CPSC 2720 [FALL 2021]

Dragon Slayer

Team Garrus

Riley Morse, Samira Azimi, Tinuola Alagbe

December 7, 2021

TABLE OF CONTENTS

INTRODUCTION

Our game will be D&D inspired role play game where the player will be able to choose their class such as a warrior, wizard, tank, and have certain characteristics based off this. Players will start the game in a set location and from there will make text-based choices to playout the story line. Each choice that is made will result in a different outcome and will result in changes to the players experience, level, gold, and items. Players will interact with different non-player characters through actions such as buying, selling, attacking, etc. The objective is the game is to make it to the end and complete an assigned task while you are still alive.

PROJECT MANAGEMENT

TEAM ROLES

Team Member	**Design - Draft**	**Design – Final**	**Implementation - Basic**	**Implementation - Final**
Riley	Phase Lead	Design Lead	QA Lead	Reporting Lead
Darion	Design Lead	QA Lead	Reporting Lead	Phase Lead
Samira	QA Lead	Reporting Lead	Phase Lead	Design Lead
Tinuola	Reporting Lead	Phase Lead	Design Lead	QA Lead

TEAM ROLE RESPONSIBILITIES

Phase Lead

- Understands what needs to be accomplished in the phase.
- Asks questions/clarification from instructor(s) on behalf of the group.
- Identifies (in collaboration with the team) tasks to be done and distributes them.
- Identify problems and lead group discussions about how to solve them.
- Makes sure the group is on track for meeting deadlines.
- Focuses on the phase wholistically

Design Lead

- Propose and collect design ideas from the group.
- Create or maintain the UML diagrams so they reflect the current design of the software.
- Focus on making sure we are following good software engineering principles.

Quality Assurance Lead

- Ensure that the team is making a quality software product.
- Review pull requests to the master branch to ensure that they meet the team's quality standards.
- Creates a testing plan / schedule.
- Implement tests.

Reporting Lead

- Organizes the work to be done for the report(s).
- Collects team's contributions to the reports (e.g. design diagrams from Design Lead).
- Records the team's ideas / action items during meetings.
- Writes (in collaboration with the team) the team report.

RISK MANAGEMENT

Requirements/Design/Estimation

1. The team planned a project that is too large.
   If the scope of the project becomes unmanageable, we will meet as a group and decide how we can scale the project back to complete it in a way that ensures the work we do is quality.
2. The team underestimated how long parts of the project would take.
   If this happens, we will make sure to communicate this with other members in the group to ensure we stay on track for deadlines. From there we can discuss ways to move forward with the rest of the project.
3. Major changes to design are needed during implementation.
   We will update the group on what changes need to be made and work together to ensure that the new implementation will run smoothly. We will update our documentation to be inline with our new changes.

People

1. Addition or loss of team member.
   At this point the group will need to reassess what needs to be done to either catch a new member up to speed on what we have worked on so far or discuss how we can evenly distribute tasks that are unassigned due to the loss of a member.
2. Unproductive team member(s).
   As a group we will continue to check in on progress led by the phase lead. If a member is seemingly unproductive the phase lead will contact them to ask why this is the case and offer possible solutions to get members back on track.
3. Team member(s) lacking expected technical background (e.g. don't know C++).
   If this happens, we need to make sure to communicate our struggles to the team, and from there we can work together to help supplement the lack of technical background or provide resources that will assist the struggling member.
4. Major life events (e.g. long-term illness, death of family member, birth of a child).
   In the case that this happens, the group will need to have a meeting to discuss redistribution of tasks to ensure that deadlines are met.

Learning & Tools

1. Inexperience with new tools.
   Members with little experience with new tools can communicate this and from there the group will be able to provide resources.
2. Learning curve for tools steeper than expected.
   From this point we will assess if the tool is worth using and if there are any alternative methods to ensure we can stay on track to complete the project.

DEVEVLOPMENT PROCESS

CODE REVIEW PROCESS

The quality assurance lead will review code before committing it to the master branch to ensure that no conflicts occur.

