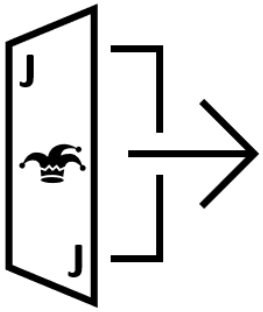
CPSC 2720 FALL 2021

Life2.0-Logout



Joker

Albert Arrieta, Ayobami Abejide, Hana Kim, Kurt Clarke

2021-10-27

TABLE OF CONTENTS

INTRODUCTION

Simple Version

1) You are a beta tester for a new CAPTCHA system involving terminal commands.
2) This CAPTCHA system involves you passing several tests that will test out how human you are.
3) If you pass enough of the tests then you win and are allowed to logout of the game, or you will be able to after we fully release the new system improvements.
4) Your questions, comments, and concerns about the new improvements are greatly appreciated!!!

Fun Version

Dear valued Life2.0 User,

We have recently discovered an exploit using our new terminal interface which viruses have been exploiting to force user logouts. We deeply apologize to those that have been affected via forced logout or by a sense of surreality. We have done our best to maintain a consistent environment, but we understand some of the recent server events have disrupted normal gameplay, we hope things will soon be back to normal.

Here at Joker Industries, we take pride in our reputation for quality and unrelenting professionalism. To prevent further attacks, we are implementing a set of sophisticated CAPTCHA's so that you and only you can decide when your Life2.0 experience ends. The new CAPTCHAs have been implemented and are undergoing Beta-test for which you have been selected for as a tester!

We would appreciate you trying out the new logout system provide us with important user data and feedback for us to improve the logout system. When you are ready, please run Life2.0-Logout from your nearest terminal, pass the CAPTCHAs, and then provide us any feedback and recommendations on your experience.

We thank you for you for playing Life2.0 and hope you continue to use our services in and out of game. Afterall no one knows you better than Joker Industries, every breath, every move, and every thought has been recorded to enhance your gameplay experience in this world of our making.

Joker Industries, truth through lies.

Sincerely,

Jack M. Napier
Head of Customer Relations
Joker Industries
10534 – Barnaby North, Gotham

PROJECT MANAGEMENT

TEAM ROLES

Team Member	**Design - Draft**	**Design – Final**	**Implementation - Basic**	**Implementation - Final**
Albert	QA Lead	QA Lead	QA Lead	QA Lead
Ayobami	Reporting Lead	Reporting Lead	Reporting Lead	Reporting Lead
Hana	Design Lead	Design Lead	Design Lead	Design Lead
Kurt	Phase Lead	Phase Lead	Phase Lead	Phase Lead

