Team Name: Nyx

Team Letter: ‘N’

Don Castillo

Tyler Justinen-Teite

Victor Besson

Due Date: 17/02/2020

Table of Contents

1. Introduction

Nyx's idea is to create a text-based adventure game as a medium for us to apply our software development and collaborative skills. Throughout this project we aim to keep the workload small at each step by breaking down the classes into core functionality, with the ability to expand upon them later. One way this goal will be achieved is through creating each class as dynamically as possible, as hard-coding functionality could result in us needing to change entire systems (depending on what it is obviously) to add new features.

You are an adventurer that has come across a very old secret text and after many years of deciphering the text you have discovered a location to a place of immense knowledge and endless power. The game begins with the adventurer going to this location and discovering the spire.

Disciple of the Spire is a dungeon crawler, inspired by the popular *Slay the Spire* genre, and the classic RPG, Dungeons and Dragons. The general gameplay flow will be starting in the entrance of the spire, then each room will have the option to traverse into one of one-to-three randomly selected rooms. These rooms may contain monsters, bosses, chests, a shop, or a puzzle room. The player will gain experience, loot, abilities, and money from monsters, chests and bosses to advance through the spire's ever-increasing difficulty. Once the player has completed twenty-four rooms, they will reach the final boss, where their skills will truly be tested, and their journey will either come to an end or they can continue endlessly through the depths of the spire.

This report was written as part of the Design Phase of the project. The Project Management section of this report explains how the team is structured, and details the roles and responsibilities of each team member. It also identifies foreseeable risks that could prevent the team from completing the project and how the team will handle those risks. Lastly, the Development Process section describes how the team will collaborate together throughout the game development. It also explains the software

design and provides visual concepts of game design implementation through a UML diagram.

2. Project Management

2.1. Team Organization

This section introduces the members of the Nyx team as well as the initial designation of each member throughout the phases of the project. It also explains the specific roles and responsibilities that each designation requires.

Nyx team is composed of three members:

- Tyler Justinen-Teite
- Victor Besson
- Don Castillo

One of the goals of the project is for the members to gain more experience in a team-oriented work setting and apply good software development practices. Every member of the team is expected to fill the roles of Software Developer and Tester. In addition to that, everyone is given a specific role to fulfill as determined by the course instructor. The roles will be rotating in every phase of the project to give each member an experience of handling different roles. The initial designation of each member is shown in the table below and the designation descriptions are provided in the next section.

Table 1. Nyx members' initial designation

DESIGNATION	PROJECT PHASE		
	DESIGN	IMPLEMENTATION	TESTING / MAINTENANCE
Project Lead	Tyler		
Phase Lead	Victor	Tyler	Don
QA Lead	Tyler	Don	Victor
Document Lead	Don	Victor	Tyler

2.1.1. Project Lead

The project lead oversees the entire project and the entire team throughout all the phases of the project. We have decided to have this position available in order to lessen team members confusion, keep the project moving forward, and have someone direct the team members towards certain and achievable goals. The project lead will also provide guidance to the phase lead if the latter cannot fulfill his roles and responsibilities. Nyx team has decided to make Tyler the project lead because of his in-depth knowledge about the nature of the game and his extensive experience in software development. Lastly, the project lead is in charge of managing the team's repository.

2.1.2. Phase Lead

The phase lead oversees the project in a particular phase. There are three phases in this project (design, implementation, and testing or maintenance phases), each phase will be led by a different team member. The roles of the phase lead are as follows:

- Understands the requirements for the phase. The instructor will be reached out to ask for clarification about phase requirements.
- Keeps everyone on track regarding phase requirements, meeting agendas, phase tasks, and tasks deadlines.
- Delegates tasks to team members, and makes sure tasks are evenly distributed.
- Lead team meetings.

2.1.3. Quality Assurance Lead

The quality assurance (QA) lead makes sure that the deliverable for each phase is of high-quality and does not contain any errors. Some of the specific roles and responsibilities of the QA lead are follows:

- Creates bug report or issue report on Gitlab repository
- Performs error-checking or unit test in any possible testable item in the source code using Google Test..
- Make sure that the program compiles and/or the build has passed every test including style check, static analysis clean, memory check, etc.
- Coordinates creation of system-level testing plan.

2.1.4. Documentation Lead

The documentation lead oversees the creation and maintenance of project reports and document deliverables in a particular phase. Some of the specific roles and responsibilities of the documentation lead are as follows:

- Keeps everyone posted about document requirements in each phase.
- If the document or report requires everyone to have input or participation, the documentation lead decides who writes what.
- Collects all written documents from the members and compiles it into one file.

- Makes sure the document has met the requirement of the phase before pushing or uploading it into the repository. If the document is to be handed in on Moodle, he will also make sure that it is submitted on time.
- In the last phase of the project, the documentation lead will produce a user manual and a README file.
- Oversees the documentation and comments in the source code. He will make sure he follows the Doxygen documentation convention.

2.2. Risk Management

Risk management describes how the team will address foreseeable risks and issues that could prevent the team from completing the project. This explains how the team will cope with feature creep, how to approach software implementation, how to handle design changes, how to manage team members, and what tools will be used.

2.2.1. Maintaining an Achievable Goal

To manage feature creep on this project, our team has decided to create the core functionality of our game first, then create the characters, weapons, and monsters in such a way that allows us to have a simple game with the ability to expand later if time permits. Abstraction and encapsulation are going to be a key step in our design process, to ensure that systems can interact without being dependent on each other.

2.2.2. Software Development Approach

In order to keep the project in progress and on track, we will be practicing an agile approach to software development called scrum. In this framework, since our goal is to create a working software in the early stage of the development, we will develop the software iteratively. Every meeting (scrum meeting), we will set up goals and create cards (to-do cards) that will let

team members know which tasks are done, being done, and yet to be done. We will use Slack to update each other about the progress of each individually assigned task. This will lessen confusion and prevent merge conflicts. As much as possible, for every function and unit changes, members are expected to commit and push these as soon as possible. This will aid the repository manager (project lead) to know where the error came from as soon as it occurs.

2.2.3. Handling Major Changes in the Project

Any possible changes in the project during implementation will be discussed during the meetings. The decision to accommodate any changes will depend on two conditions: whether it is necessary for the project or if there is enough time to do that. The goal of the team is to have a working software that follows the requirement of the course and in accordance with the initial plan or design. If major changes are inevitable, we will incorporate those in a different branch other than the master branch so that if the changes are successful, we will then merge it into the master branch. If the changes are unsuccessful, we will abandon it.

We have designated the project lead to manage the repository, with responsibilities which involves deciding whether the testing branch, for example, can be merged into the master branch. If merge conflict happens, the member has to notify the project lead about this and resolve it together.

2.2.4. Handling Possible Team Restructuring

If we think that there is a need to restructure the team, possibly due to a member not doing his roles or slacking off, we will have to reassign him to do tasks he is comfortable or familiar with. If the phase lead cannot fulfill his tasks, the project lead will have to take over as the new phase lead. A member who may lack technical skills will be asked to shadow

another member or assign him to do less technical tasks like documentation (he might be reassigned as the new document lead). If someone should leave the team, for whatever reason, the leaving member will have to orient the other members who will be assigned to tasks that he will leave behind. Dr. Anvik will be informed about this. Should a new member be added to the team, he will be asked to assist the Quality Assurance Lead.

2.2.5. Learning & Tools

Our plan is to use the tools that are readily available in the lab and are taught in the class. Gitlab will host our source code and everyone has been assigned a role as maintainers. We will be using terminal-based Git as our version control, and Code Blocks as our code editor. The team believes that everyone has adequate experience with these tools and will not have problems using them. If the team decides to use other tools that are not taught in class, we will consult Dr. Anvik.

3. Development Process

This section describes the process that the team will follow in developing the software. It discusses how the code will be reviewed, how the team will communicate, and how the repository will be managed.

3.1. Coding

3.1.1. Writing the Code

Every member, as a software developer and tester, is expected to take initiatives in coding. Before coding, members should inform each other on what functionalities will be written by whom. Team members will be responsible for communicating on Slack what files they are currently working on. Each team member will create branches for the

tasks that they are working on, and their branches will be reviewed and merged by the team-lead.

