItems sind Elemente, die den Grundriss eines Raums darstellen. Dazu gehören Wände, Türe und Fenster. Die Typen sind im Attribut type festgehalten.

LayoutItemType
Das Enum LayoutItemType verwaltet die Typen von LayoutItems. Die Typen sind für die Validierung notwendig. Außerdem kann die Textur abgefragt werden.



Game

Game verwaltet Spielrunden mit einem spezifischen Room. Ein Spiel wird vom RoomieBoomieManager gestartet. Die Methode startCounter() überwacht die Zeit, die ein User zum Platzieren eines Gegenstandes über addItem() braucht. Zur Validierung der Platzierung hält ein Game einen GameValidator. Je nach Ergebnis der Validierung wird über einen AudioClip ein Sound abgespielt. Platzierte Items werden in einer List gespeichert, Fehlplatzierungen im Atribut failureCount. Die Methode startGame() gibt am Ende des Spiels zurück, ob der User das Level geschafft hat oder nicht.

Validator

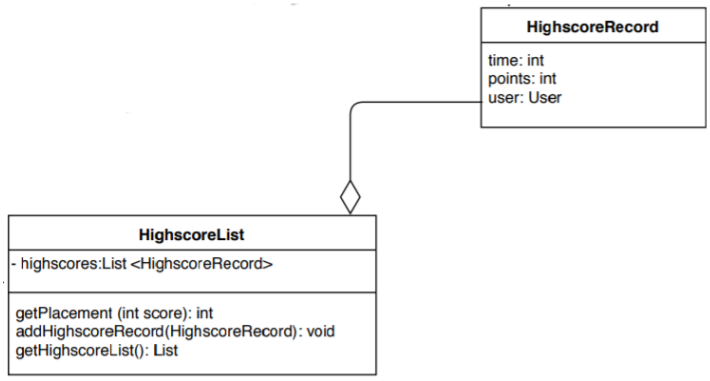
Der Validator kann sowohl im Editor als auch im Spiel validieren, ob ein Item an einer bestimmten Stelle platziert werden darf. Er hält immer eine Referenz auf den Inhalt des aktuell bearbeiteten Raums und kann validieren, ob

- ein ganzer Raum die Voraussetzungen zum Spielen erfüllt
- ob ein Grundriss-Objekt (LayoutItem) an einer Stelle platzierbar ist
- ob ein PlacableItem im Spiel oder in der Spielvorbereitung im Editor platzierbar ist

Die Validierung erfolgt anhand des Arrays, das die Elemente beinhaltet sowie auf deren Eigenschaften, die in ihren Enums abfragbar und vergleichbar sind. Bei Fehlern werden ValidationExceptions geworfen, die den genauen Grund der Fehlplatzierung beschreiben.

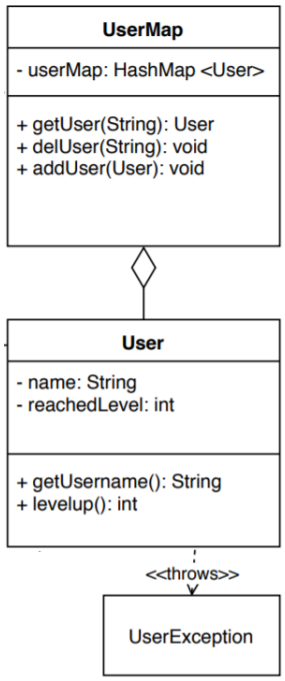
RoomEditor

Der RoomEditor übernimmt die Aufgaben zum Erstellen und Bearbeiten eines Raums im Kreativ-Modus. Er bekommt beim Initialisieren die Maps mit allen verfügbaren Items sowie einen vorhandenen oder neuen Raum. Dieser kann über die Methoden addItem() mit Items befüllt werden, wobei ein Validator die Platzierungen überprüft, bevor sie an den Raum weitergeleitet werden. Für das Speichern eines Raums verfügt diese Klasse über einen JsonHandler.



HighscoreList
Eine HighscoreList gehört immer zu genau einer RoomPreview. Sie beinhaltet eine Liste von Highscore-Einträgen des Objektes HighscoreRecord. Über getPlacement() kann abgefragt werden, auf welchem Platz es ein User mit einer bestimmte Punktanzahl geschafft hat. Highscores werden über eine RoomPreview mit addHighscoreRecord() hinzugefügt.