TEAM ROLE RESPONSIBILITIES

RISK MANAGEMENT

DEVEVLOPMENT PROCESS

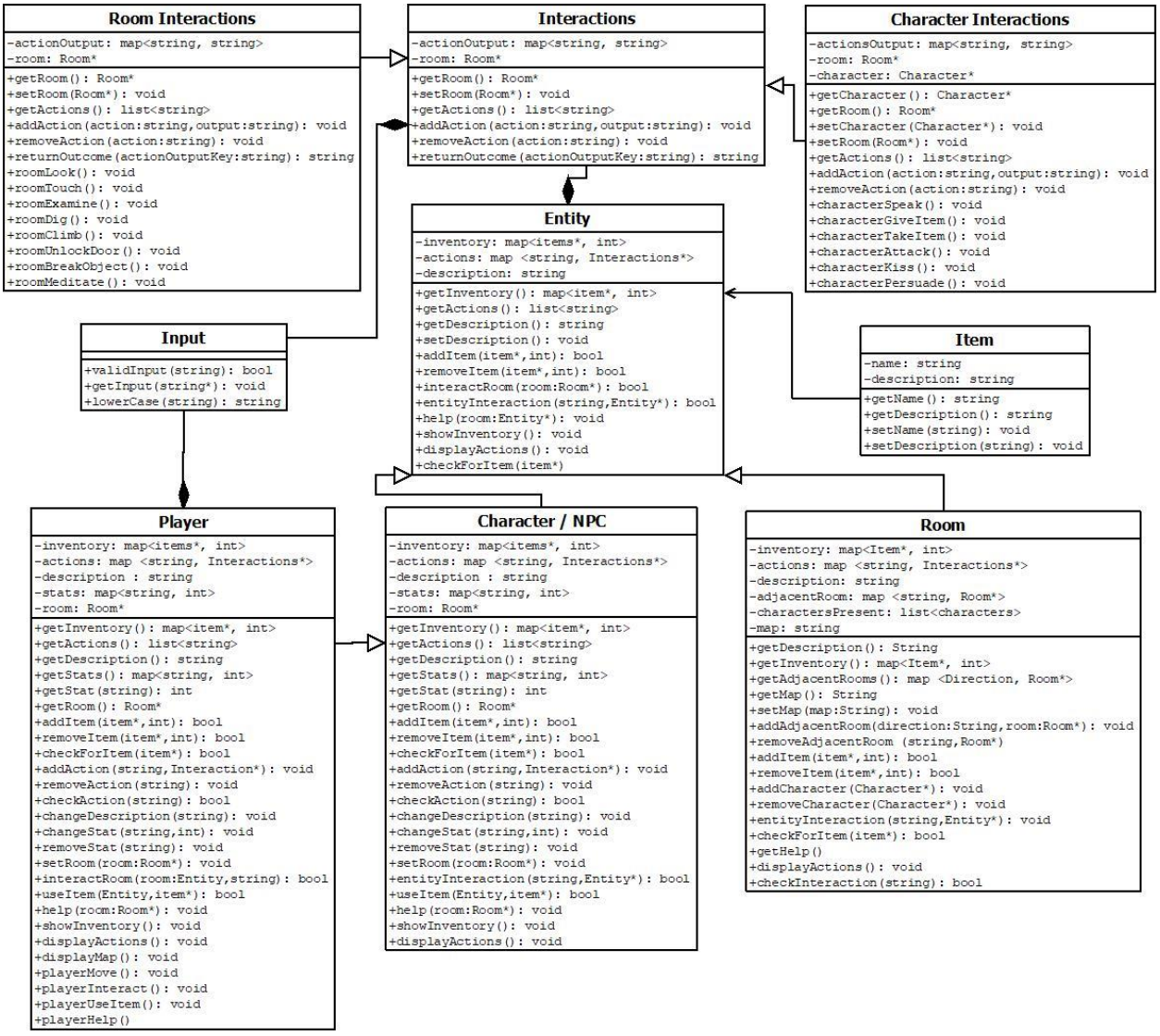
CODE REVIEW PROCESS

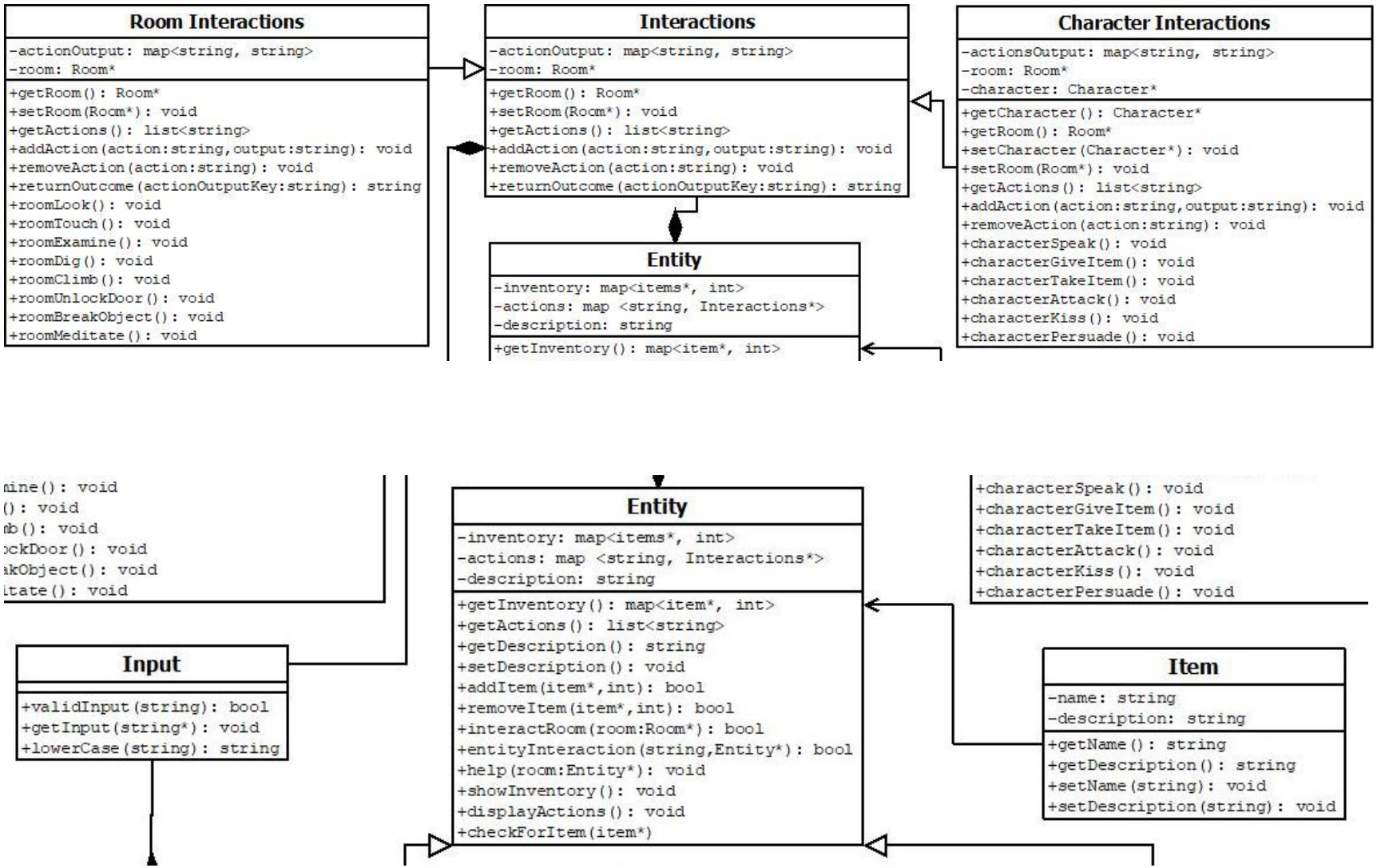
COMMUNICATION TOOLS

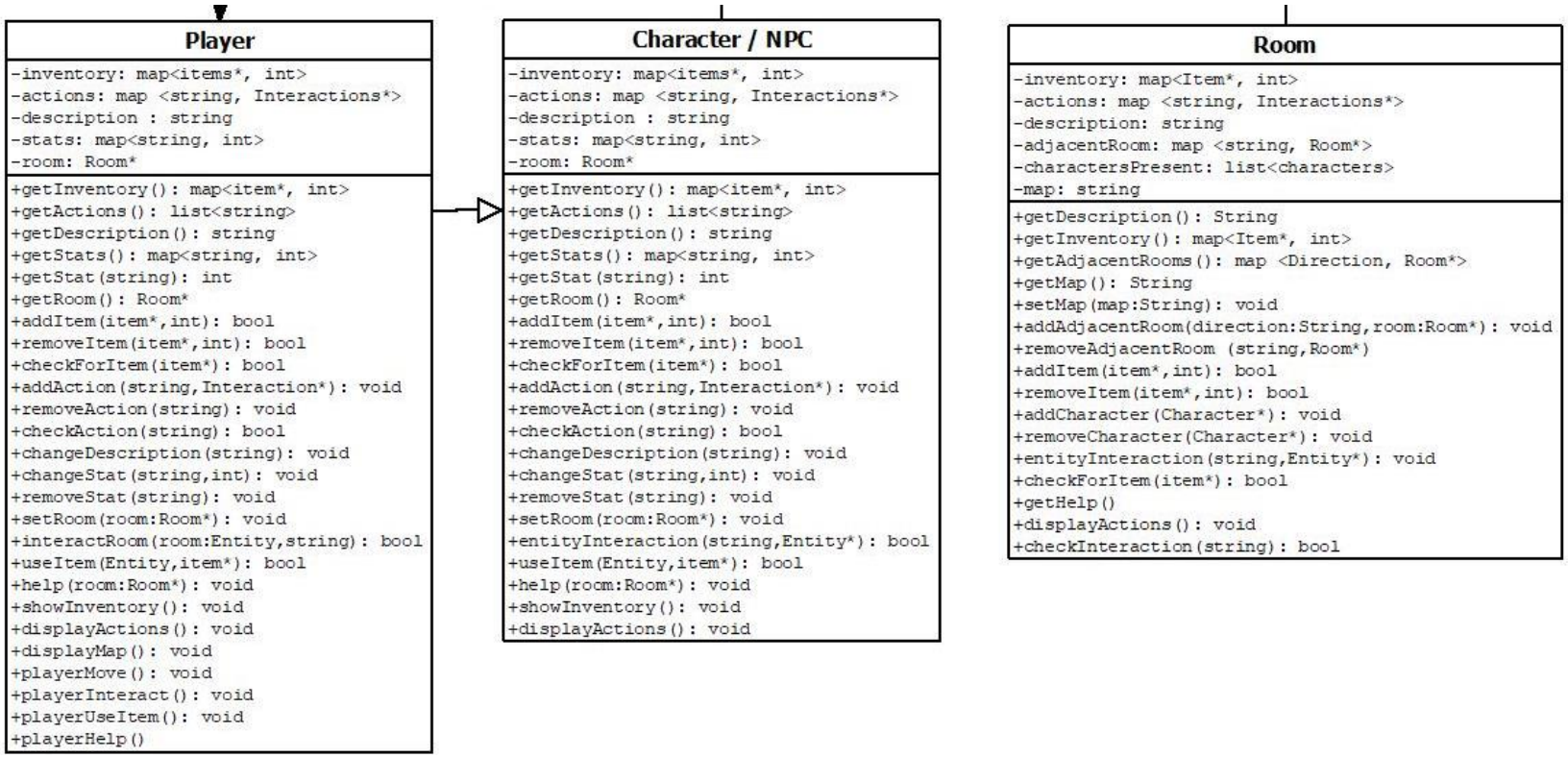
CHANGE MANAGEMENT

SOFTWARE DESIGN

DESIGN – CLASS DIAGRAMS

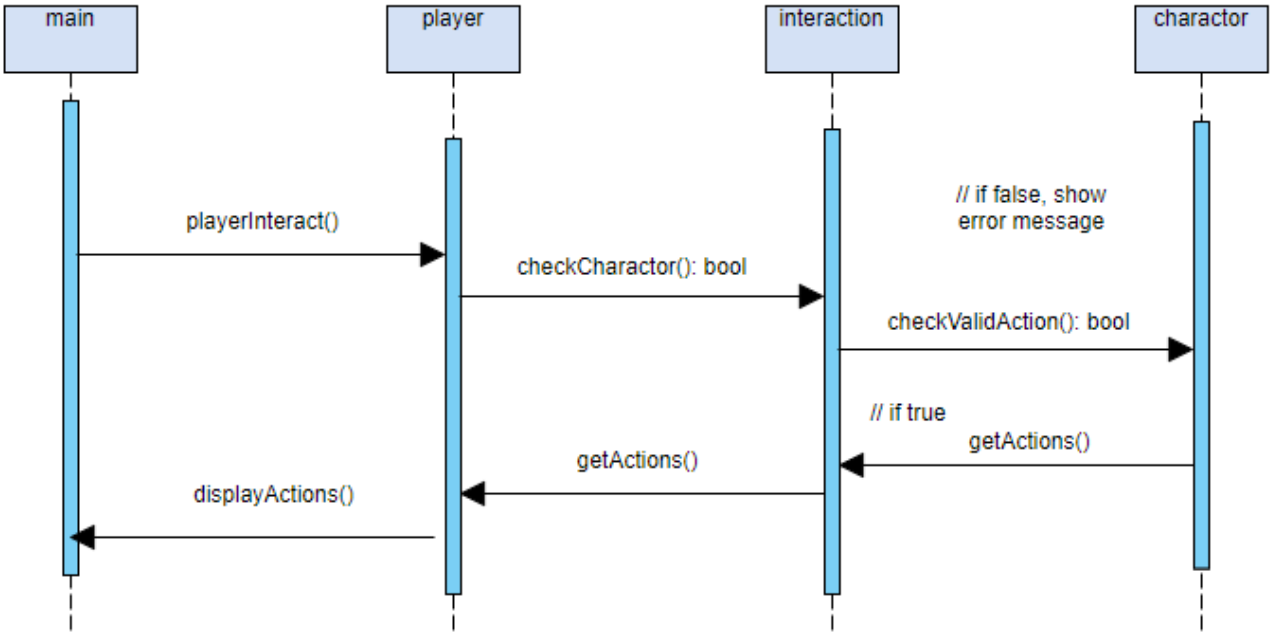



Room Interactions
-actionOutput: map<string, string>
-room: Room*
+getRoom(): Room*
+setRoom(Room*): void
+getActions(): list<string>
+addAction(action:string,output:string): void
+removeAction(action:string): void
