CPSC 2720 [FALL]

Escape the Lost Keep

[Team Samara]

[Alex Stephen, Sam Beauregard, Kierra Gagnon, Jackie Eshriew]

[Dec 7th 2021]

Page Break

TABLE OF CONTENTS

INTRODUCTION

Escape the Lost Keep is a software that will implement a text-based game with Rooms, NPC's, Items, and Players. The objective is that you awake in a castle and are trapped in the dungeon. Each room contains clues which will either aid you in getting into the next room or aid you further down the line, you must use the items and clues found to navigate through and escape the castle.

Escape The Lost Keep

This is a text-based puzzle solving game that will let the player practice their problem-solving skills. This game consists of 11 rooms in which the player will traverse to find clues, characters, and items to further develop their story.

- Each Room
    - Every room will contain either an item, NPC or a puzzle to solve, in which the player must figure out the needed item or story progression in order to further the plot.
        - There is a total of 11 rooms in this game
- Each Item
    - Every item will have a designated spot for it to be used and the player must possess the item to use it. The player must take action to get the item and use the item.
        - If the item is not in the player's possession, then the player may not progress this riddle, when they possess the needed item, a new room may be unlocked, and another item may be obtained, or character NPC dialogue may be unlocked when returning to talk to the NPC again.
- Each NPC
    - Each Non-Playable Character will have a dialogue with the player hinting towards another riddle or item that may aid the player, when the player has completed the NPC's riddle, they will get an end of dialogue text indicating that the riddle has been solved.
- Each Riddle
    - Contains the information and implementation for solving a riddle.

PROJECT MANAGEMENT

TEAM ROLES

<u>**Team Member**</u>	<u>**Design - Draft**</u>	<u>**Design – Final**</u>	<u>**Implementation - Basic**</u>	<u>**Implementation - Final**</u>
Kierra Gagnon	Phase Lead	Design Lead	QA Lead	Reporting Lead
Jackie Eshriew	Design Lead	QA Lead	Reporting Lead	Phase Lead
Aragorn	QA Lead	Reporting Lead	Phase Lead	Design Lead
Boromir	Reporting Lead	Phase Lead	Design Lead	QA Lead

Team Role Responsibilities

Each member will shift between each room throughout each phase of the project to have a equal chance at trying out each role. Each team member will also help one another if one member is struggling on a problem.

RISK MANAGEMENT

1. We will prioritize the list of classes to be developed
   a. The main class for the game to run
   b. How the class for Player will interact with the other classes and methods to work properly and display the proper content on screen. The game will also display the user’s actions in the room.
   c. The class NPC will focus on how each NPC will react to the player, wither it being giving the player and item or hinting to the player a clue to escaping.
   d. The item class will store all the possible items to obtain within the game and where they will be used if the player has it in the inventory
   e. The movement will locate which room the player is in and what movements are possible in that room.
   f. The help class will give the Player a hint and what puzzles has not been solved.
2. We agree to have meetings though discord and short in person meetings, when possible, every week to determine the progress made on the project and what goal needs to be met over the coming week.

DEVEVLOPMENT PROCESS

CODE REVIEW PROCESS

1. Team member will take note of proposals and bring it up in meetings in discord or in person and will also added documentation of the proposal for the other members to review and discuss. There must be a copy of the documentation available to all member in discord, email or google docs.
2. All ideas for project changes must be discussed and agreed to before implementing them into the implementation phase.

COMMUNICATION TOOLS

We will primarily use discord to communicate with one another and to share updates on sections individuals are working on. During the testing phase group members will request the assistance of Nicole Wilson and Dr. John Anvick when facing difficulties.