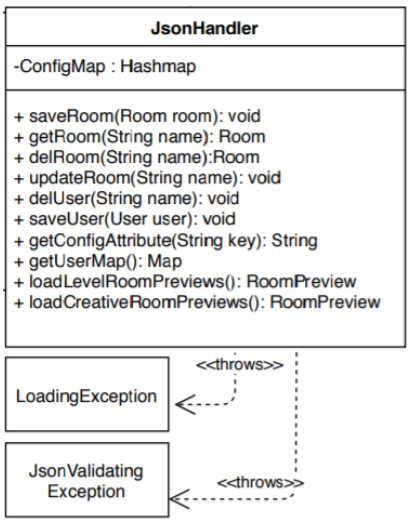
HighscoreRecord
Ein HighscoreRecord ist ein Eintrag in einer Highscore-Liste. Es beinhaltet die benötigte Zeit, den User und seine Punkte.



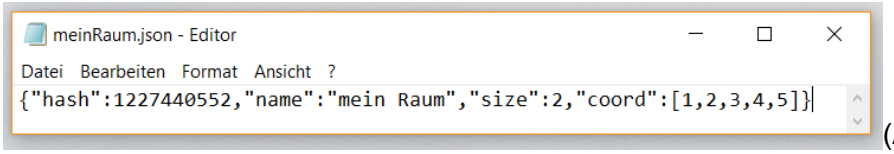
User
Profileintrag eines Users. Ein User wird zu Beginn einer Spielsitzung angelegt und in einer UserMap gespeichert. Wenn ein Name schon verhanden ist, wird eine **UserException** geworfen.

UserMap
Hier werden die User unter einem String verwaltet. Sie können erstellt, gelöscht und ausgegeben werden.

7.2 Persistence



JsonHandler
Der JsonHandler ist für das persistente Speichern und Laden von Objekten und Konfiguration-Attributen zuständig und ist damit die Schnittstelle zwischen Programm und Speichersystem. Die Klasse übernimmt auch die komplette Umwandlung von Klassen in Json-Objekte, die dann im Dateisystem als. json-Dateien abgelegt werden und umgekehrt.
Die Umwandlung der Elemente User, Room und HighscoreList soll aufgrund von Übersichtlichkeit und Zentralisierung nicht in jeder Klasse eigenständig durchgeführt werden. Stattdessen erhält der Handler einfach nur das zu speichernde Objekt über eine jeweils der Klasse zugeschnittene Methode. Dort werden die relevanten Attribute gesichert und im JSON-Format gespeichert. Beim Laden werden die Dateien über den Dateinamen angesprochen. Daher muss bei den get()-Methoden nur der Name des Objekts als String übergeben werden. Der Handler gibt dann das fertig konfigurierte Objekt zurück.
Beim Speichern generiert der Handler zusätzlich noch einen Hash-Wert, der eine Manipulation der gespeicherten Dateien ausschließt. Sollte beim Laden der Wert nicht mit den enthaltenen Informationen übereinstimmen, wird eine JsonValidatingException geworfen. Fehler beim Laden werden über eine LoadingException signalisiert.
Die Trennung in User, Room, Highscore und Konfiguration erfolgt durch Ordner im Dateisystem. Bei Initialisierung des Programms liest der Handler schon die Konfigurations-Attribute ein, die dann über getConfigAttribute() mit einem String abzufragen sind.