3.1.2. Code Review

Each member is expected to thoroughly review his own code before pushing them into the repository. Code review covers the following:

- Checking for errors in syntax, logic, and runtime.
- Checking for styling errors and memory leaks.
- Performing static analysis.
- Making sure the program still compiles.
- Testing class methods or functions to see if it returns intended data or object.
- Writing brief descriptions on functions and variables to help with Doxygen documentation and help other developers understand the code.

As mentioned, the QA lead will perform general code review but it will not be as thorough as the code review of each member. The documentation lead will use the comments written by each developer to generate Doxygen documentation.

3.1.3. Debugging

If a bug occurs, the member who encounters the bug will have to issue a bug report on Gitlab and inform the team about it. If it is an easy-to-fix bug, the member can debug it and inform the team about the occurrence of the bug and how it was fixed. Each team member will be responsible for handling their own bugs during the development process. We, of course, will help one another with any issues they are having, as it always helps to have fresh eyes when fixing bugs. For fixing bugs, we will create branches for each bug so that they can be solved without any other code changing to help isolate the problem.

3.2. Communication: Tools and Channels

Our team is going to use Slack for team communication. The team also has each other's phone numbers in case Slack is down, or they need to call/text another member for quicker communication.
Nyx will also be using the issues (boards) system built into GitLab to keep track of what everyone is working on and updates on the progress thus far. Team members are expected to create their own bug reports, and report any bugs to the team during scrum meetings.

4. Software Design

This section describes and illustrates the software design of the Disciple of the Spires game. First, a brief description of the process of the game will be explained. Then game design rationale will be discussed.

4.1. Design: How the Game Works

The adventure game is composed of several important components that altogether make the game work (see Figure 1). The game allows the player to start a new game, save the progress of the game, and continue playing the game. At the start of the game, the player will choose his character's attributes like name and race. The player will traverse through different kinds of rooms. Each room connects to other rooms. A room could be an ordinary room, a puzzle room where the player will have to complete, a chest room where a player can loot items, and a monster room where the player will have to defeat the monster in it before going to the next level or room. Along the way, the player will gain experience and abilities which will aid the player to defeat the next monsters or boss. The movement of the player throughout the game is triggered by a dice roll. See Figure 2 and Appendix 5.2 for more details about the game sequence.

4.2. Design Rationale

The team will first write the essential components of the game like the super classes (e.g. Base Character and Items class) and other stand-alone classes (e.g. Abilities, Dice class). The main class that manages the entire game will be written incrementally to allow for testing and making sure that the class methods are properly invoked. Sub classes will be coded as soon as the majority of the methods in the base classes are available.

As mentioned in the Risk Management section, we will practice good object-oriented practice in C++ in order to make implementation fast, minimize error occurrence, and reduce repetition of code. We will utilize C++ concepts like abstraction and encapsulation to create classes that bundle together methods and variables that are related to each other. This also supports data hiding which is important in data integrity since the game will involve a lot of data manipulation. Inheritance and a good use of method overloading and overriding will help developers create usable class methods and functions and avoid redundancy (like two methods which basically do the same thing).

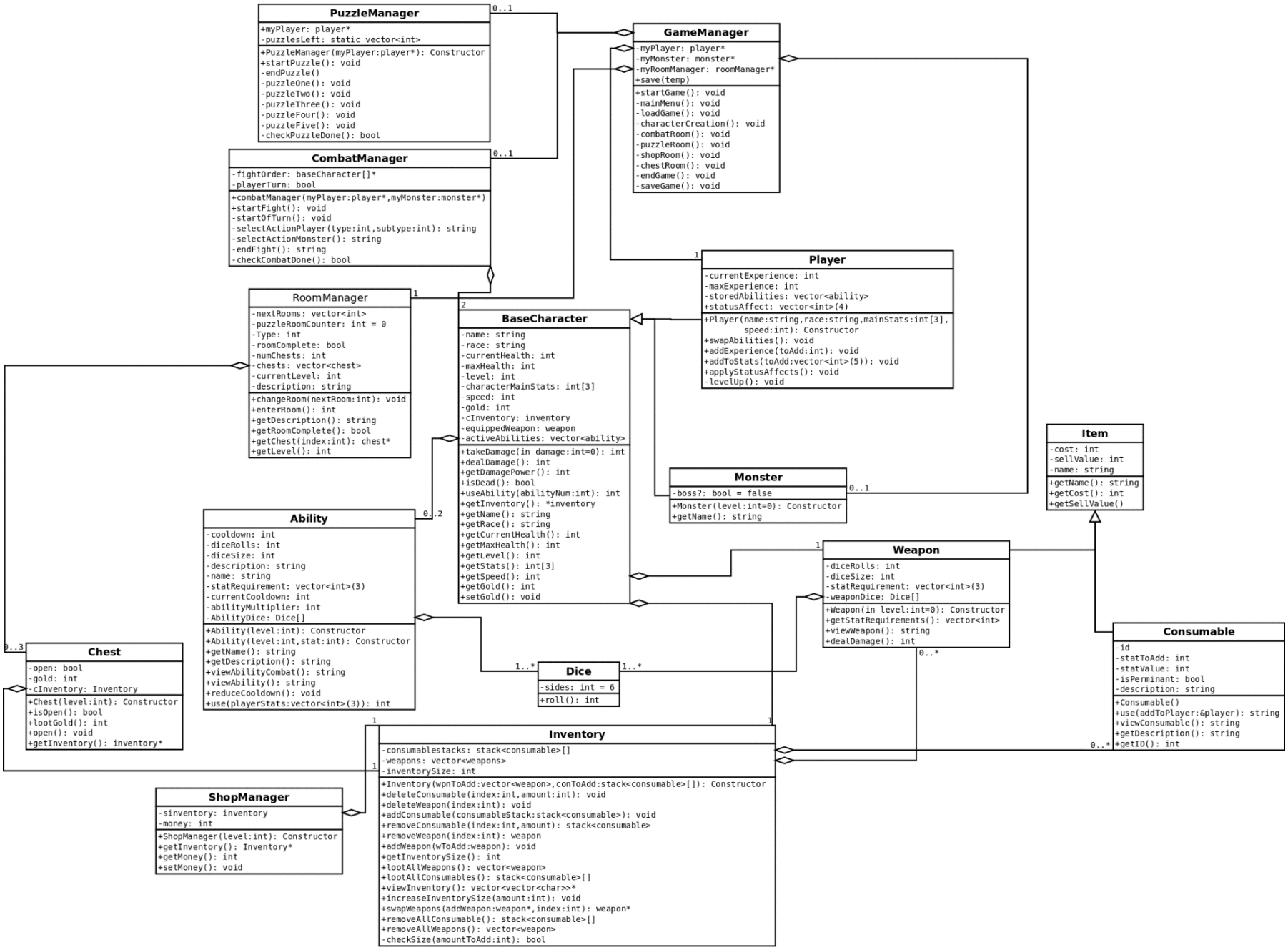


Figure 1. UML Diagram of the Disciple of the Spire (see Appendix 5.1 for close-up view)