+returnOutcome(actionOutputKey:string): string
+roomLook(): void
+roomTouch(): void
+roomExamine(): void
+roomDig(): void
+roomClimb(): void
+roomUnlockDoor(): void
+roomBreakObject(): void
+roomMeditate(): void
Interactions
-actionOutput: map<string, string>
-room: Room*
+getRoom(): Room*
+setRoom(Room*): void
+getActions(): list<string>
+addAction(action:string,output:string): void
+removeAction(action:string): void
+returnOutcome(actionOutputKey:string): string
Character Interactions
-actionsOutput: map<string, string>
-room: Room*
-character: Character*
+getCharacter(): Character*
+getRoom(): Room*
+setCharacter(Character*): void
+setRoom(Room*): void
+getActions(): list<string>
+addAction(action:string,output:string): void
+removeAction(action:string): void
+characterSpeak(): void
+characterGiveItem(): void
+characterTakeItem(): void
+characterAttack(): void
+characterKiss(): void
+characterPersuade(): void
Entity
-inventory: map<items*, int>
-actions: map <string, Interactions*>
-description: string
+getInventory(): map<item*, int>
Entity
-inventory: map<items*, int>
-actions: map <string, Interactions*>
-description: string
+getInventory(): map<item*, int>
+getActions(): list<string>
+getDescription(): string
+setDescription(): void
+addItem(item*,int): bool
+removeItem(item*,int): bool
+interactRoom(room:Room*): bool
+entityInteraction(string,Entity*): bool
+help(room:Entity*): void
+showInventory(): void
+displayActions(): void
+checkForItem(item*)
Input
+validInput(string): bool
+getInput(string*): void
+lowerCase(string): string
Item
-name: string
-description: string
+getName(): string
+getDescription(): string
+setName(string): void
+setDescription(string): void