COMMUNICATION TOOLS

We will primarily be using Discord to communicate over the course of the project. This will be where we discuss meeting times, communicate any issues we are having, and share resources.

CHANGE MANAGEMENT

The quality assurance lead will manage changes to the software and merge these changes to the main branch at the appropriate time. Any issues with this process will be communicated to team members to ensure we are all on the same page.

SOFTWARE DESIGN

DESIGN – CLASS DIAGRAMS

Refer to figures 1-4

DESIGN – SEQUENCE DIAGRAMS

Refer to figures 5 & 6

CLASS DESCRIPTIONS

Character

Holds player values such as name, class, health, inventory, etc. A new player will be created when the game begins based off whatever class the user chooses. NPCs are also characters since much of their functionality is the same as a player.

CharacterClass

Holds class related variables such as strength, charisma, defence related to the class the player selects. Also holds a vector of abilities that the character will have access to.

Abilities

Stores information on each ability's attributes such as name, and the integer amount the ability changes.

Item

Items change player values such as health, strength, defence, or intelligence.

Game

Gamestate will be used for most of the functionality of our game. It will store the Player, the current location, and options available to the user.

Location

The location consists simply of the name of the location, a bool deciding whether the location is safe or not, and a vector of NPC's that are currently in that location that the player can interact with.

NPC

Holds information to do with characters that the player can interact with inside of the game. Some NPCs will be friendly and assist the player along their journey while others may try to stop you in your tracks.

APPENDICES

APPENDIX A: FIGURES AND TABLES

Figure 1 (Overview)