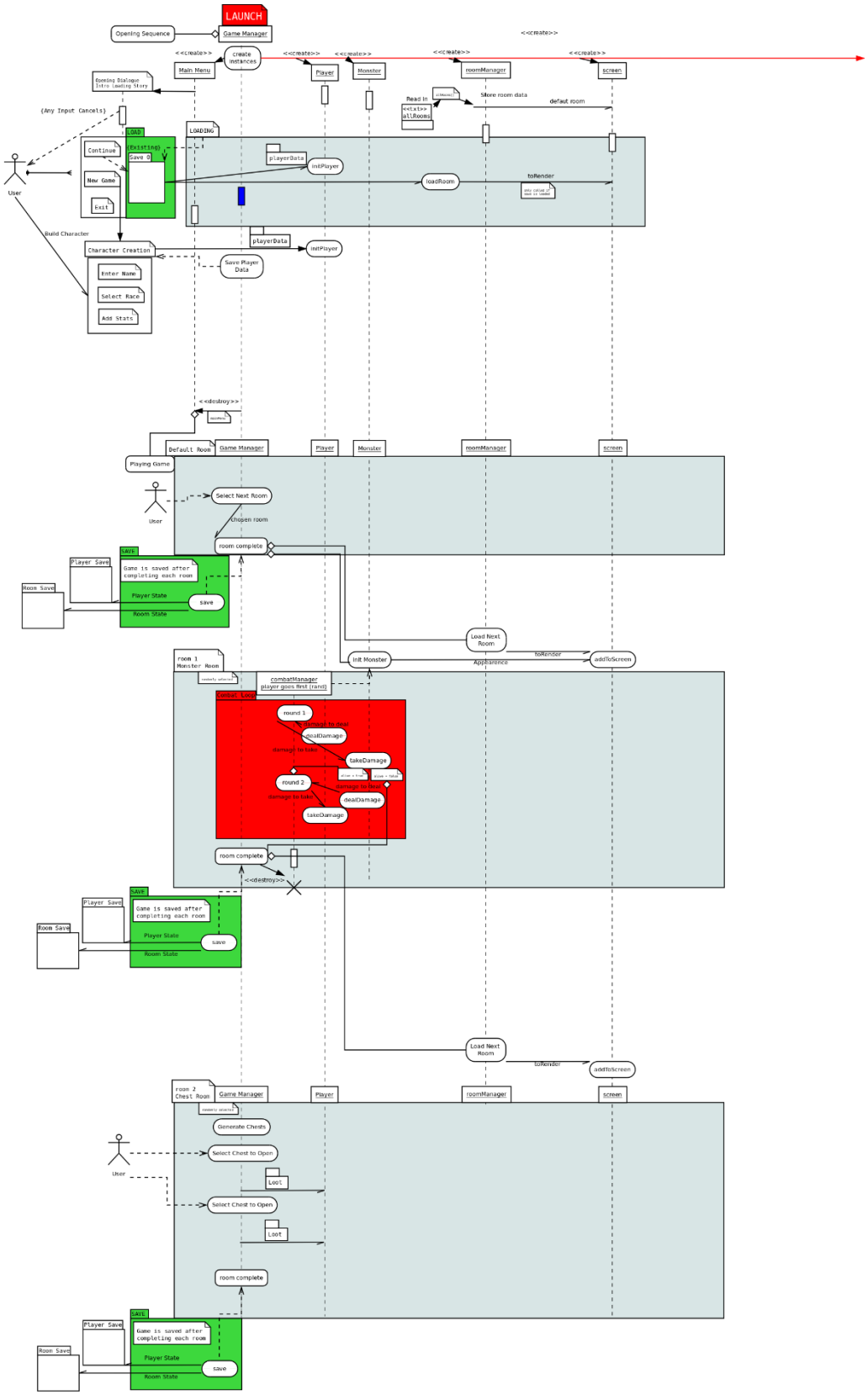
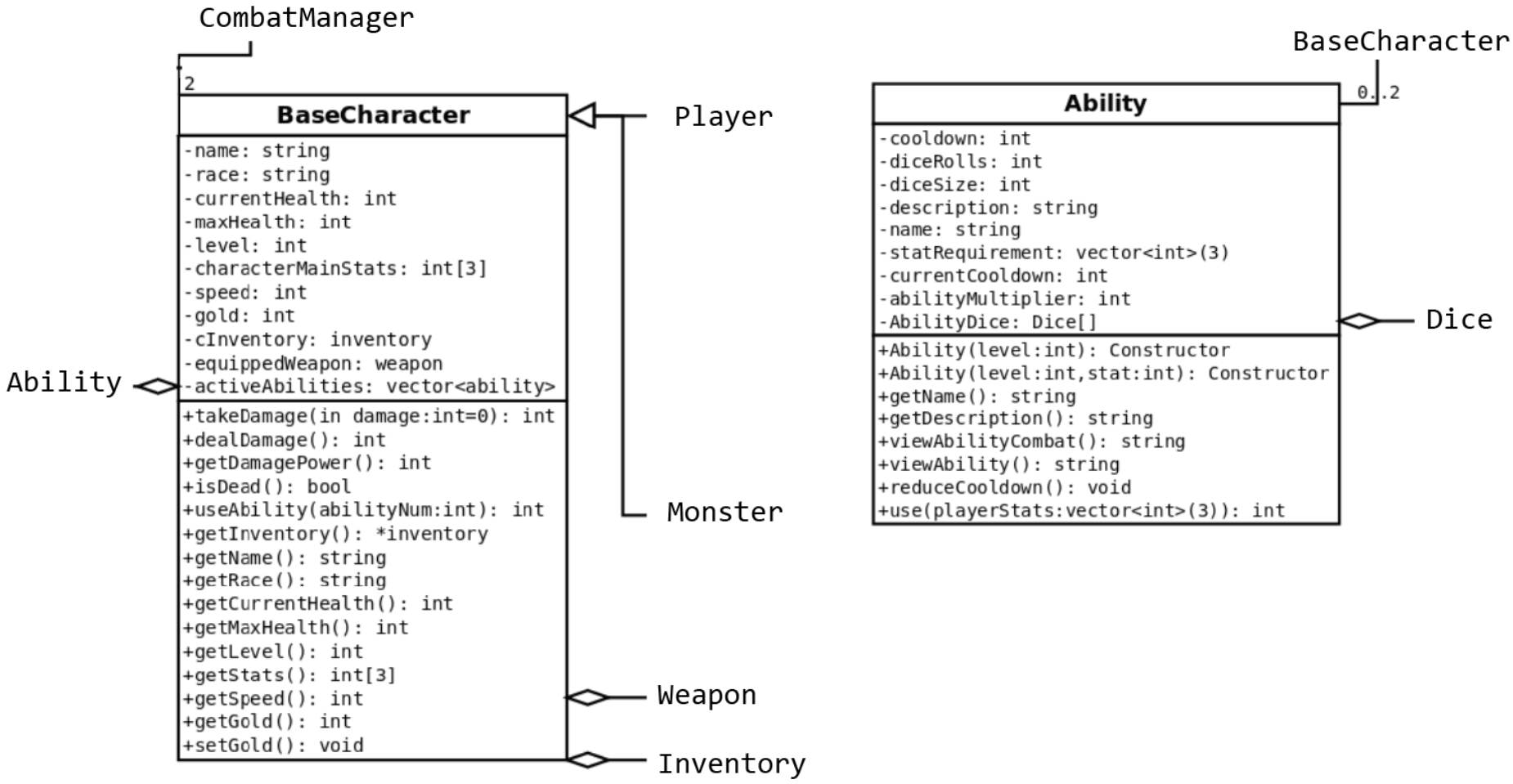
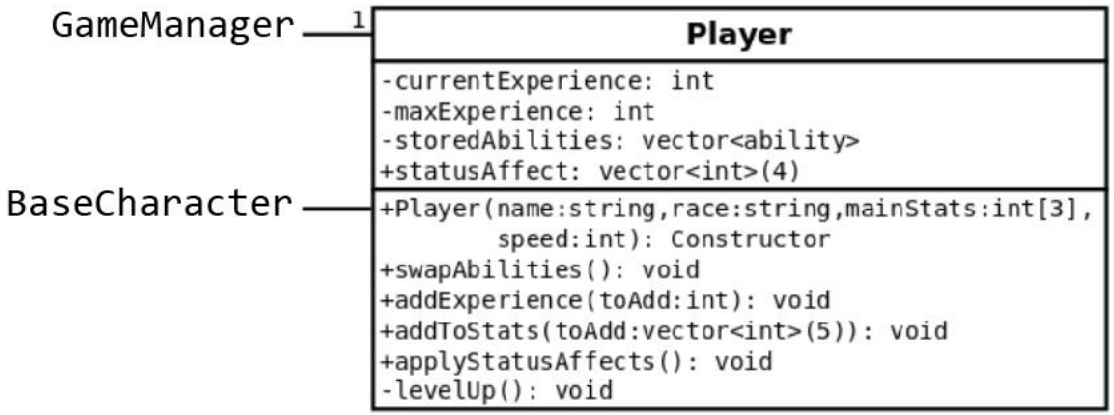