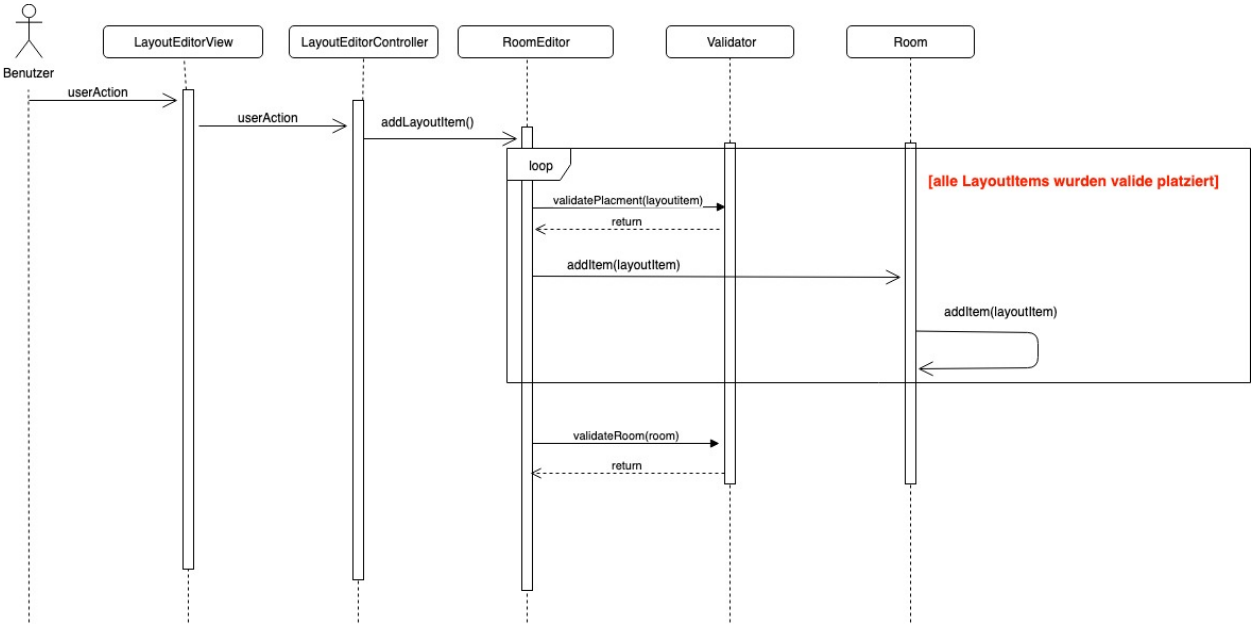


(Abgespecktes Beispiel**)**

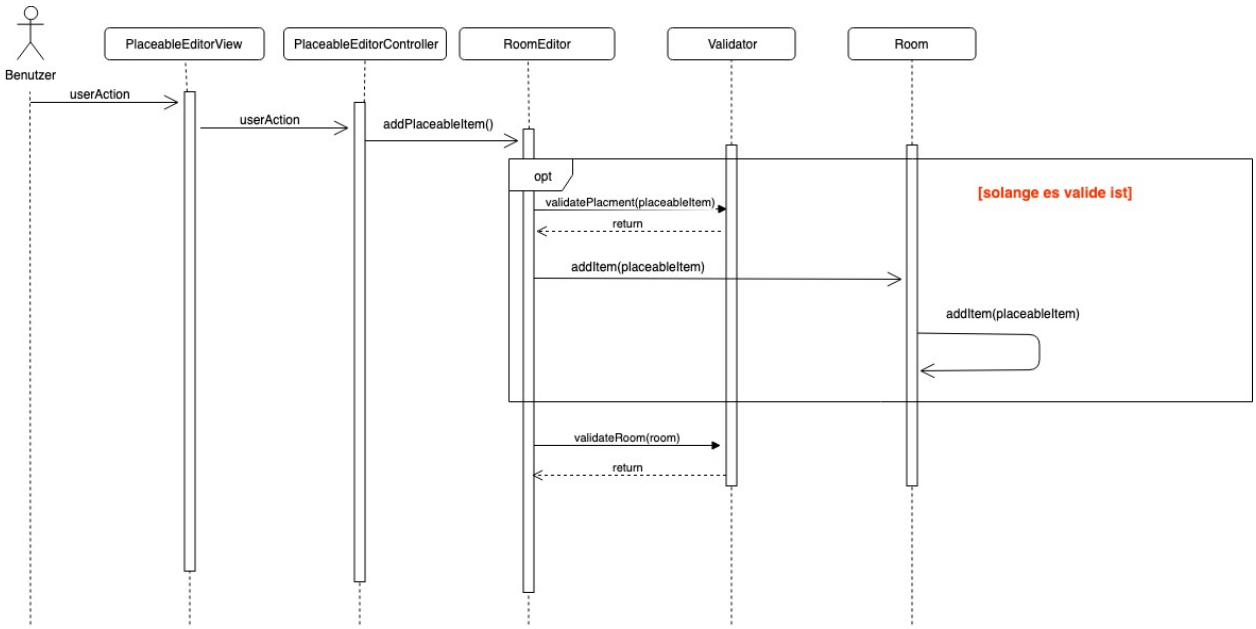
JSON-File
Über ein deartiges JSON-File werden die zu speichernden Elemente abgelegt.