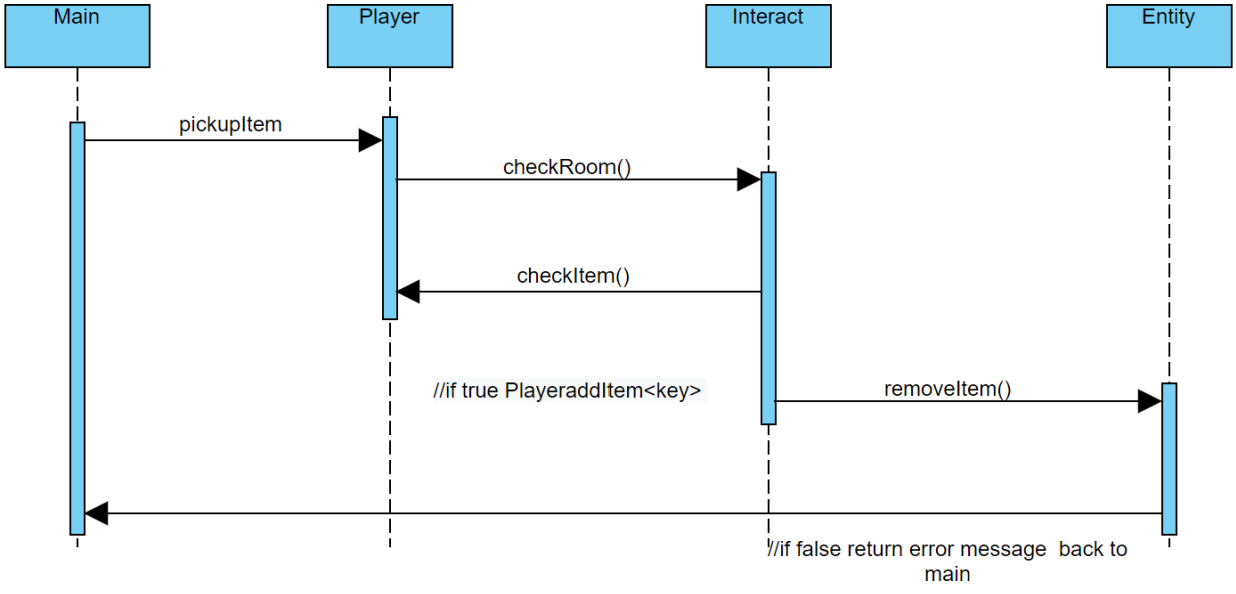

Player
-inventory: map<items*, int>
-actions: map <string, Interactions*>
-description : string
-stats: map<string, int>
-room: Room*
+getInventory(): map<item*, int>
+getActions(): list<string>
+getDescription(): string
+getStats(): map<string, int>
+getStat(string): int
+getRoom(): Room*
+addItem(item*,int): bool
+removeItem(item*,int): bool
+checkForItem(item*): bool
+addAction(string,Interaction*): void
+removeAction(string): void
+checkAction(string): bool
+changeDescription(string): void
+changeStat(string,int): void
+removeStat(string): void
+setRoom(room:Room*): void
+interactRoom(room:Entity,string): bool
+useItem(Entity,item*): bool
+help(room:Room*): void
+showInventory(): void
+displayActions(): void
+displayMap(): void
+playerMove(): void
+playerInteract(): void
+playerUseItem(): void
+playerHelp()
Character / NPC
-inventory: map<items*, int>
-actions: map <string, Interactions*>
-description : string
-stats: map<string, int>
-room: Room*
+getInventory(): map<item*, int>
+getActions(): list<string>
+getDescription(): string
+getStats(): map<string, int>
+getStat(string): int
+getRoom(): Room*
+addItem(item*,int): bool
+removeItem(item*,int): bool
+checkForItem(item*): bool
+addAction(string,Interaction*): void
+removeAction(string): void
+checkAction(string): bool
+changeDescription(string): void
+changeStat(string,int): void
+removeStat(string): void
+setRoom(room:Room*): void
+entityInteraction(string,Entity*): bool
+useItem(Entity,item*): bool
+help(room:Room*): void
+showInventory(): void
+displayActions(): void
Room
-inventory: map<Item*, int>
-actions: map <string, Interactions*>
-description: string
-adjacentRoom: map <string, Room*>
-charactersPresent: list<characters>
-map: string
+getDescription(): String
+getInventory(): map<Item*, int>
+getAdjacentRooms(): map <Direction, Room*>
+getMap(): String
+setMap(map:String): void
+addAdjacentRoom(direction:String,room:Room*): void
+removeAdjacentRoom (string,Room*)
+addItem(item*,int): bool
+removeItem(item*,int): bool
+addCharacter(Character*): void
+removeCharacter(Character*): void
+entityInteraction(string,Entity*): void
+checkForItem(item*): bool
+getHelp()
+displayActions(): void
+checkInteraction(string): bool