CHANGE MANAGEMENT

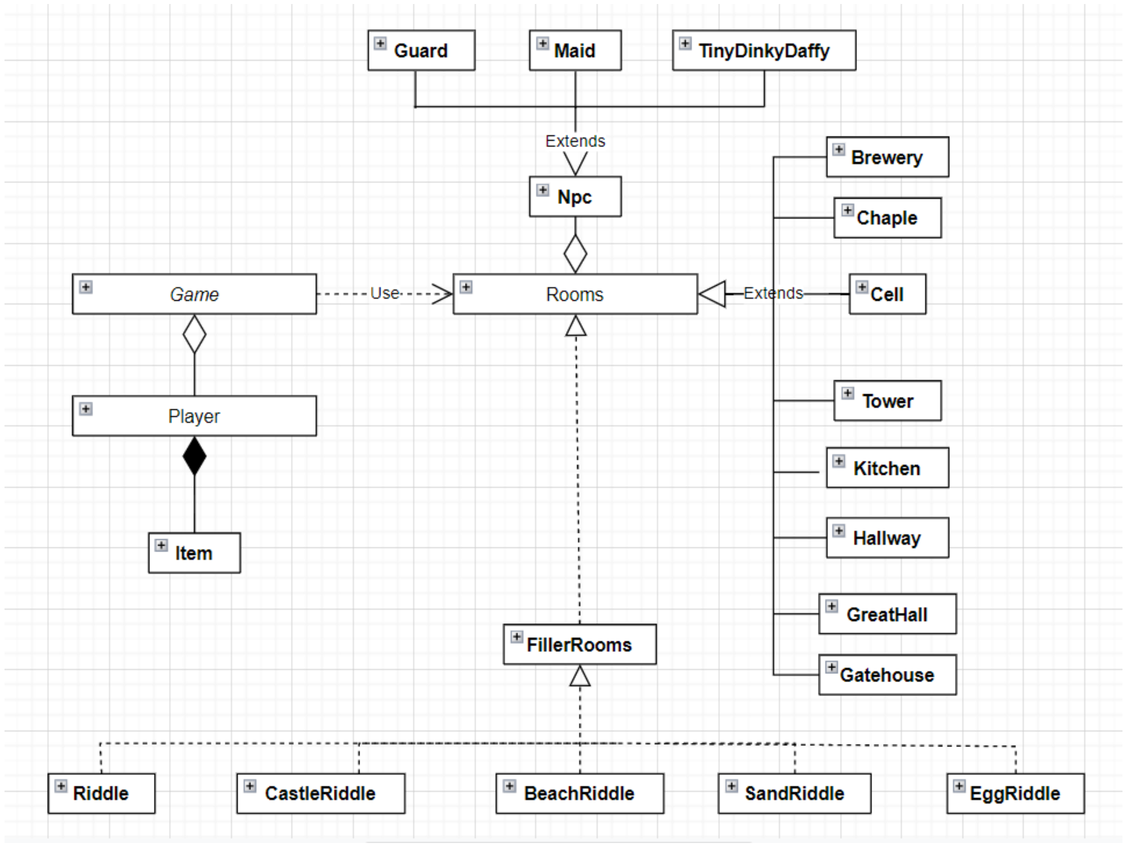
The Quality Assurance Lead will manage all bug reports and merge requests.