8. Laufzeitsicht
Im Folgenden werden verschiedene Szenarien und das Zusammenspiel der einzelnen Pakete und Klassen in diesen Szenarien anhand von UML-Sequenzdiagrammen erläutert.

8.1 Create Room

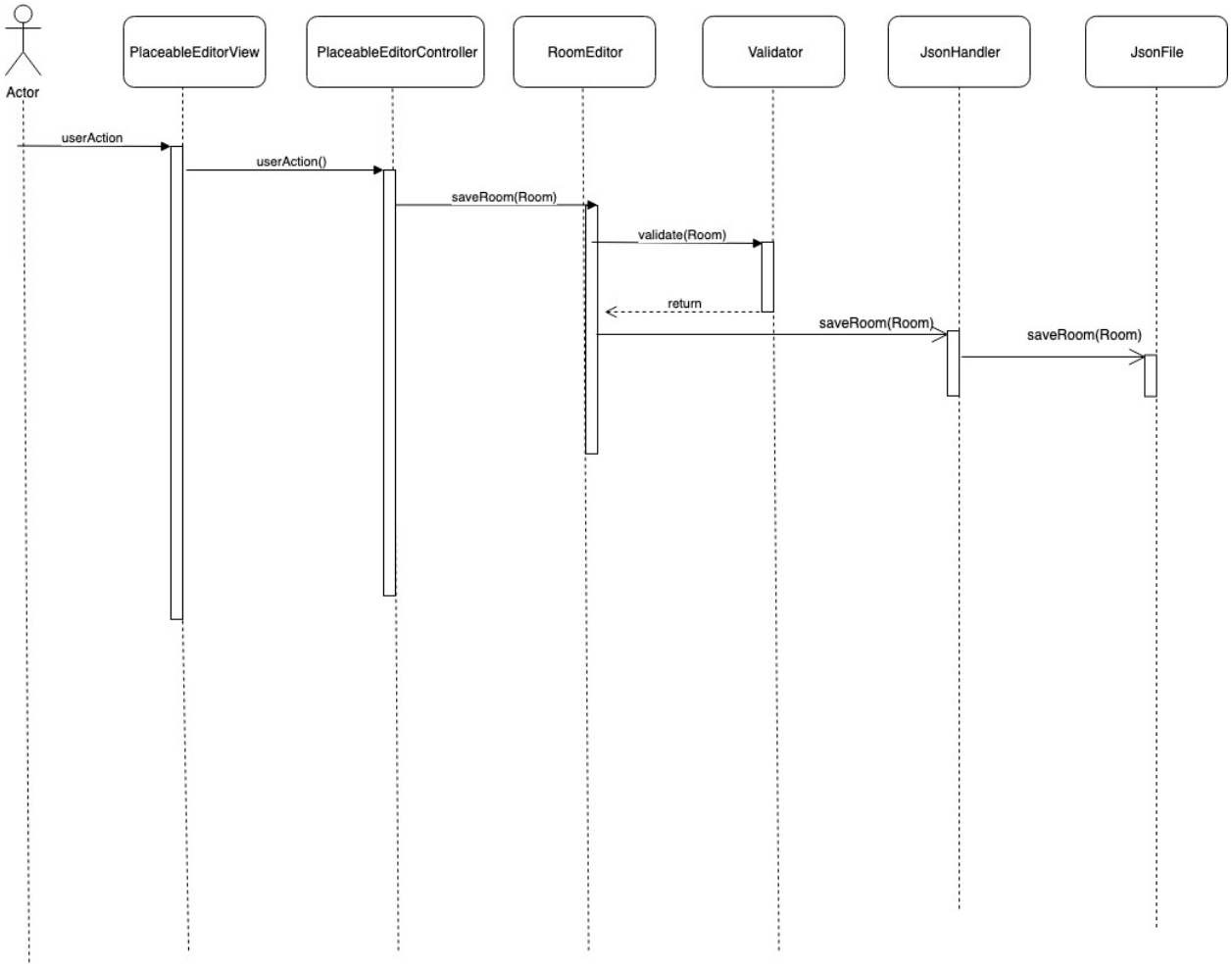


Wählt der User im Menü die Option Raum erstellen aus, so wird er in die LayouEditorView weitergeleitet und hat die Möglichkeit die Grundstruktur des Raumes zu erstellen. Dabei checkt der Validator jedes mal ob das Objekt regelgemäß platziert wurde. Wenn das der Fall ist wird das Objekt in den Raum hinzugefügt und wenn das Objekt nicht regelkonform