**Oh Ship**

Requirements Analysis Document

Rae Vroman, Gilad Berman, Zachary Cox

**Table of Contents**

1. Introduction-**Rae**
2. Proposed System-**Rae**
   1. Overview
   2. Functional Requirements
   3. Non-Functional Requirements
   4. System Models
      1. Use case model- **Rae**
         1. Use Case Diagrams
         2. Use Case Descriptions
      2. Object Model- **Zach**
         1. Class Diagram
         2. Class Diagram description
      3. Dynamic Model- **Rae, Zach, Gilad**
         1. Sequence Diagrams
            1. Command Menue Feature

Save Game **Z**

Exit Game **G**

View Map **G**

Load Game **R**

Resume Game **R**

Check Inventory **R**

* + - * 1. Item Feature

Use Item **G**

Pick Up Item **G**

Un-Equip Item **G**

Examine Item **G**

Equip Item **G**

Drop Item **G**

* + - * 1. Combat Feature

Win Fight **R**

Surrender Hint **G**

Escape Fight **G**

Surrender Fight **G**

Lose Fight **R**

Receive Damage **R**

Attack Monster **R**

* + - * 1. Navigation Feature

Go to Room **R**

* + - * 1. Puzzle Feature

Examine Puzzle **Z**

Solve Puzzle Incorrectly **R**

Ignore Puzzle **R**

Request Hint **R**

Solve Puzzle Correctly **Z**

* + - * 1. Room Feature

Enter/Get Room Description **R**

Search Room **R**

* + 1. User interface- **Zach**
       1. Screen mockups

**Section 1: Introduction**

The game “Oh Ship” will be a texted based adventure game with a GUI. The objective will be to navigate your way off a ship that has been attacked by pirates. The player will interact with it’s environment through rooms, items, puzzles and fights.

**Section 2: Proposed System**

2.1 Overview

The proposed system is a text based adventure game with a GUI. The user will be able to click on rooms on the map to travel to them. In these rooms, they will encounter items, fights and puzzles. There will be some puzzles that need to be completed before going on with the game, and some puzzles are optional. The premise of the game is that the user finds themselves alone in a room on a ship. They come to find out that pirates have taken over their ship. Using items that they find, they will solve puzzles and fight pirates until they get to the top of the ship, save the captain and escape on a boat.

2.2 Functional Requirements

* + 1. Deleted Requirements

MF0- Options Menu

NF1- Check Navigation Options

NF2- Return to Past Room

IF5- Give Item

CF1- Combat Structure

CF2- Description of Enemy Attack

CF3- Attack Options

CF5- Weapon Attack

CF6- Defend

NF1- Check Navigation Options

2.2.2 Changed/Updated Requirements

ID: NF2

TITLE: Go to Room

DESC: The user must be able to choose a room on the map to travel to. This will only work if the user has already visited the room, or the room they are attempting to go to is connected to the room they are currently in, and doesn’t have any barriers such as a puzzle or a locked door.

RAT: So the user can have a way to navigate the map.

ID: CF0

TITLE: Lose Fight

DESC: The user must be able to run out of health points and be killed if ten or more points of damage are taken without healing.

RAT: In order to have a way for the user to lose in combat.

ID: CF4

TITLE: Attack Enemy

DESC: The user must be able to attack the enemy each turn. Each attack does 2 damage.

RAT: So that the user can take all the HP from the monster and win the fight.