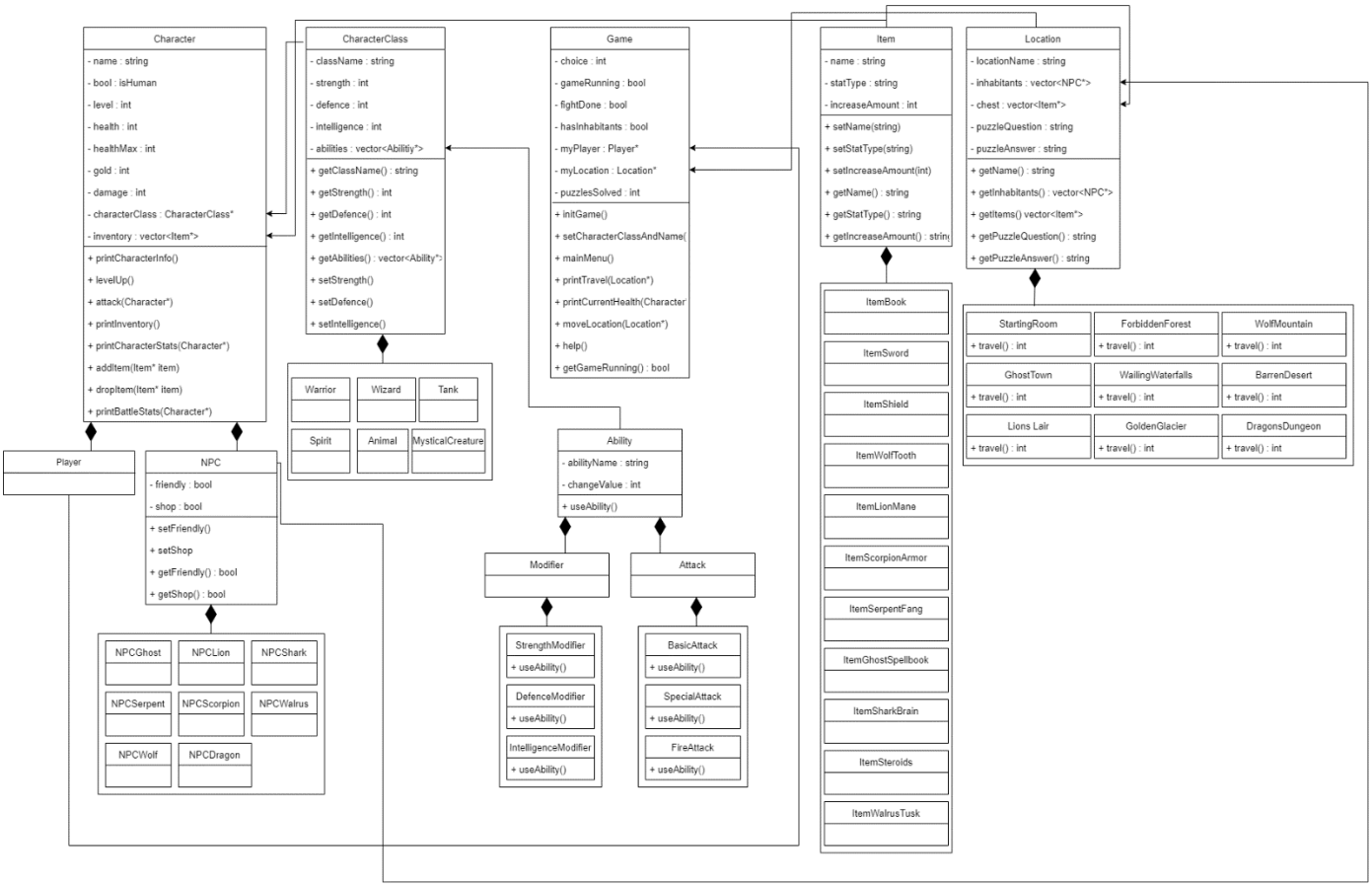


Figure 2

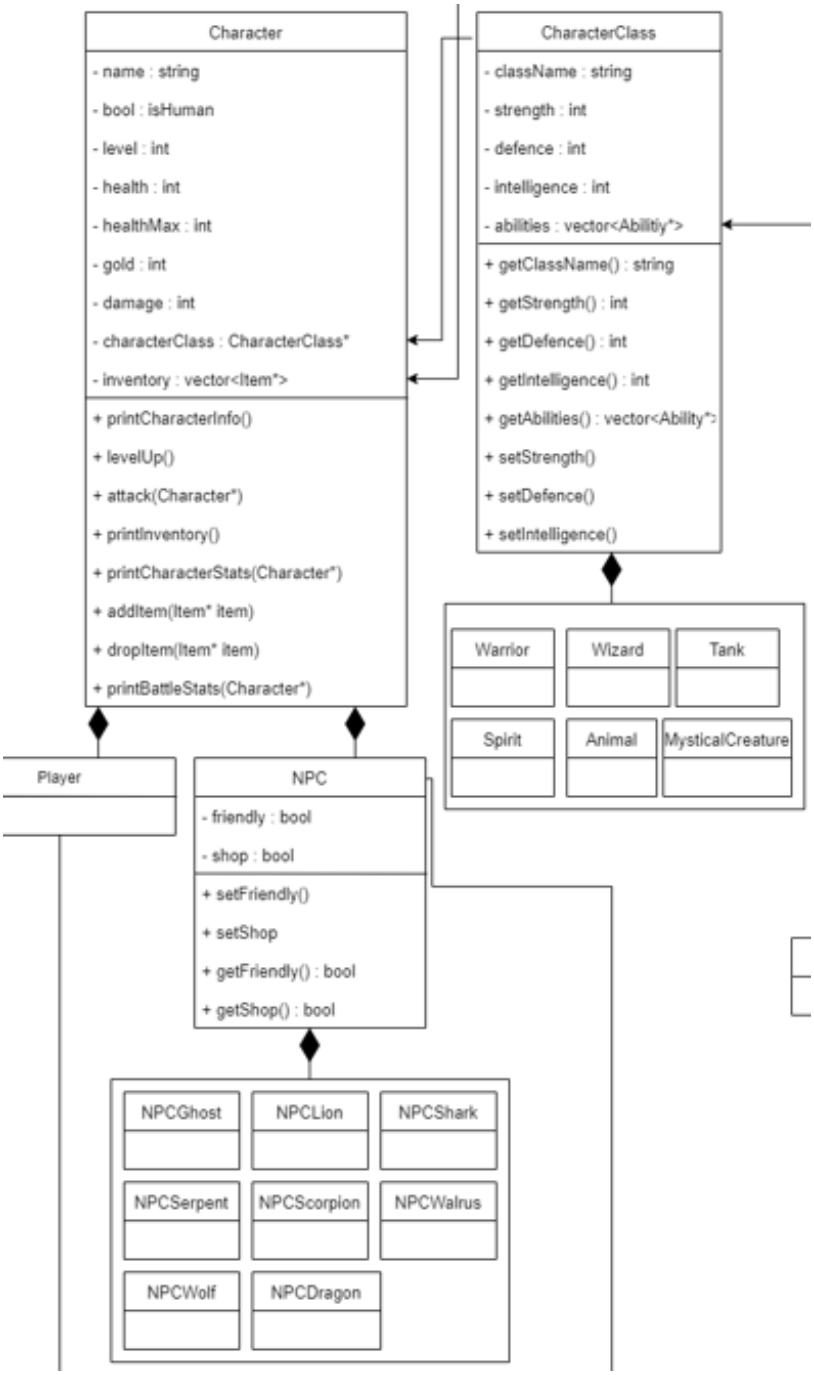

Character
- name : string
- bool : isHuman
- level : int
- health : int
- healthMax : int
- gold : int
- damage : int
- characterClass : CharacterClass*
- inventory : vector<Item*>
+ printCharacterInfo()
+ levelUp()
+ attack(Character*)
+ printInventory()
+ printCharacterStats(Character*)
+ addItem(Item* item)
+ dropItem(Item* item)
+ printBattleStats(Character*)
CharacterClass
- className : string
- strength : int
- defence : int
- intelligence : int
- abilities : vector<Ability*>
+ getClassName() : string
+ getStrength() : int
+ getDefence() : int
+ getIntelligence() : int
+ getAbilities() : vector<Ability*>
+ setStrength()
+ setDefence()
+ setIntelligence()
Warrior
Wizard
Tank
Spirit
Animal
MysticalCreature
Player
NPC
- friendly : bool
- shop : bool
+ setFriendly()
+ setShop
+ getFriendly() : bool
+ getShop() : bool
NPCGhost
NPCLion
NPCShark
NPCSerpent
NPCScorpion
NPCWalrus
NPCWolf
NPCDragon