platziert wurde dann wird eine Exception geworfen. Es wird ebenso ein passender Sound abgespielt wenn Regeln missachtet werden. Um einen Raum zu erstellen muss man alle LayoutItems vollständig und regelkonform platziert haben.



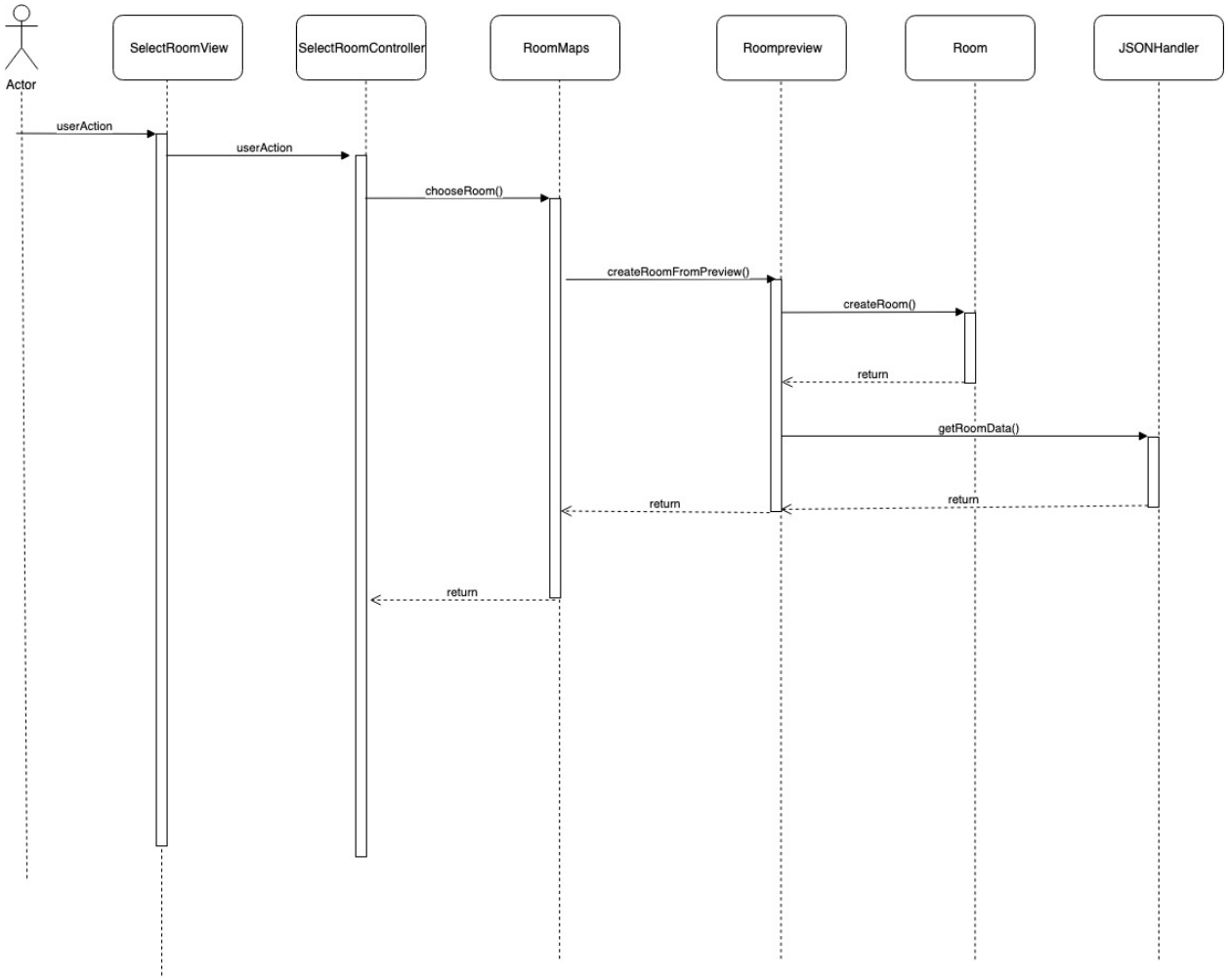
Nach dem erfolgreichen Platzieren der LayoutItems wird der User auf die PlaceableEditorView weitergeleitet und kann dort PlaceableItems setzen. Dabei werden ebenso die Platzierungen des Users überprüft und im Optimalfall übernommen. Der User kann darin so viele Objekte platzieren solange die Objekte valide platziert sind.