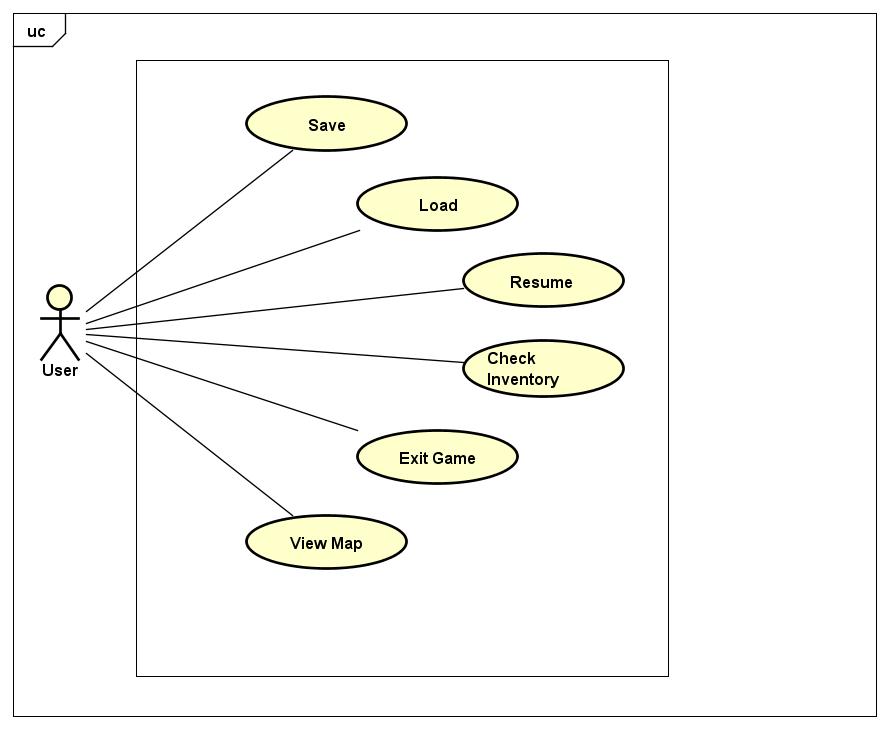
2.3 Non Functional Requirements- No changes made