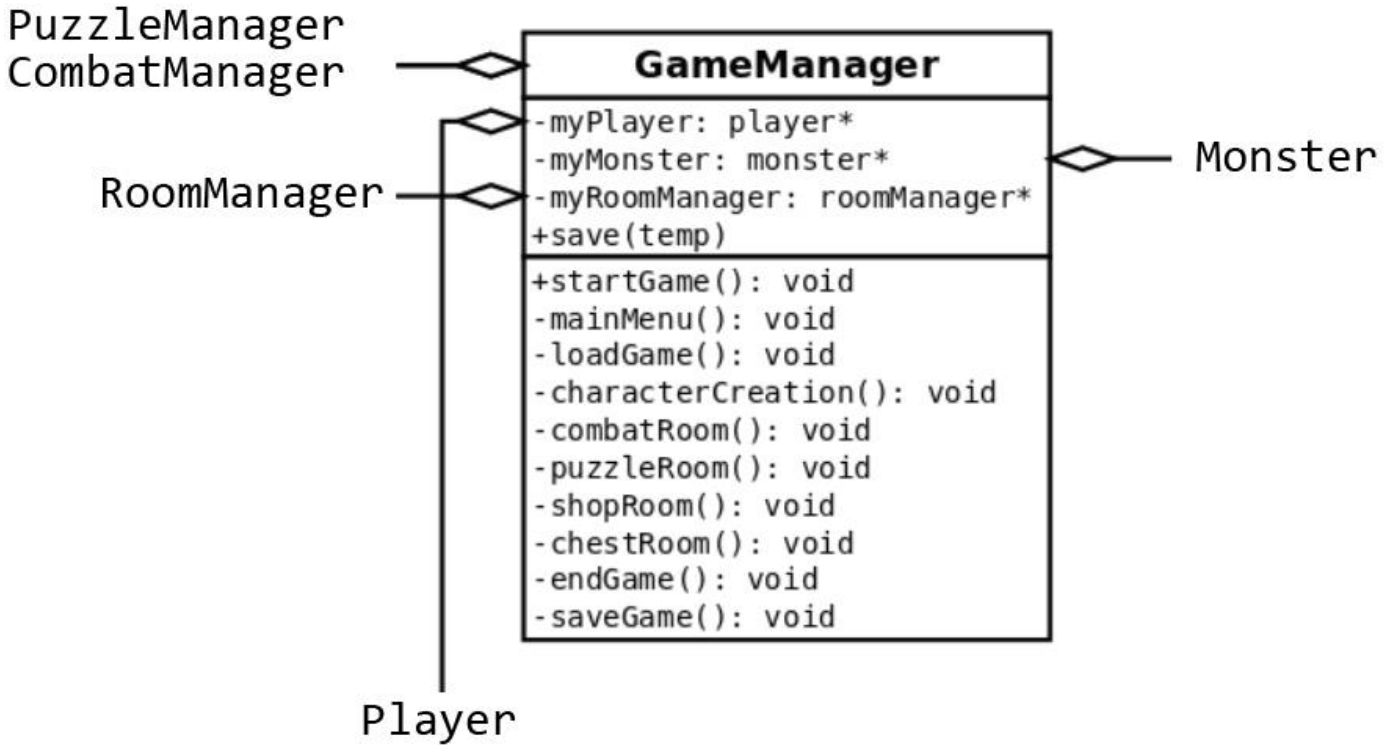
Figure 2. Sequence Diagram of the Disciple of the Spire (see Appendix 5.2 for close-up view)