8.2 Save Room



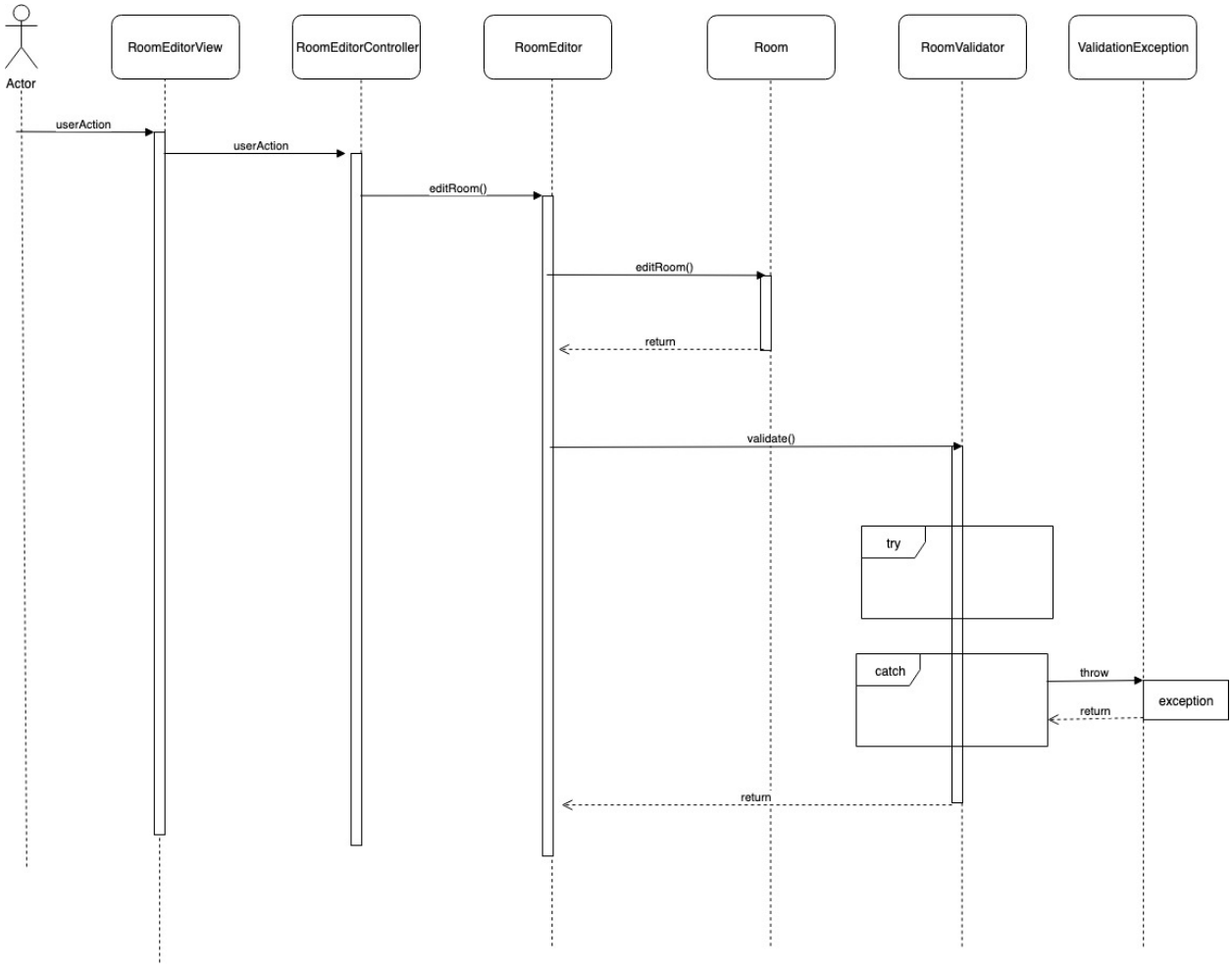
Der User platziert alle erwünschten Items und speichert nun über Button in der View. Diese leitet Input an Controller weiter welcher nun Methode saveRoom der Klasse Roomeditor aufruft. Der Raum wird zuerst über Validator.validate validiert und falls keine Validierungsfehler auftreten anschließend über JsonHandler in einer JsonFile gespeichert.