Figure 3

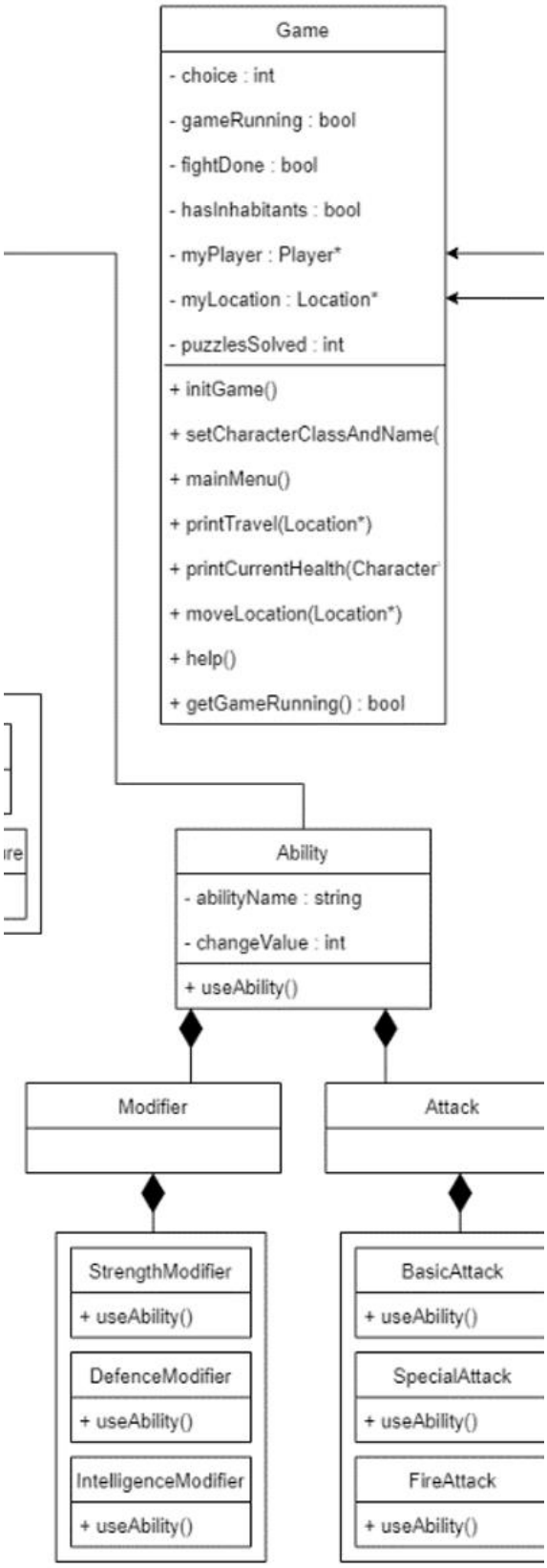

Game
- choice : int
- gameRunning : bool
- fightDone : bool
- hasInhabitants : bool
- myPlayer : Player*
- myLocation : Location*
- puzzlesSolved : int
+ initGame()
+ setCharacterClassAndName(
+ mainMenu()
+ printTravel(Location*)
+ printCurrentHealth(Character
+ moveLocation(Location*)
+ help()
+ getGameRunning() : bool
Ability
- abilityName : string
- changeValue : int
+ useAbility()
Modifier
Attack
StrengthModifier
+ useAbility()
DefenceModifier
+ useAbility()
IntelligenceModifier
+ useAbility()
BasicAttack
+ useAbility()
SpecialAttack
+ useAbility()
FireAttack
+ useAbility()