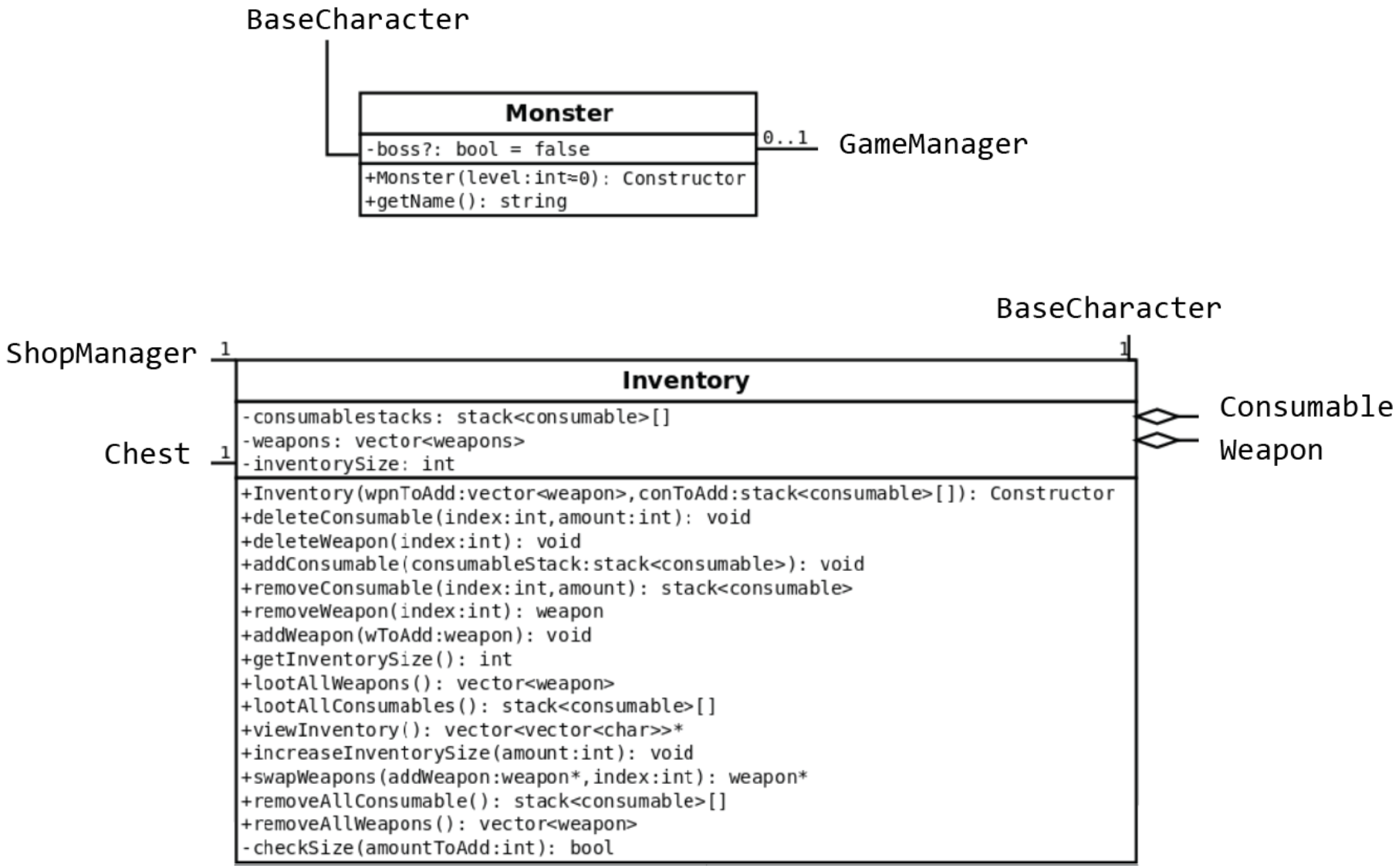
5. Appendix

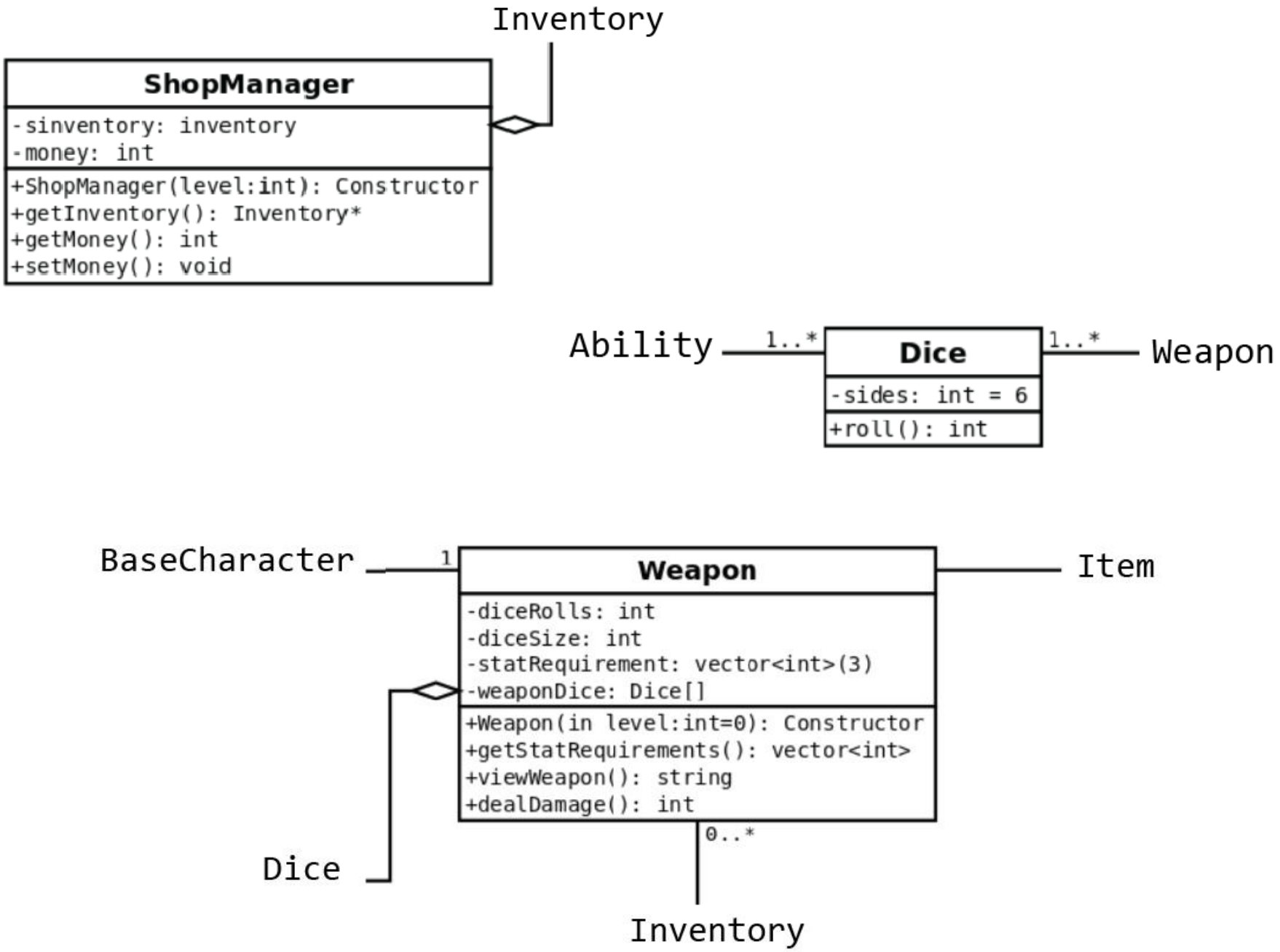
5.1. UML Diagram Classes

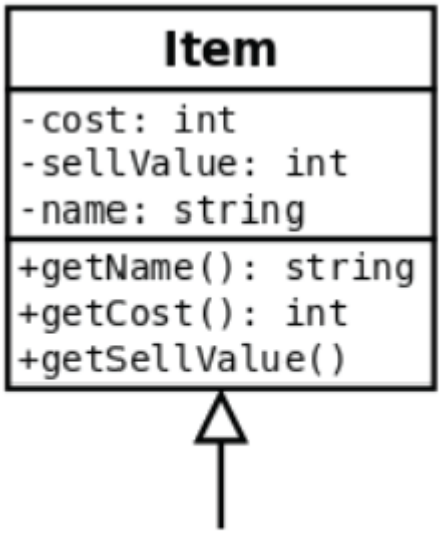



GameManager
1
Player
-currentExperience: int
-maxExperience: int
-storedAbilities: vector<ability>
+statusAffect: vector<int>(4)
BaseCharacter
+Player(name:string,race:string,mainStats:int[3],
speed:int): Constructor
+swapAbilities(): void
+addExperience(toAdd:int): void
+addToStats(toAdd:vector<int>(5)): void
+applyStatusAffects(): void
-levelUp(): void

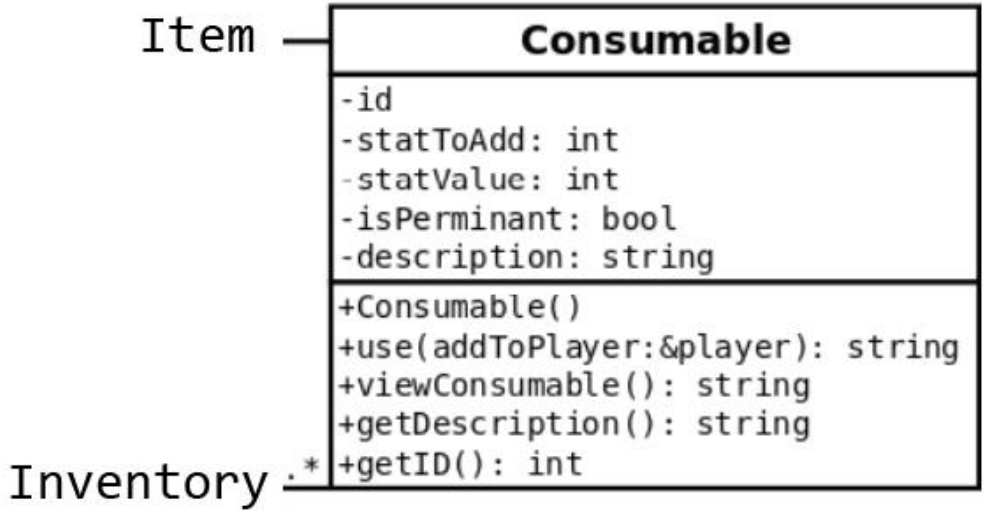

PuzzleManager
CombatManager
GameManager
-myPlayer: player*
-myMonster: monster*
-myRoomManager: roomManager*
+save(temp)
Monster
RoomManager
+startGame(): void
-mainMenu(): void
-loadGame(): void
-characterCreation(): void
-combatRoom(): void
-puzzleRoom(): void
-shopRoom(): void
-chestRoom(): void
-endGame(): void
-saveGame(): void
Player


BaseCharacter
Monster
-boss?: bool = false
+Monster(level:int=0): Constructor
+getName(): string
0..1
GameManager
BaseCharacter
ShopManager
1
1
Inventory
-consumablestacks: stack<consumable>[]
-weapons: vector<weapons>
-inventorySize: int
Chest
1
Consumable
Weapon
+Inventory(wpnToAdd:vector<weapon>,conToAdd:stack<consumable>[]): Constructor
+deleteConsumable(index:int,amount:int): void
+deleteWeapon(index:int): void
+addConsumable(consumableStack:stack<consumable>): void
+removeConsumable(index:int,amount): stack<consumable>
+removeWeapon(index:int): weapon
+addWeapon(wToAdd:weapon): void
+getInventorySize(): int
+lootAllWeapons(): vector<weapon>
+lootAllConsumables(): stack<consumable>[]
+viewInventory(): vector<vector<char>>*
+increaseInventorySize(amount:int): void
+swapWeapons(addWeapon:weapon*,index:int): weapon*
+removeAllConsumable(): stack<consumable>[]
+removeAllWeapons(): vector<weapon>
-checkSize(amountToAdd:int): bool