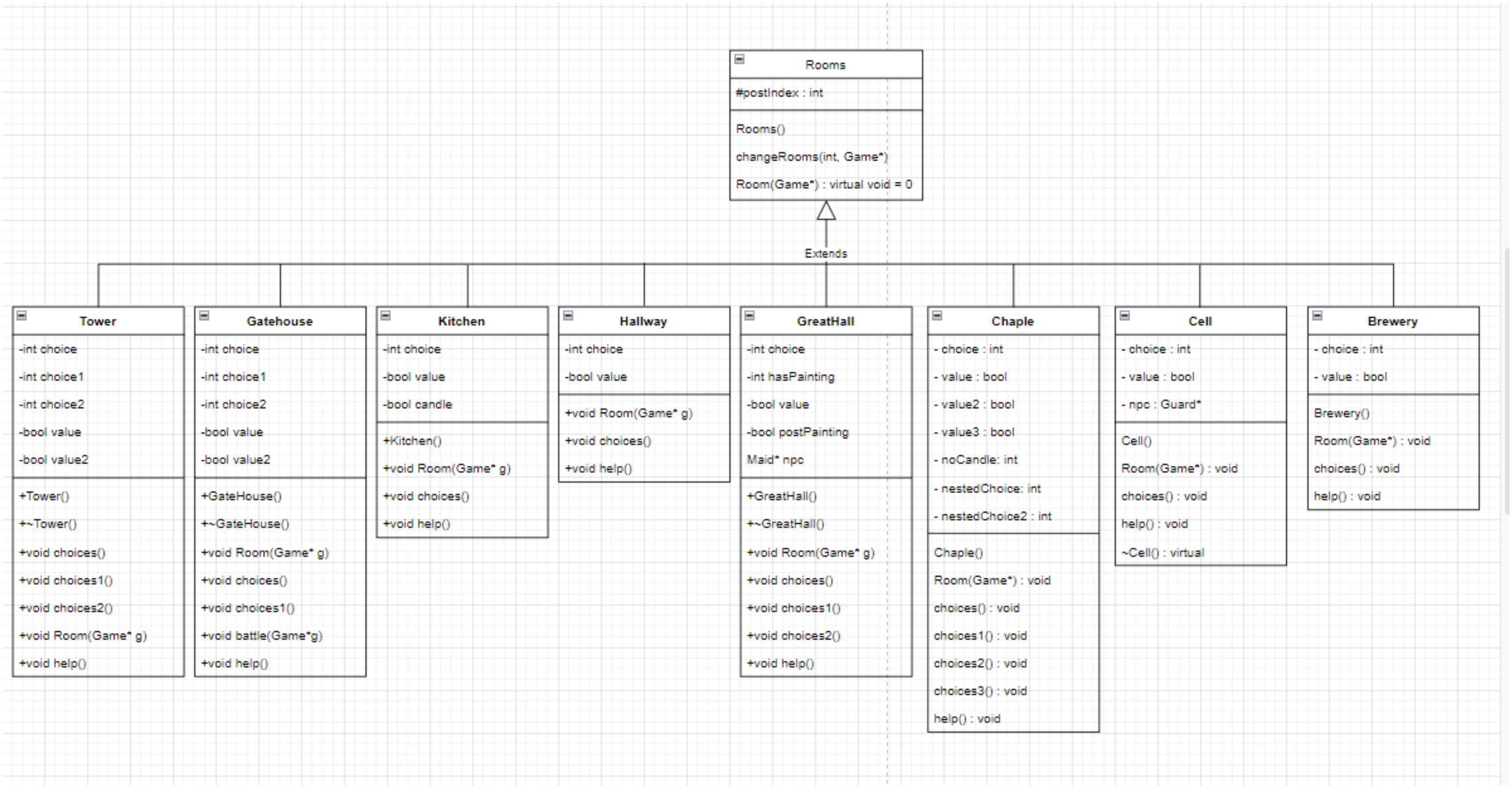
1. The Lead will identify errors and bugs in the methods and report back to the initial team member or members to discuss and fix the code before requesting another merge.