Figure 4

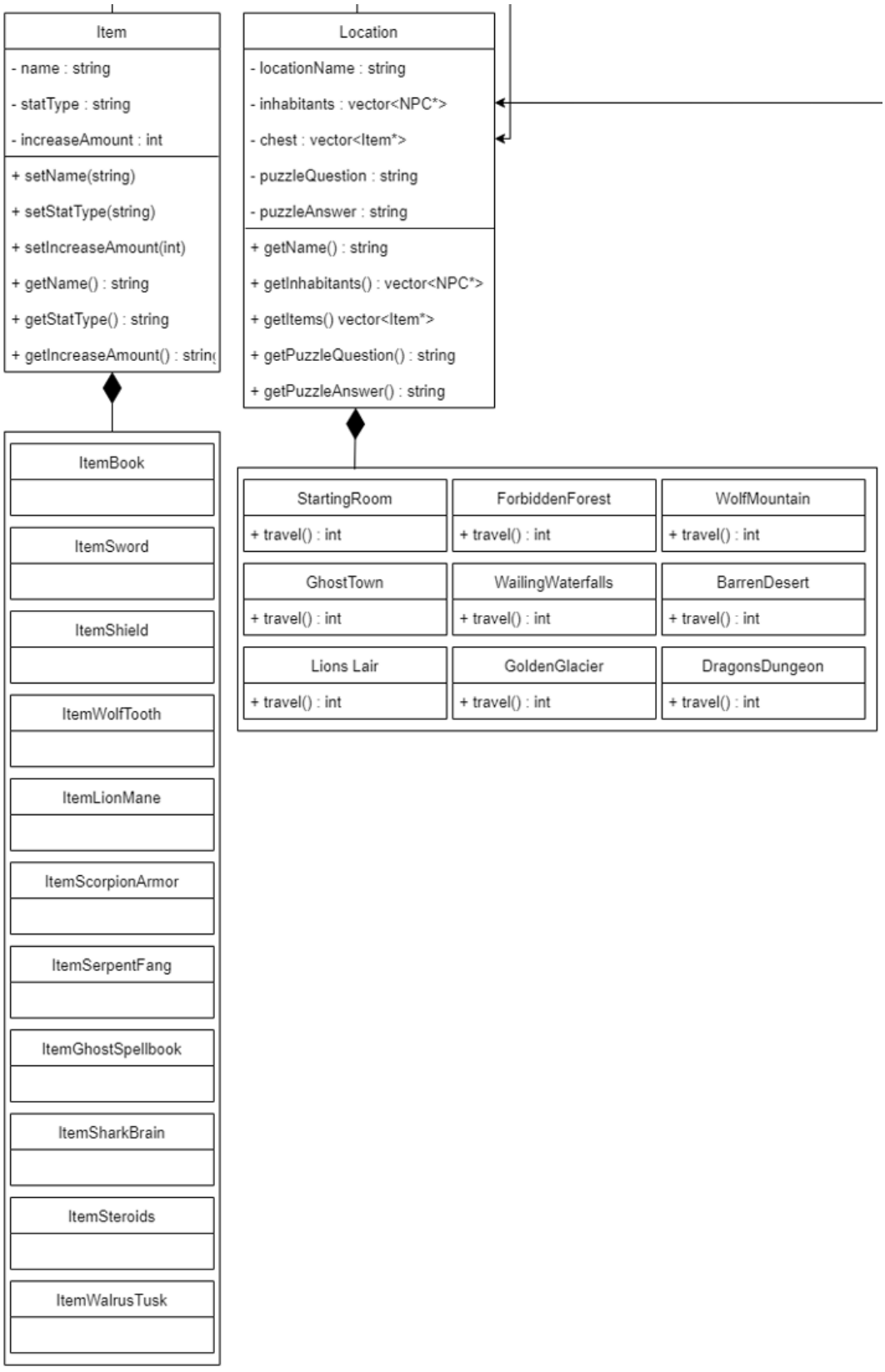
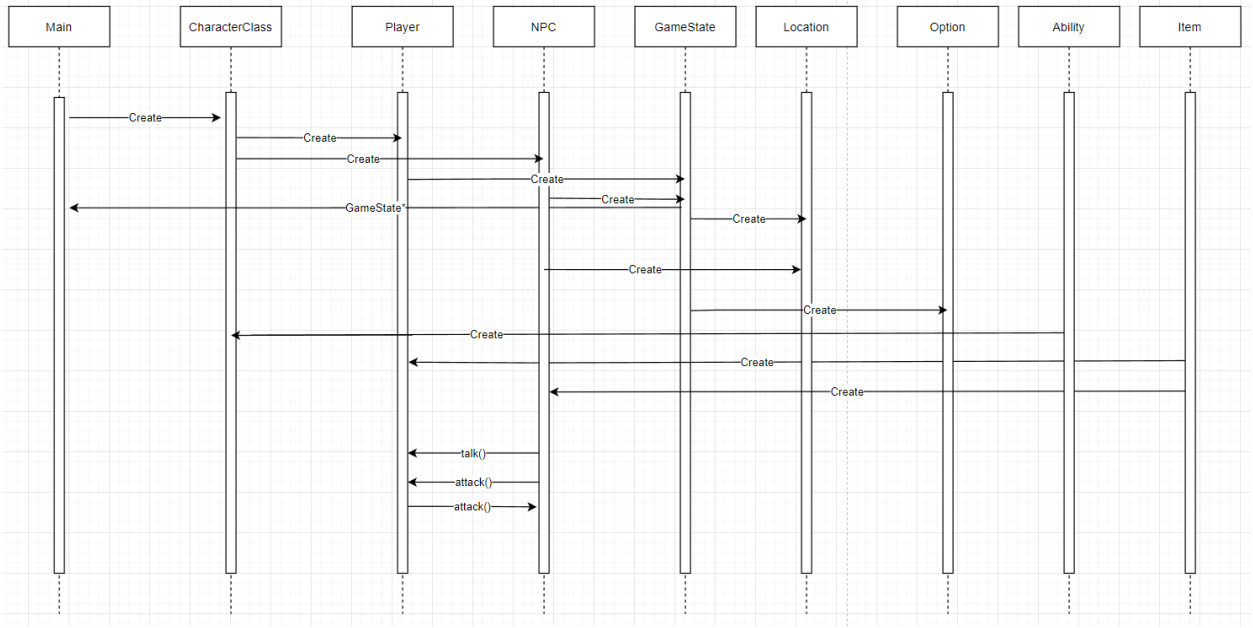
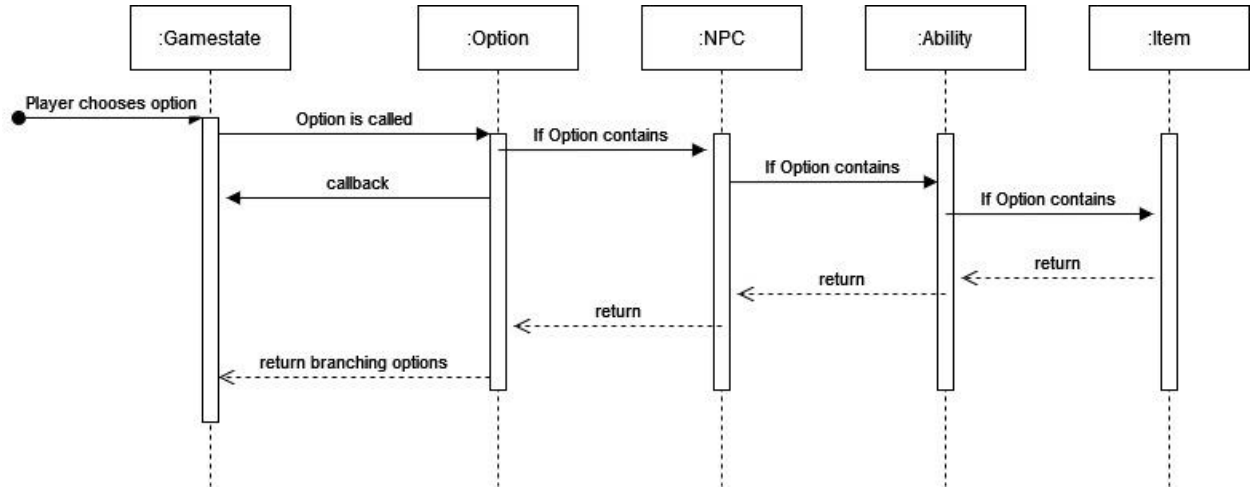