8.3 Choose Room



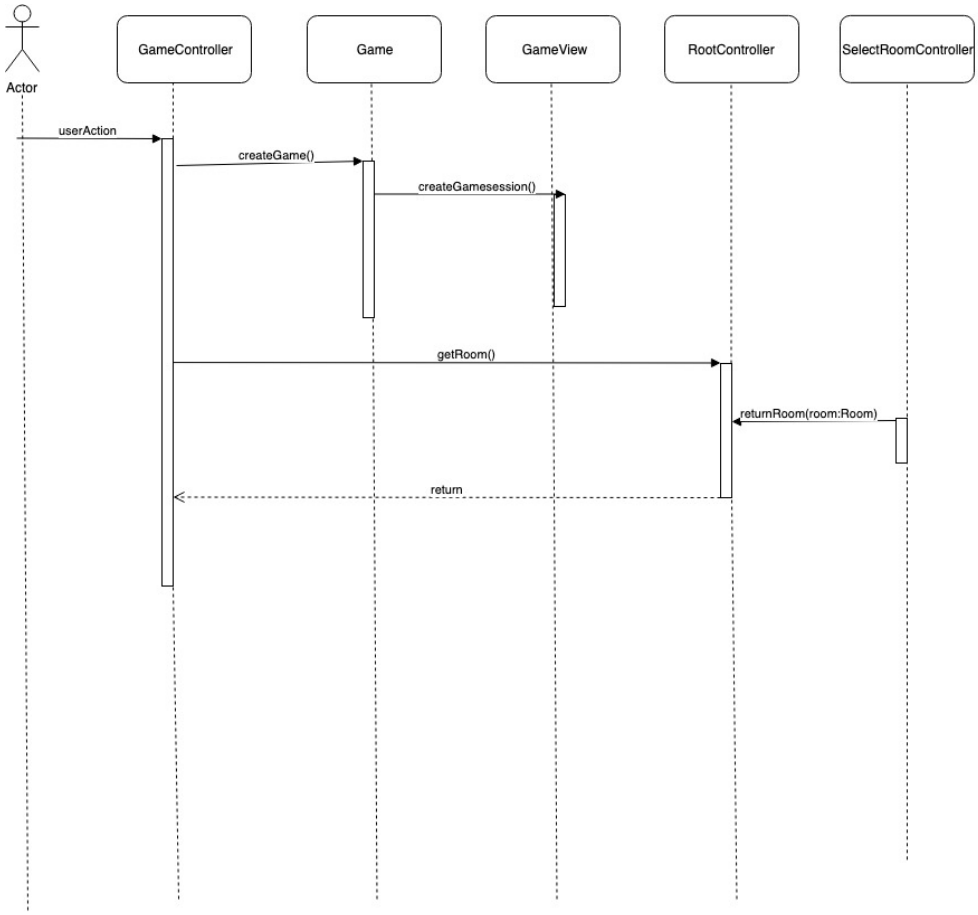
User wählt einen Raum aus und klickt ihn an. Methode zum erstellen eines benutzbaren Raumes, von angeklickter RoomPreview wird angestoßen. Hierfür benutzt Roompreview Jsonhandler um gewünschten Raum vollständig zu erstellen und wieder zurückzureichen.