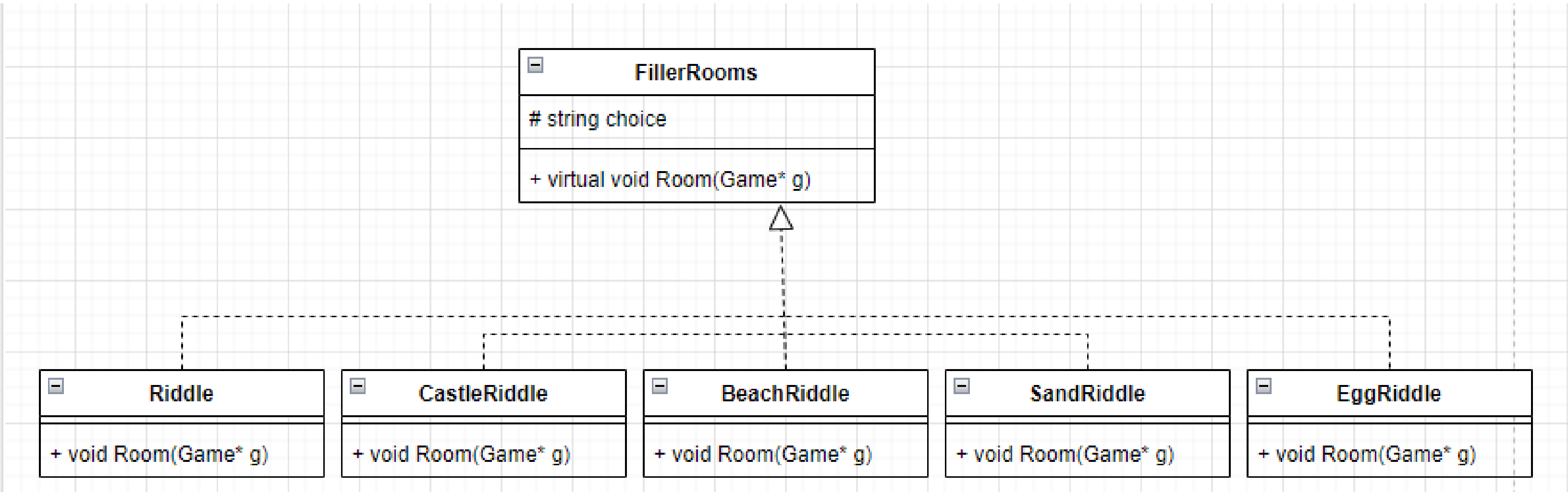
SOFTWARE DESIGN

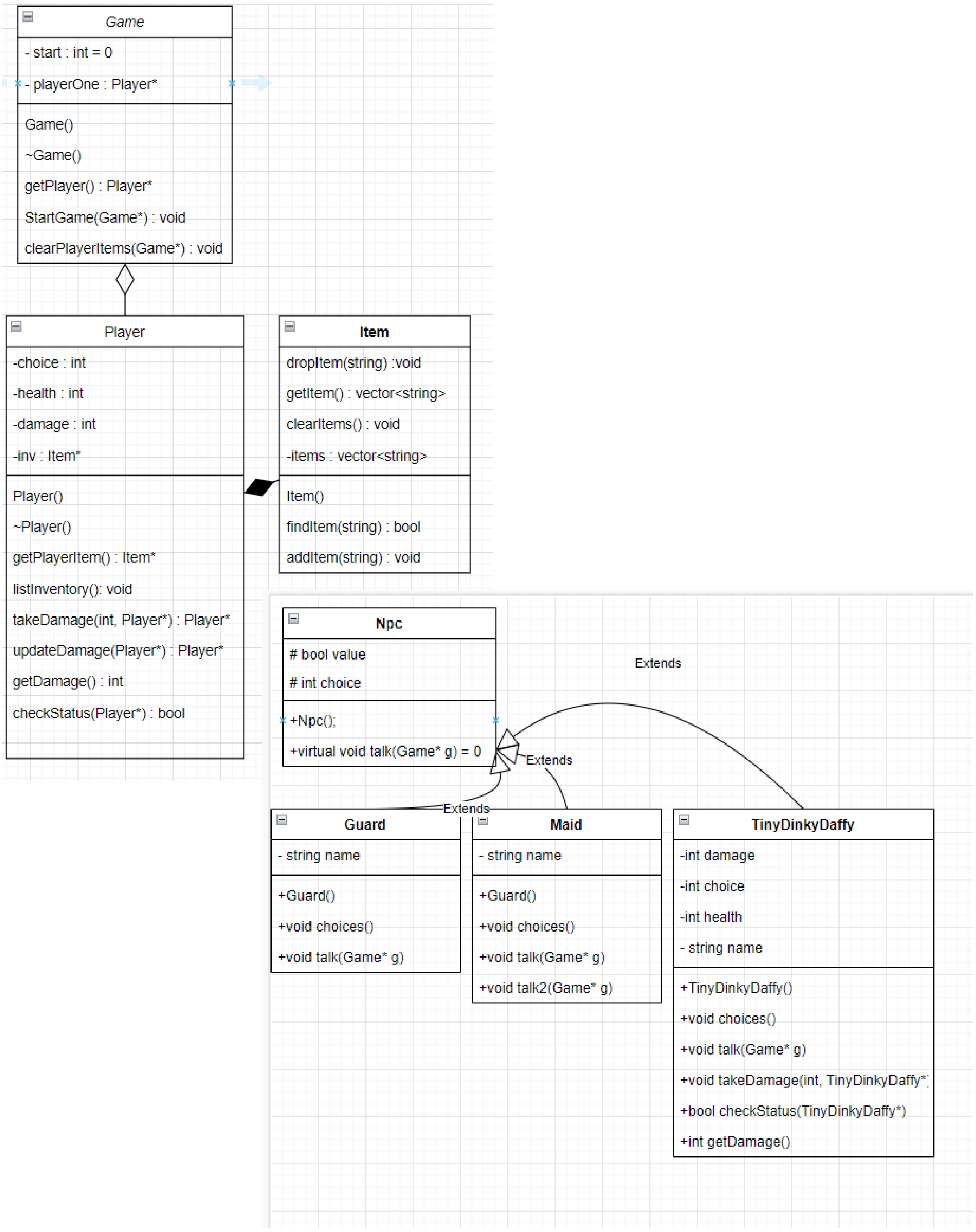
The goal of this project is to work as a group to make the design decisions, illustrate the principles of Practical Software Development in CPSC2720 and use what was taught in the course to better the project and individual knowledge.