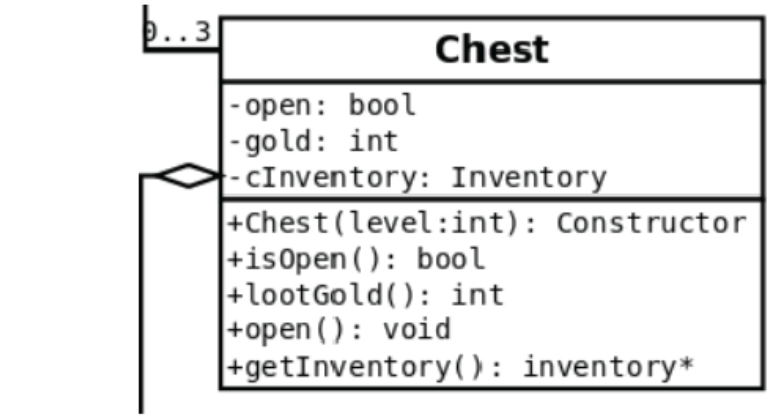

Inventory
ShopManager
-sinventory: inventory
-money: int
+ShopManager(level:int): Constructor
+getInventory(): Inventory*
+getMoney(): int
+setMoney(): void
Ability
1..*
Dice
-sides: int = 6
+roll(): int
1..*
Weapon
BaseCharacter
1
Weapon
-diceRolls: int
-diceSize: int
-statRequirement: vector<int>(3)
-weaponDice: Dice[]
+Weapon(in level:int=0): Constructor
+getStatRequirements(): vector<int>
+viewWeapon(): string
+dealDamage(): int
Item
Dice
0..*
Inventory