8.4 Edit Room



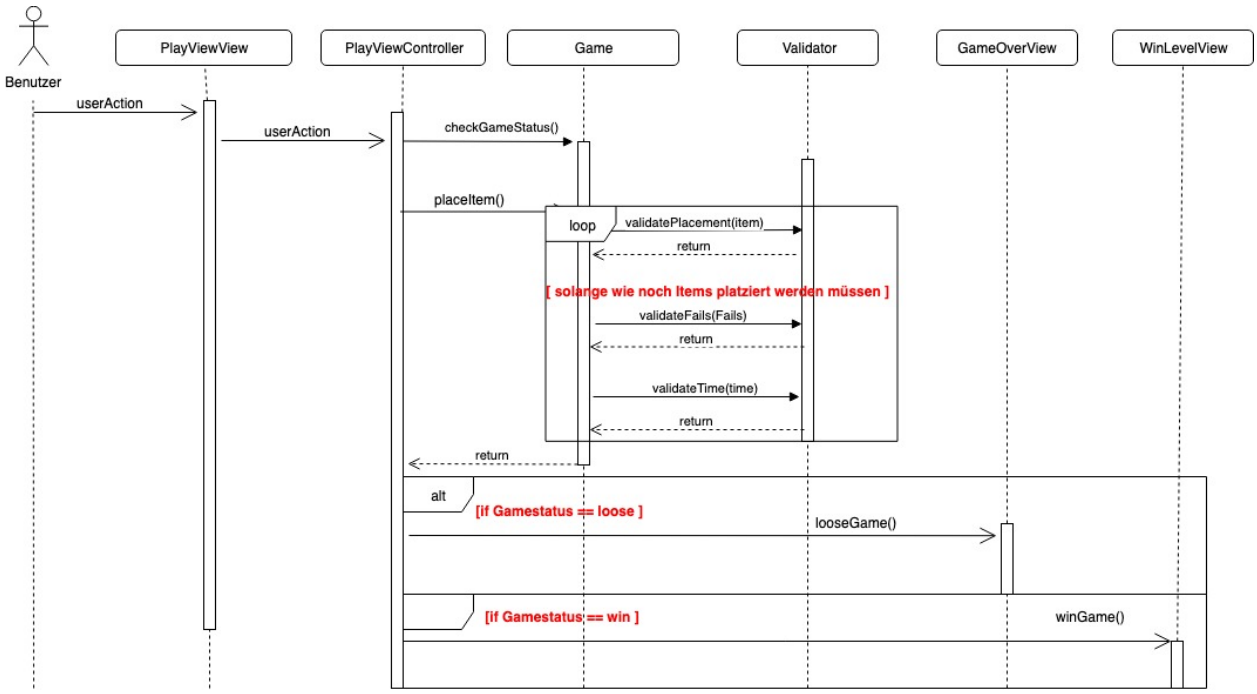
User Editiert Raum und platziert beliebig viele LayoutItems. Jedes platzierte Item wird von RoomEditor im Raum platziert und über Validator validiert. Validator returned true, oder wirft eine Exception mit dem genauen Fehler.

8.5 Create Game session



Nachdem der User einen Raum ausgewählt hat wird der Raum an den RootController weitergegeben und schlussendlich dem GameController übergeben damit dieser aus den Attributen des Raumes eine SpielSession erstellen kann.

8.6 Place objects in Game



User platziert Gegenstand im Spiel. Eingabe wird zuerst über Validator validiert und falls dieser false returned ist das Spiel vorbei und die GameOverView wird angezeigt. Returned er true gab es keinen Regelverstoß und das Spiel geht weiter.

9. Qualitätsanforderungen

Die Qualitätsanforderungen entsprechen den funktionalen (Punkt 5) und nicht-funktionalen Anforderungen (Punkt 8) aus dem Pflichtenheft, Version 1.1.