DESIGN – CLASS DIAGRAMS

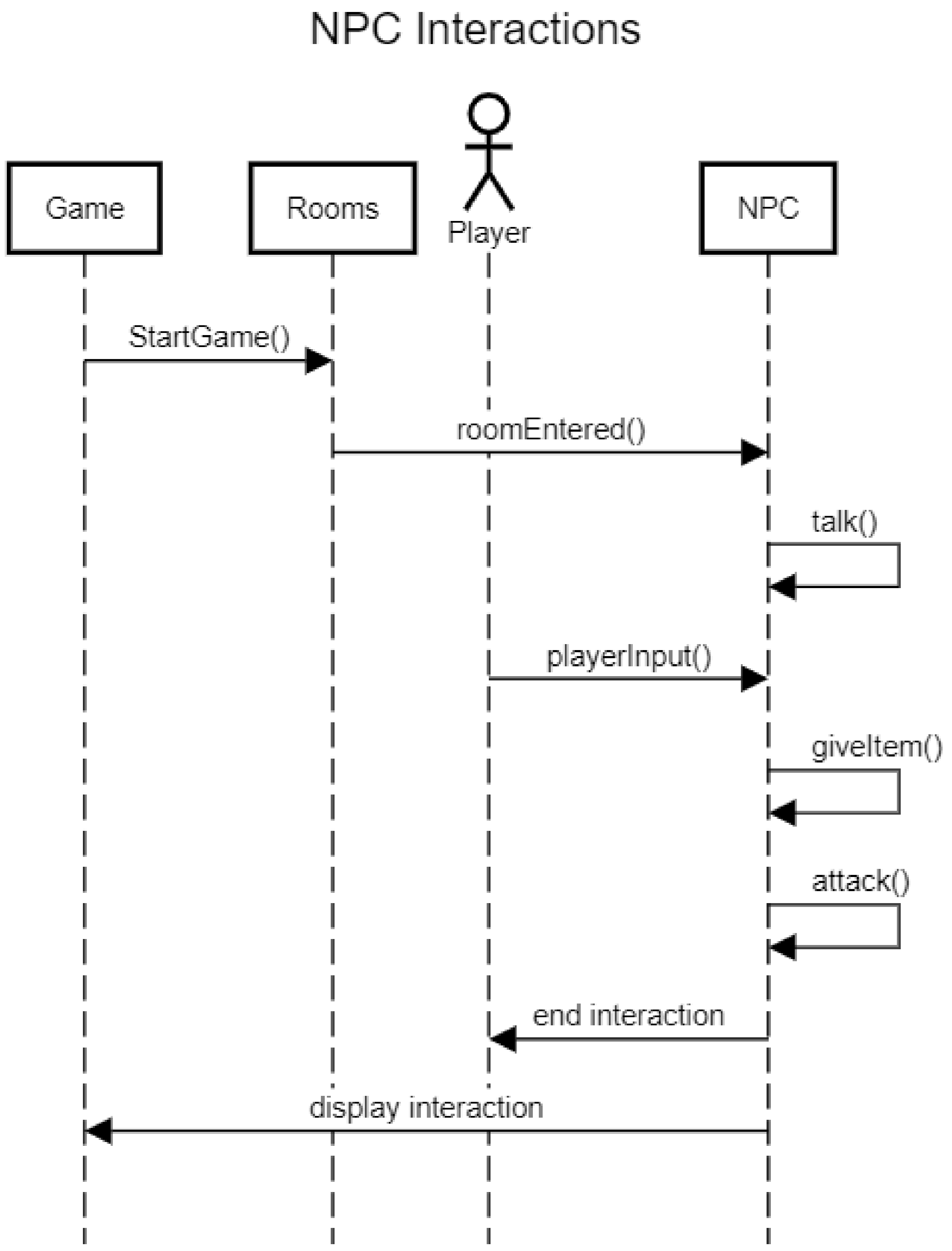


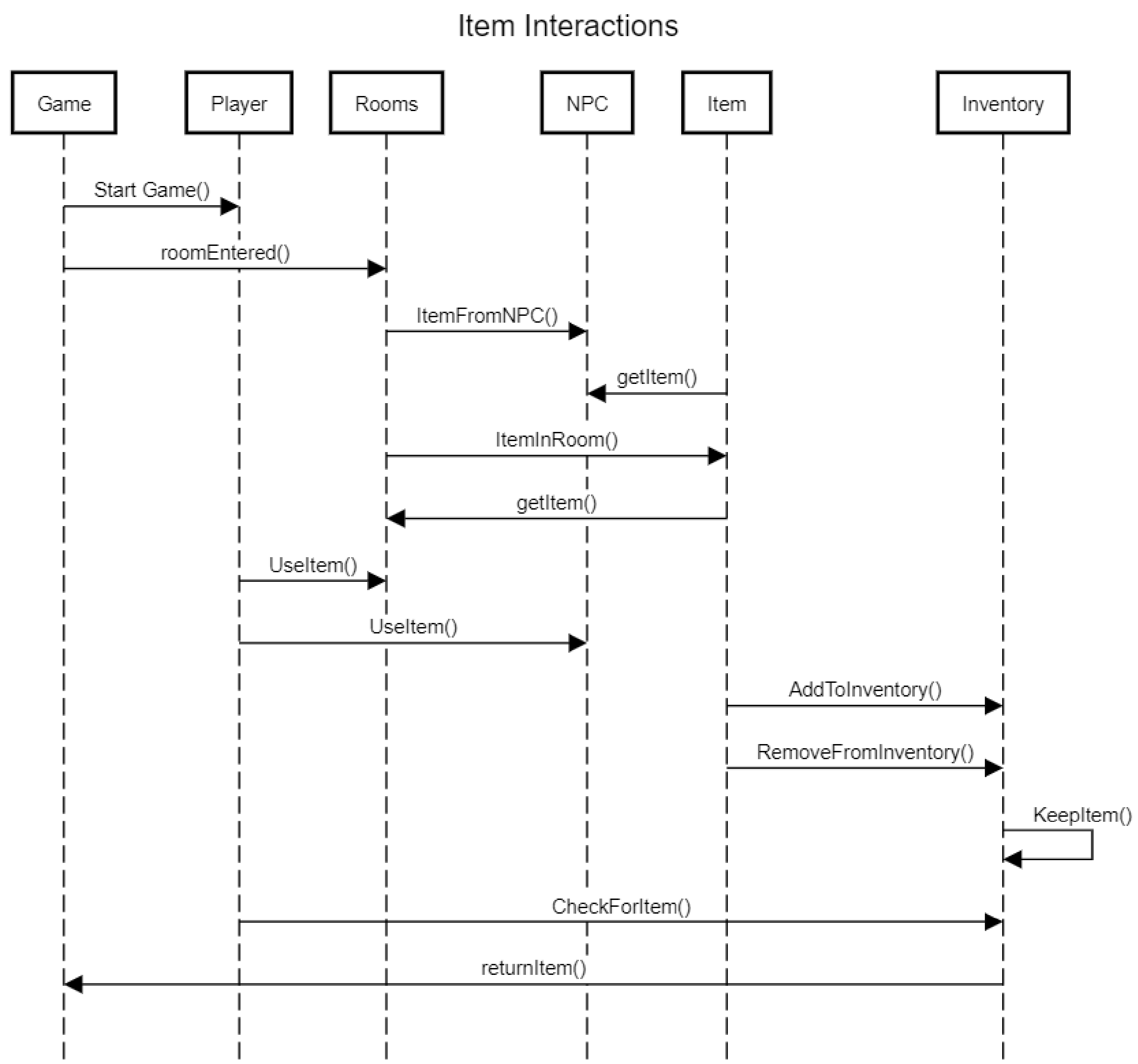

Rooms
#postIndex : int
Rooms()
changeRooms(int, Game*)
Room(Game*) : virtual void = 0
Extends
Tower
-int choice
-int choice1
-int choice2
-bool value
-bool value2
+Tower()
+~Tower()
+void choices()
+void choices1()
+void choices2()
+void Room(Game* g)
+void help()
Gatehouse
-int choice
-int choice1
-int choice2
-bool value
-bool value2
+GateHouse()
+~GateHouse()
+void Room(Game* g)
+void choices()
+void choices1()
+void battle(Game*g)
+void help()
Kitchen
-int choice
-bool value
-bool candle
+Kitchen()
+void Room(Game* g)
+void choices()
+void help()
Hallway
-int choice
-bool value
+void Room(Game* g)
+void choices()
+void help()
GreatHall
-int choice
-int hasPainting
-bool value
-bool postPainting
Maid* npc
+GreatHall()
+~GreatHall()
+void Room(Game* g)
+void choices()
+void choices1()
+void choices2()
+void help()
Chaple
- choice : int
- value : bool
- value2 : bool
- value3 : bool
- noCandle: int
- nestedChoice: int
- nestedChoice2 : int
Chaple()
Room(Game*) : void
choices() : void
choices1() : void
choices2() : void
choices3() : void
help() : void
Cell
- choice : int
- value : bool
- npc : Guard*
Cell()
Room(Game*) : void
choices() : void
help() : void
~Cell() : virtual
Brewery
- choice : int
- value : bool
Brewery()
Room(Game*) : void
choices() : void
help() : void