Item
- name : string
- statType : string
- increaseAmount : int
+ setName(string)
+ setStatType(string)
+ setIncreaseAmount(int)
+ getName() : string
+ getStatType() : string
+ getIncreaseAmount() : strin
Location
- locationName : string
- inhabitants : vector<NPC*>
- chest : vector<Item*>
- puzzleQuestion : string
- puzzleAnswer : string
+ getName() : string
+ getInhabitants() : vector<NPC*>
+ getItems() vector<Item*>
+ getPuzzleQuestion() : string
+ getPuzzleAnswer() : string
ItemBook
ItemSword
ItemShield
ItemWolfTooth
ItemLionMane
ItemScorpionArmor
ItemSerpentFang
ItemGhostSpellbook
ItemSharkBrain
ItemSteroids
ItemWalrusTusk
StartingRoom
+ travel() : int
ForbiddenForest
+ travel() : int
WolfMountain
+ travel() : int
GhostTown
+ travel() : int
WailingWaterfalls
+ travel() : int
BarrenDesert
+ travel() : int
Lions Lair
+ travel() : int
GoldenGlacier
+ travel() : int
DragonsDungeon
+ travel() : int

Figure 5/6


Main
CharacterClass
Player
NPC
GameState
Location
Option
Ability
Item
Create
Create
Create
Create
Create
GameState*
Create
Create
Create
Create
Create
Create
talk()
attack()
attack()


:Gamestate
:Option
:NPC
:Ability
:Item
Player chooses option
Option is called
If Option contains
If Option contains
If Option contains
callback
return
return
return
return branching options