**2.4 System Models**

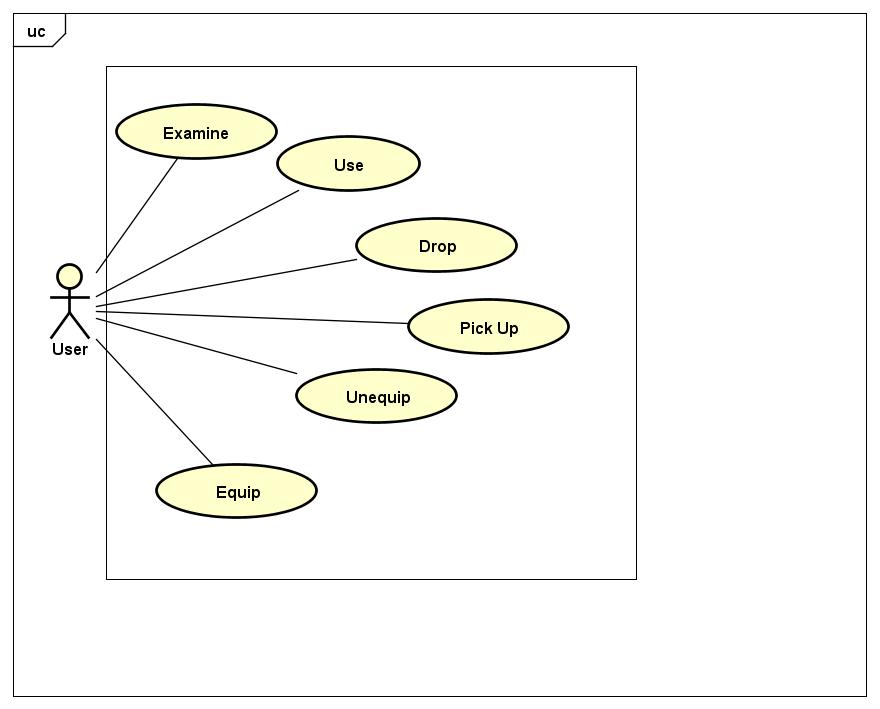
**2.4.1 Use Case Model**

**2.4.1.1 Use Case Diagrams**

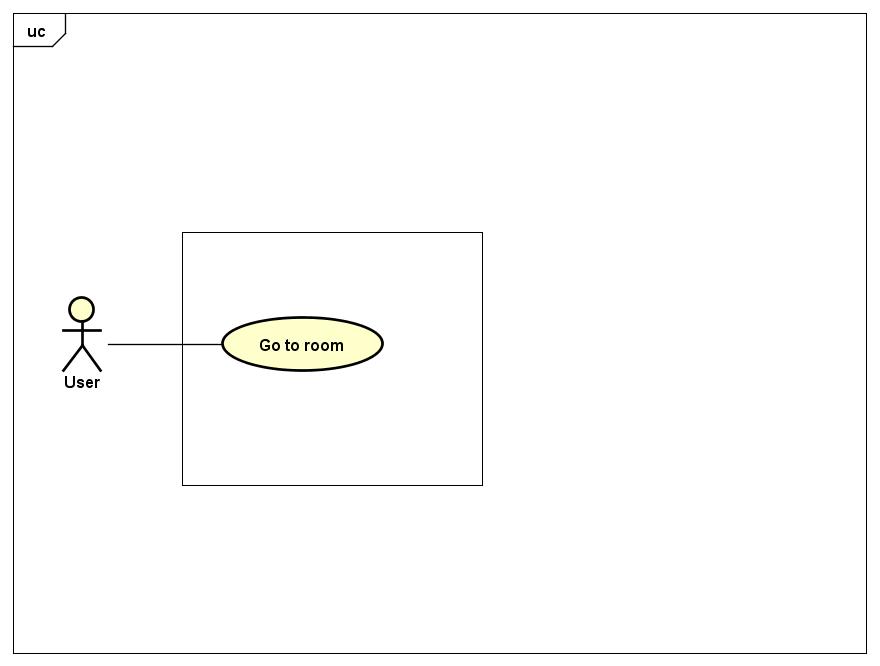
Command Menue Feature



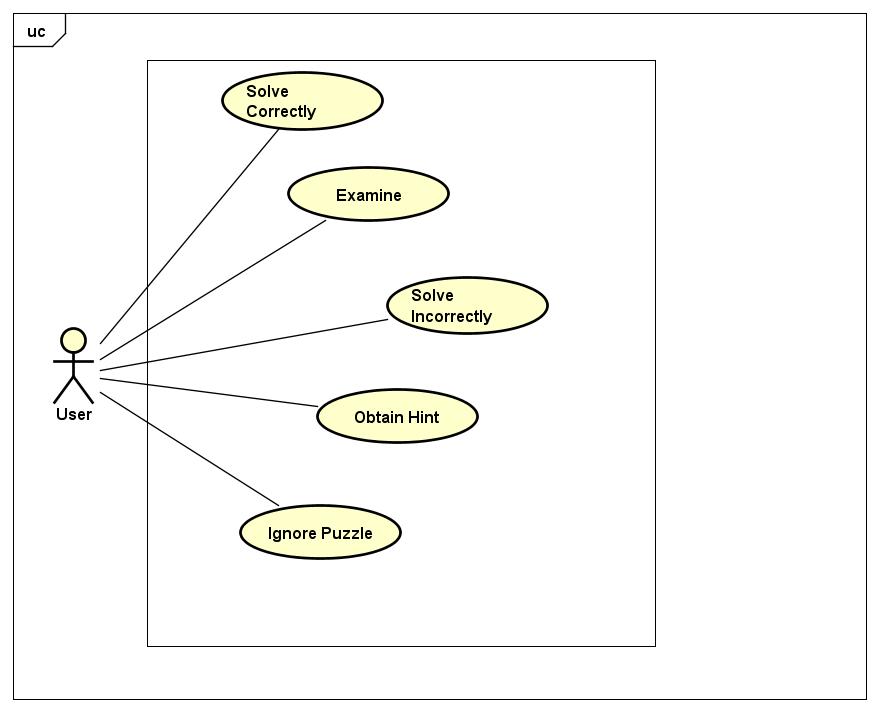
Item Feature



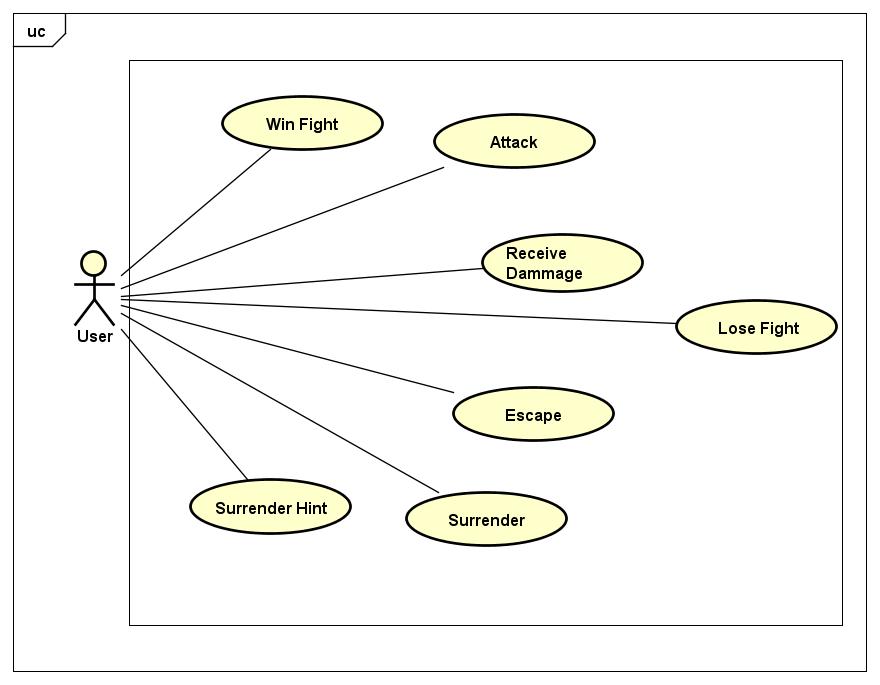
Navigation Feature



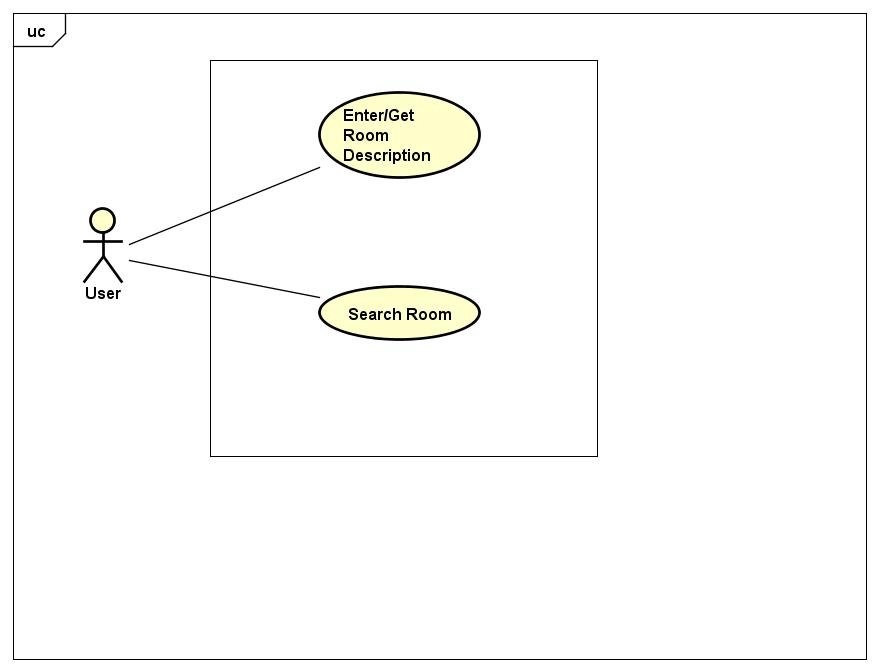
Puzzle Feature



Combat Feature



Room Feature



**2.4.1.2 Use Case Descriptions**

**Command Menu**

The user must be able to click a button to pull up the command menu. From there they can choose to save the game, load a game, resume their game, view the map, check their inventory or exit the game.

**Item**

There are many items throughout the game that the user can interact with. They can choose to pick up the item, examine the item, use the item, equip the item, unequip the item or drop the item,

**Navigation**

This game uses a GUI and will therefore have only one way to navigate. The user will click on a part of the map that they wish to navigate to, and if they have been there before, they can go to that room. If they have not been there before, the room they are attempting to travel to must be connected to the room they are currently in, and it must not need a puzzle or fight to get through.

**Puzzle**

For each puzzle, the player must choose to be able to examine the puzzle, ignore it, solve it, solve it incorrectly, or obtain a hint.