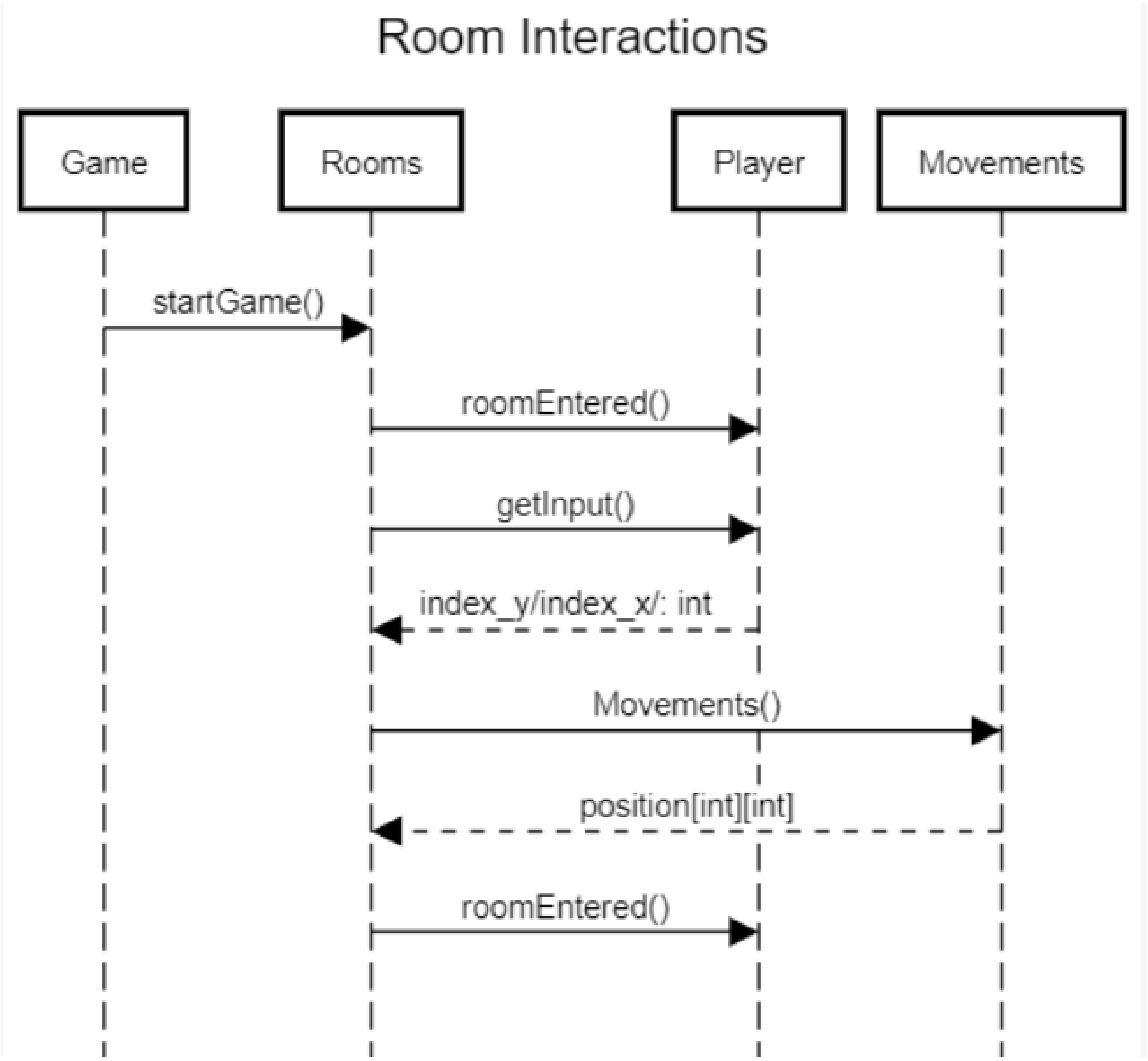

FillerRooms
# string choice
+ virtual void Room(Game* g)
Riddle
+ void Room(Game* g)
CastleRiddle
+ void Room(Game* g)
BeachRiddle
+ void Room(Game* g)
SandRiddle
+ void Room(Game* g)
EggRiddle
+ void Room(Game* g)