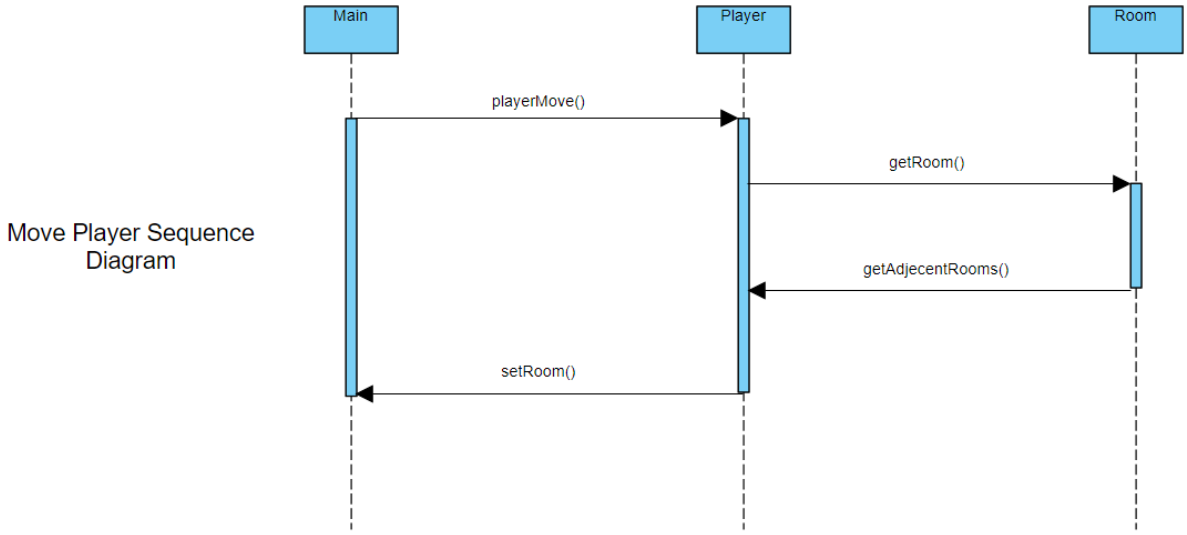
DESIGN – SEQUENCE DIAGRAMS

Player Interact with another character



Move Object in Room
Sequence Diagram





Move Player Sequence Diagram

CLASS DESCRIPTIONS

NOTE: functions covered by superclasses are not included for brevity.

ENTITY – A general container class which contains attributes that will be manipulated by Interactions to move the game forward and keep track of changes / player progress.

getInventory() – returns the entities inventory

getActions() – returns a list of available actions for that entity.

getDescription() – returns a description of the entity

setDescription – changes the description of the entity

addItem () – adds an item to the entities inventory

removeItem – removes an item from the entities inventory

interactRoom () – tests if an interaction is valid for a room

interactCharacter() – tests if an interaction is valid for a character

entityInteraction() – initiates an interaction with another entity using the string as a key for the interaction to be attempted.

help () – displays available actions and provides hints for the entity

showInventory() – displays an entities inventory

displayActions() – displays the available actions for an entity.

checkForItem – checks if an entity has an item in its inventory.