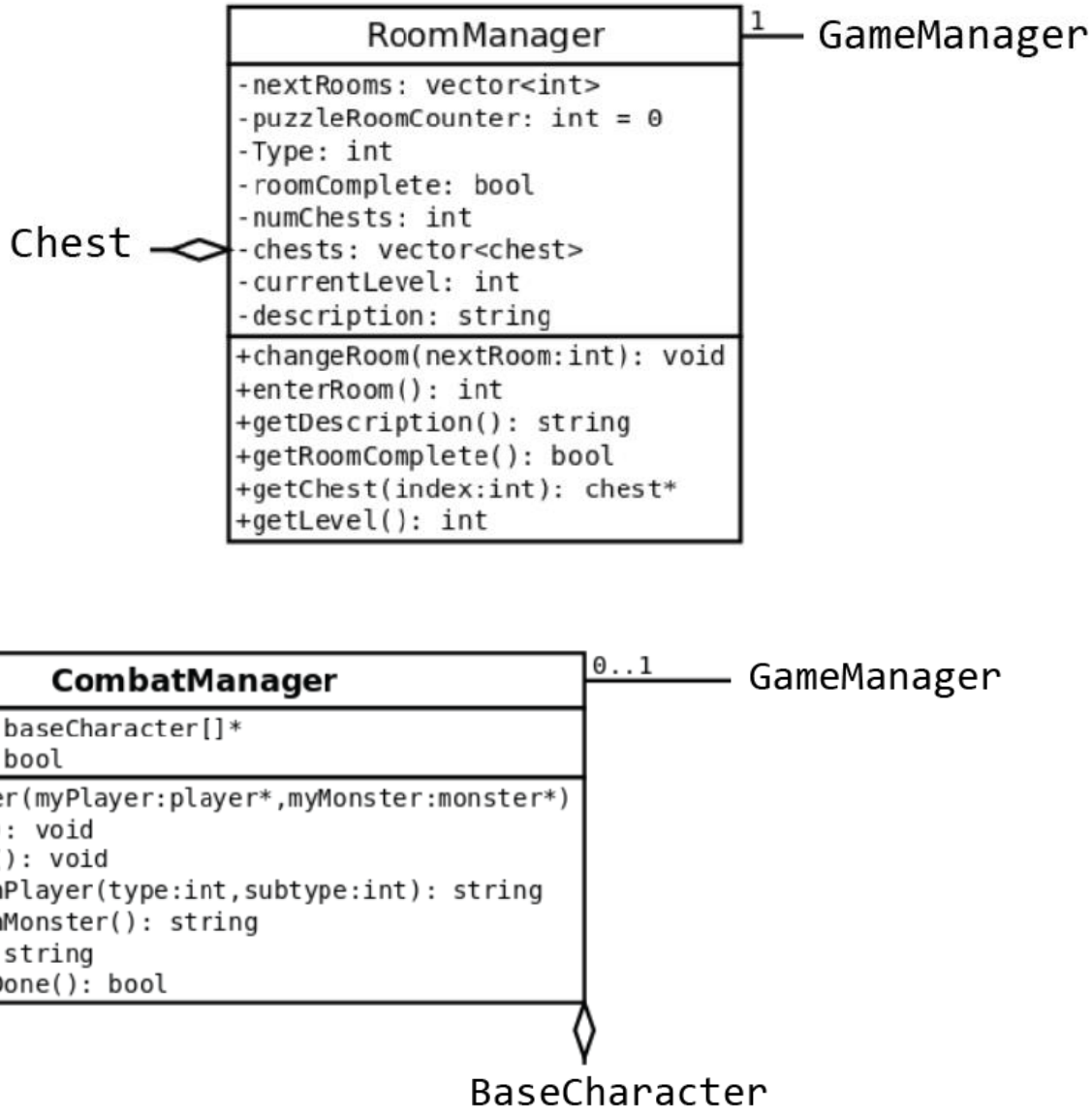
Item
Consumable

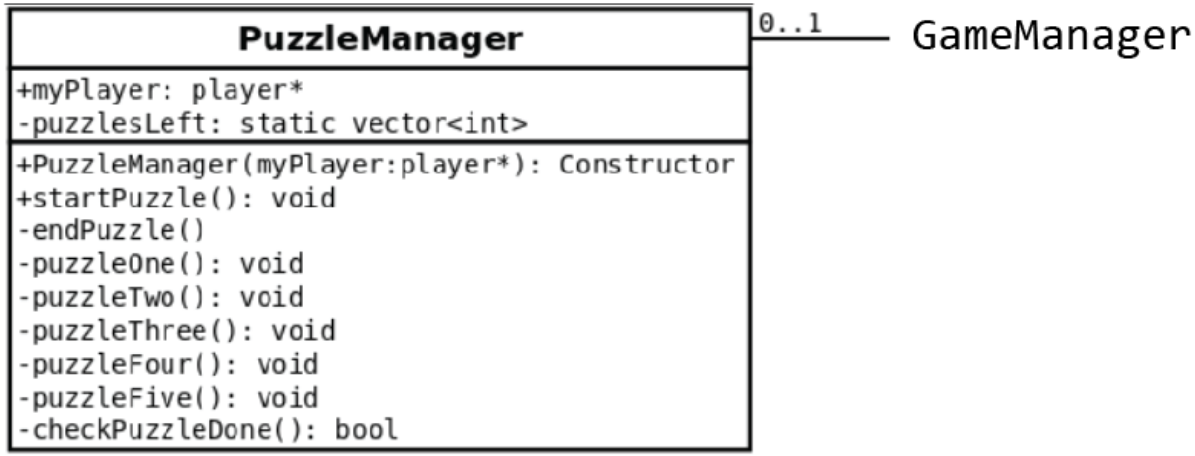


RoomManager

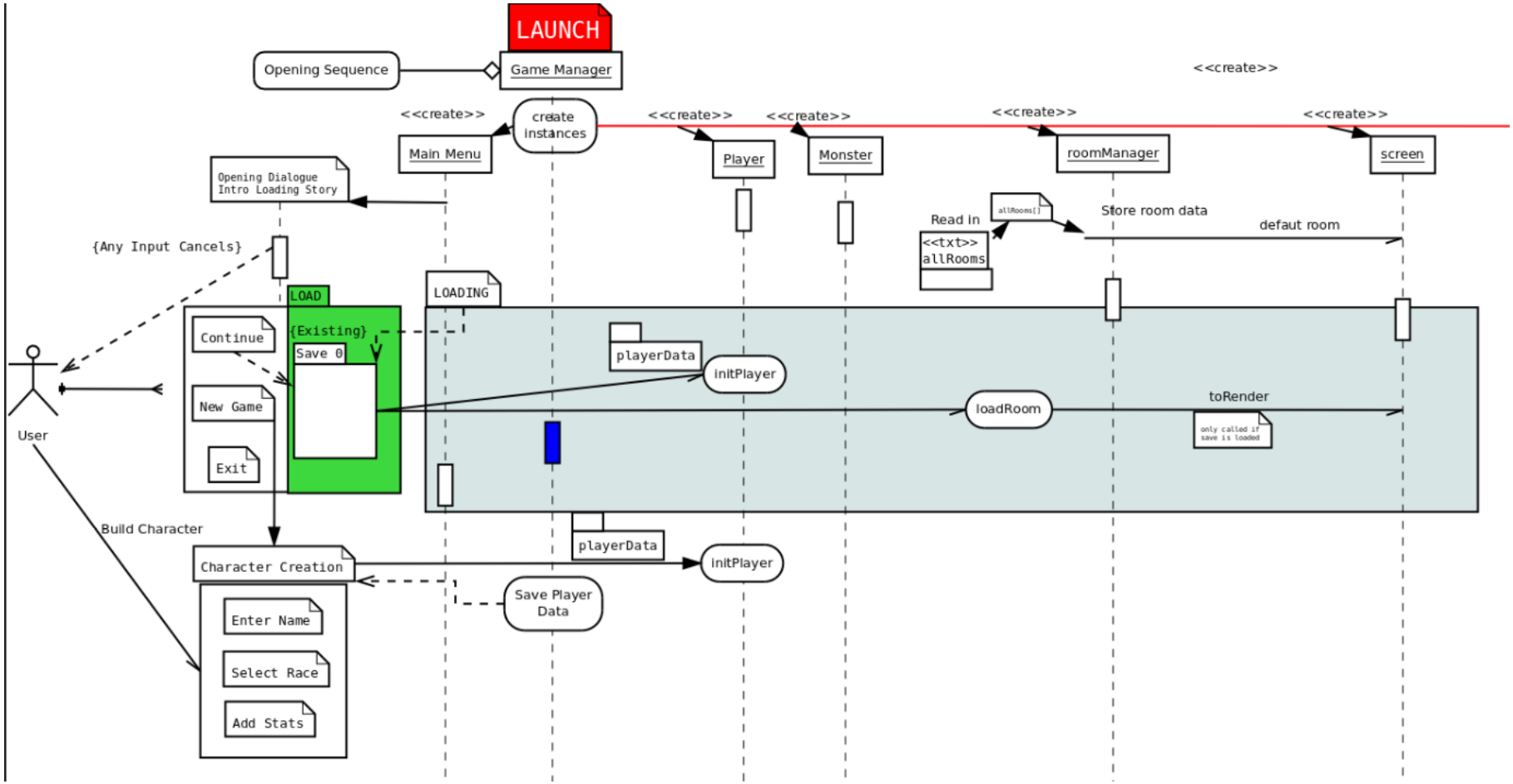


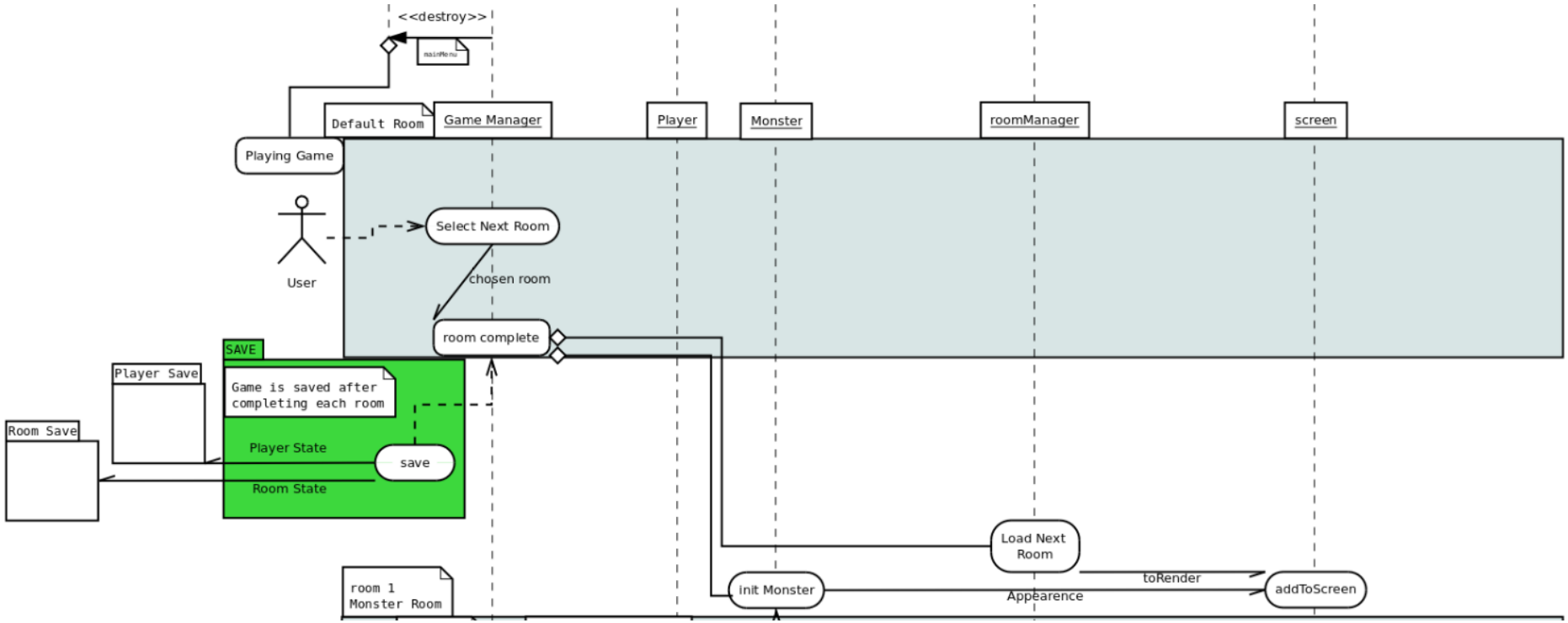
Inventory

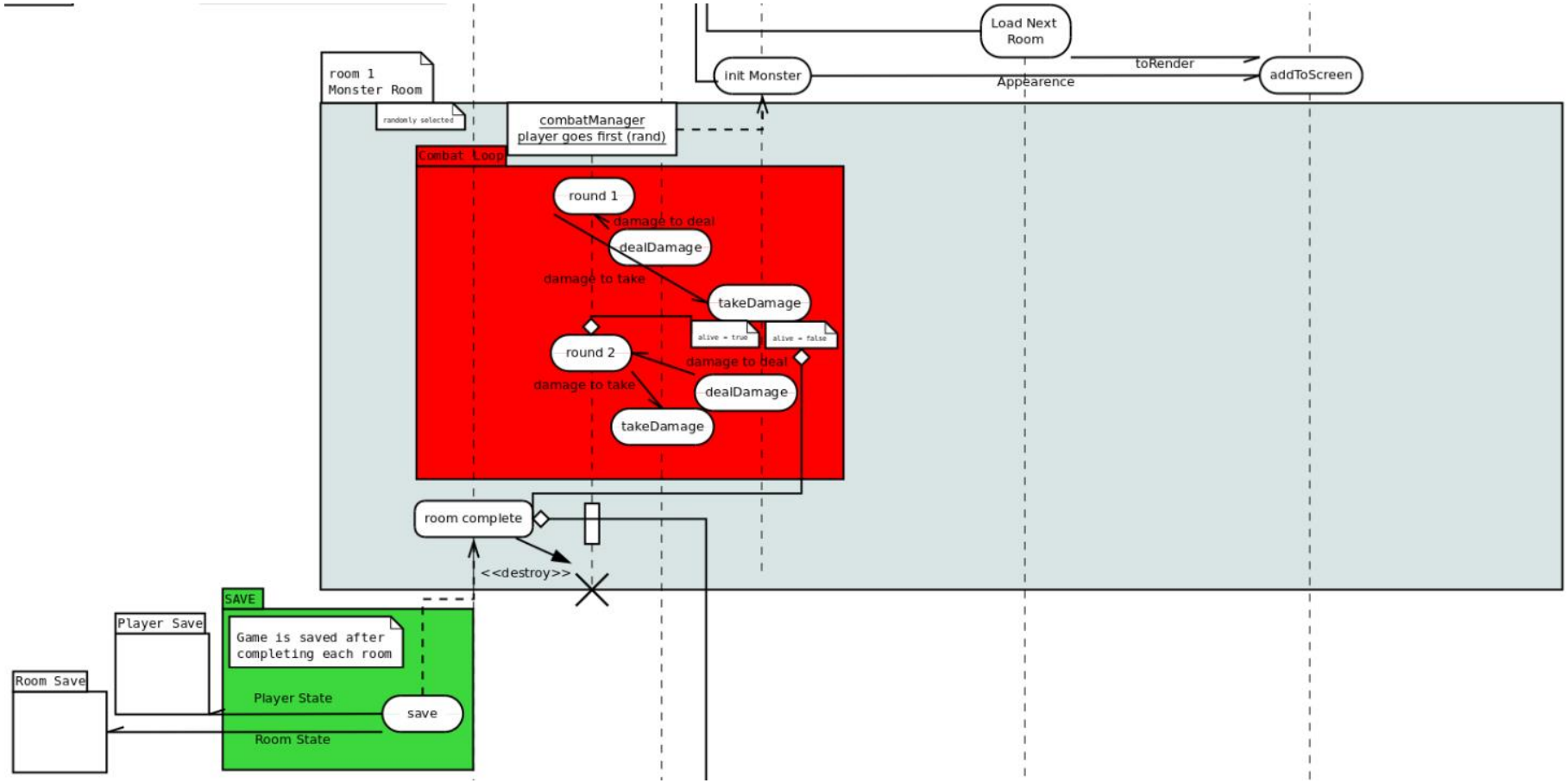

RoomManager
-nextRooms: vector<int>
-puzzleRoomCounter: int = 0
-Type: int
-roomComplete: bool
-numChests: int
-chests: vector<chest>
-currentLevel: int
-description: string
+changeRoom(nextRoom:int): void
+enterRoom(): int
+getDescription(): string
+getRoomComplete(): bool
+getChest(index:int): chest*
+getLevel(): int
1
GameManager
Chest
CombatManager
-fightOrder: baseCharacter[]*
-playerTurn: bool
+combatManager(myPlayer:player*,myMonster:monster*)
+startFight(): void
-startOfTurn(): void
-selectActionPlayer(type:int,subtype:int): string
-selectActionMonster(): string
-endFight(): string
-checkCombatDone(): bool
0..1
GameManager
BaseCharacter

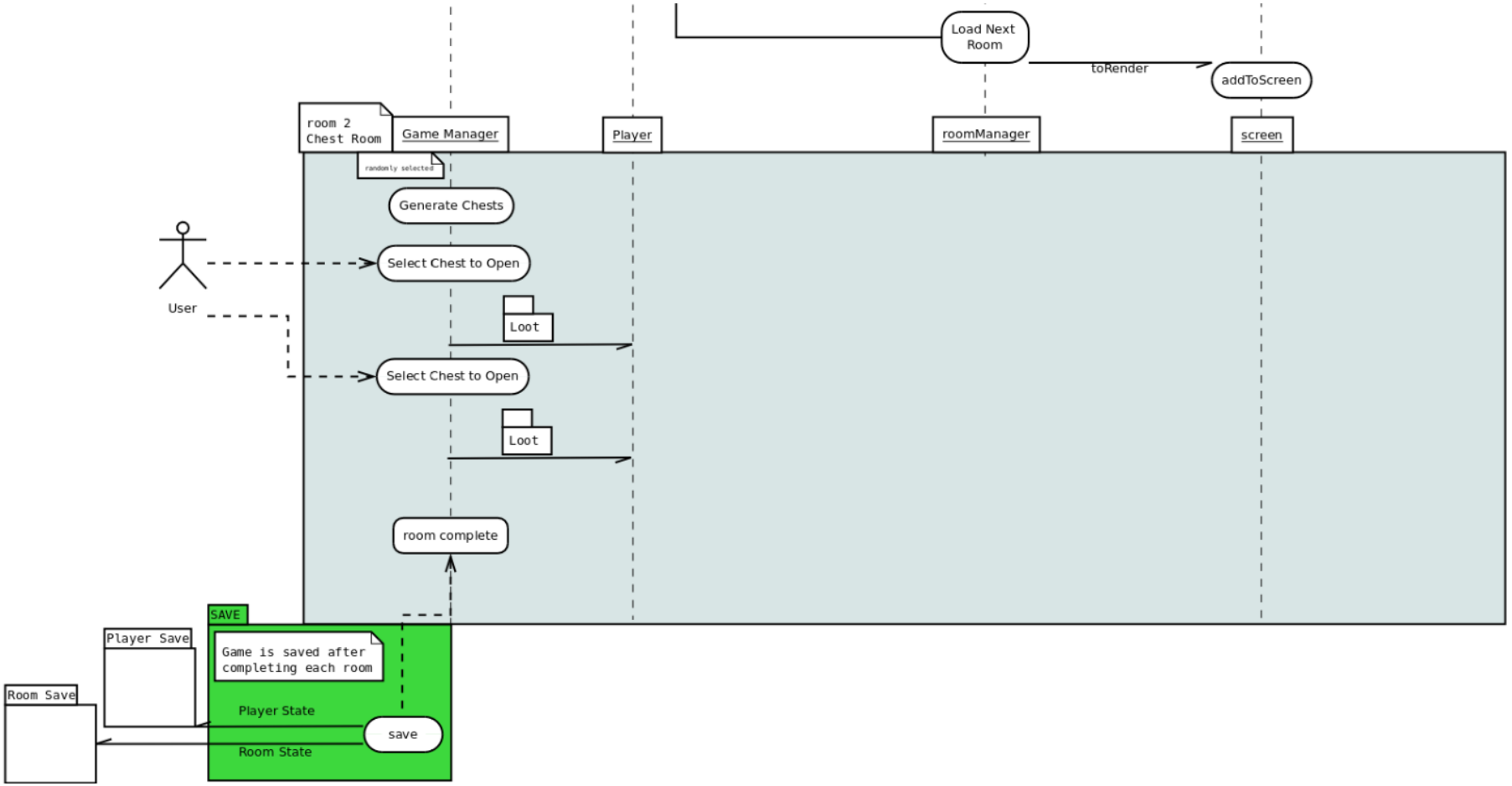

PuzzleManager
+myPlayer: player*
-puzzlesLeft: static vector<int>
+PuzzleManager(myPlayer:player*): Constructor
+startPuzzle(): void
-endPuzzle()
-puzzleOne(): void
-puzzleTwo(): void
-puzzleThree(): void
-puzzleFour(): void
-puzzleFive(): void
-checkPuzzleDone(): bool
0..1
GameManager

5.2. Sequence Diagram




<<destroy>>
mainMenu
Default Room
Game Manager
Player
Monster
roomManager
screen
Playing Game
User
Select Next Room
chosen room
room complete
SAVE
Player Save
Game is saved after completing each room
Room Save
Player State
save
Room State
Load Next Room
toRender
room 1 Monster Room
init Monster
Appearence
addToScreen


Load Next Room
toRender
init Monster
Appearence
addToScreen
room 1
Monster Room
randomly selected
combatManager
player goes first (rand)
Combat Loop
round 1
damage to deal
dealDamage
damage to take
takeDamage
alive = true
alive = false
round 2
damage to deal
damage to take
dealDamage
takeDamage
room complete
<<destroy>>
SAVE
Player Save
Game is saved after completing each room
Room Save
Player State
save
Room State


Load Next Room
toRender
addToScreen
room 2
Chest Room
Game Manager
Player
roomManager
screen
randomly selected
Generate Chests
User
Select Chest to Open
Loot
Select Chest to Open
Loot
room complete
SAVE
Player Save
Game is saved after completing each room
Room Save
Player State
save
Room State