Game
- start : int = 0
- playerOne : Player*
Game()
~Game()
getPlayer() : Player*
StartGame(Game*) : void
clearPlayerItems(Game*) : void
Player
-choice : int
-health : int
-damage : int
-inv : Item*
Player()
~Player()
getPlayerItem() : Item*
listInventory(): void
takeDamage(int, Player*) : Player*
updateDamage(Player*) : Player*
getDamage() : int
checkStatus(Player*) : bool
Item
dropItem(string) :void
getItem() : vector<string>
clearItems() : void
-items : vector<string>
Item()
findItem(string) : bool
addItem(string) : void
Npc
# bool value
# int choice
+Npc();
+virtual void talk(Game* g) = 0
Extends
Extends
Extends
Guard
- string name
+Guard()
+void choices()
+void talk(Game* g)
Maid
- string name
+Guard()
+void choices()
+void talk(Game* g)
+void talk2(Game* g)
TinyDinkyDaffy
-int damage
-int choice
-int health
- string name
+TinyDinkyDaffy()
+void choices()
+void talk(Game* g)
+void takeDamage(int, TinyDinkyDaffy*
+bool checkStatus(TinyDinkyDaffy*)
+int getDamage()

DESIGN – SEQUENCE DIAGRAMS




Item Interactions
Game
Player
Rooms
NPC
Item
Inventory
Start Game()
roomEntered()
ItemFromNPC()
getItem()
ItemInRoom()
getItem()
UseItem()
UseItem()
AddToInventory()
RemoveFromInventory()
KeepItem()
CheckForItem()
returnItem()


Room Interactions
Game
Rooms
Player
Movements
startGame()
roomEntered()
getInput()
index_y/index_x/: int
Movements()
position[int][int]
roomEntered()

CLASS DESCRIPTIONS

1. Player Class: Dedicated to player input and the player's inventory

    a. Default Class constructor and destructor. Initializes private variables.
    b. Item* getPlayerItems() is used to return a specific Item object to keep the player inventory consistent across the program.
    c. listInventory() is a void function that prints a numbered list of the player's inventory to the screen. Accessed via player input.
    d. Player* takeDamage(int, Player*) returns the updated player object upon taking damage.
    e. Player* updateDamage(Player*) returns the updated player object with changed damage if the player has a specific item.
    f. Int getDamage() returns the Player's damage
    g. Bool checkStatus(Player*) returns a boolean status on whether the player has less than 1 health.

2. Game class: Dedicated to running the game and contains all items and rooms.

    a. Default class constructor and destructor. Initializes certain variables.
    b. startGame(Game*) method is what initializes the game.
    c. clearPlayerItems(Game*) empties the player's inventory upon death or restart.
    c. exitGame() can be called to end the game and exit the terminal

3. Room class: Super class that handles changing between rooms

    a. Room(): is a default constructor.
    b. Void changeRooms(int, Game*): changes the room that the player is currently in and moves then to the next or previous room.
    c. Various room subclasses such as Gatehouse, GreatHall, Cell etc.
        i. Contains a constructor
        ii. Destructor
        iii. help()
        iv. void Room(Game*) function which implements the gameplay for that room
        v. choices() which displays to the player their interaction options

4. Item class: Handles adding/removing and finding the player's items.

    a. Default constructor
    b. Bool findItem(string) searches for a specific item in the players inventory and returns a boolean value depending on the answer
    c. addItem(string) adds an item to the player's inventory
    d. dropItem(string) removes an item from the player's inventory

e. vector<string> getItem() returns the items vector for use in other classes/functions
f. clearItems() empties the inventory. Used by Game class.

5. NPC: Super class for NpcsContains void functions called in specific rooms. NPCs can give items, exchange dialogue and hints and other functionalities.

    a. Npc(): Sets Default values
    b. Virtual void talk(): virtual function for npc implementation
    c. Various Npc subclasses have constructor, choices, and talk.

6. FillerRooms:Superclass for Riddles
    a. Virtual void Room()

7. Riddle: Super class for Castle, Beach, Sand, Egg Riddle
    a. Room();
    b. Each sub room contains a Room function which contains the Riddle and implementation to go with it