ROOM – A specialized container for room specific attributes.

getMap() – returns a string which acts as a map

getAdjacentRooms() – returns adjacent rooms

setMap() - sets a rooms map

addAdjacentRoom() – adds a new room pointer to the map of rooms

removeAdjacentRoom() – removes a room pointer from the map of rooms

addCharacter() – adds a character to the list of characters in a room

removeCharacter() – removes a character from the list of characters in a room.

CHARACTER / NPC – A specialized container for Character/NPC actions for an entity modified by interactions to change the character and move the story forward.

getStats() – returns a map<string, int> of a characters stat.

getStat() – returns the value of a specific stat.

getRoom() – returns a pointer to the room that a character is currently in.

setRoom() – sets the room a character is in.

PLAYER – A specialized container to keep track of all the Player characters information. Contains calls to Player specific actions.

playerMove() – moves the player to a different room depending on their input and which room they are currently in

playerInteract() – Asks the player what they would like to interact with and what interaction they would like to perform based on the room they are in and the other characters also in that room (this may include actions self referential to the players action map).

playerUseItem() – Asks the player what item they would like to use and on what. Interaction will then check if this is a valid interaction and implement any changes that happen because of the interaction.

playerHelp() – Calls player help. This is dependent on the room the player is in and the other characters and items in that room.

INPUT – Used to provide functions for extracting inputs and formatting inputs to prevent errors and simplify the implementations of functions that use inputs.

validInput() – checks that an input is of an appropriate length and character set (this includes ending the input when a space is detected).

getInput() – Asks the user for input and then modifies a pointer with that new value.

lowercase(string) - turns all upper-case letters to lowercase to simplify input matching.

ITEM – a basic unit of inventory used for interaction matching and to provide players with description hints.

getName() – returns the name of an item

getDescription – returns the description of an item.

setName – sets an items name

setDescription – set an items description

INTERACTIONS – Acts as a set of rules for individual actions and changes entities in response to specific actions.

getRoom() – returns the room the interaction is linked to.

setRoom()- sets the room the interaction is linked to.

getActions() – returns a list of actionsOutputKeys

removeActions() – removes an output from the actionOutputs

checkValidInteractions() – checks if an interaction is valid for an entity.

ROOM INTERACTIONS – Initiates interactions and consequences in a room in response to player input.

roomLook() – get a rooms description.

roomTouch() – touch an object in a room potentially trigger additional actions.

roomExamine() – look at a rooms inventory and get a specific description of an item.

roomDig() – attempt to dig in a room may trigger additional actions

roomClimb() – attempt to climb in a room may trigger additional actions

roomUnlockDoor() – attempt to unlock a door, will ask a player to use an item and check if that item is able to unlock the door if true then modify room to include other room or new item.

roomBreakObject() – attempt to break an object may trigger additional actions

roomMeditate() – output many blank lines to clear the terminal and “you feel a sense of calm wash over you”, or a room specific variation.

CHARACTER INTERACTIONS – Initiates interactions and consequences with another character in response to player input.

getCharacter() – returns the character the interaction is linked to

setCharacter() – sets a character to the interaction.

characterSpeak() – speak to a character may trigger additional actions

characterGiveItem() – attempt to give an item to a character may trigger additional actions

characteTakeItem() – attempt to take an item from a character may trigger additional actions

characterAttack() – attempt to attack a character may trigger additional actions

characterKiss() – attempt to kiss another character may trigger additional actions

characterPersuade() – attempt to persuade another character may trigger additional actions.

APPENDICES

APPENDIX A: FIGURES AND TABLES