**Combat**

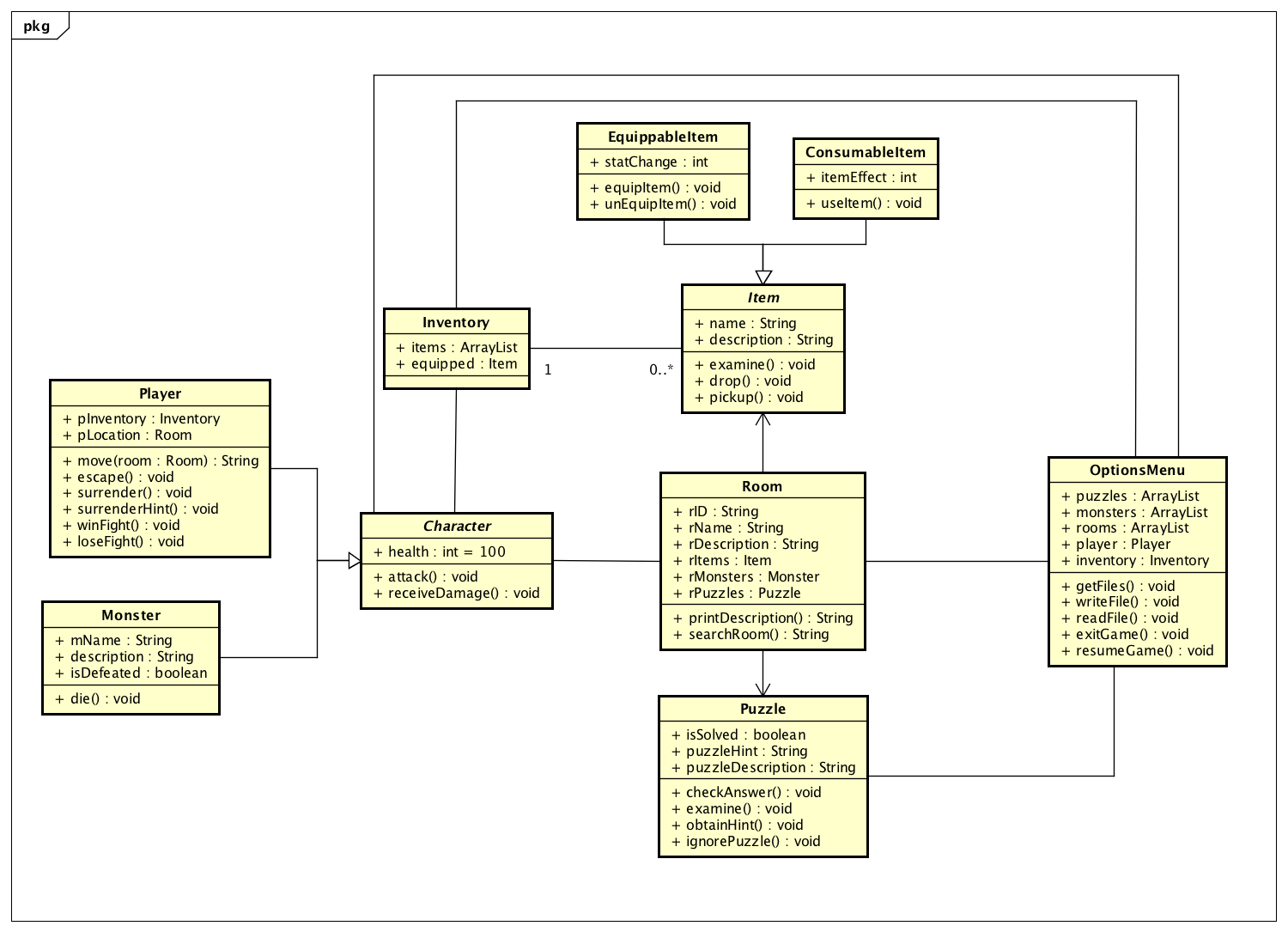
Whenever the player is engaged in combat there are many ways they can interact with the system. The player can attack a monster, win the fight, receive damage, lose a fight, escape the fight, get a hint for surrendering the fight, and surrender the fight.

**Room**

When the player enters a room, they will immediately get a description of the room. They will be able to use the “Search Room” command to search the room for items to collect to help them in the game.

**2.4.2 Object Model**

2.4.2.1 Sequence Diagrams

****

**2.4.2.2 Class Diagram Description**

***Character Class***

The **character** class is an abstract class for the **monster** and **player** class. They share health attribute, as well as basic combat related commands. It has its own inventory class and has access to the Options Menu.

**Item Class**

The **item** class is an abstract class for the **Equippable Item** and **Consumable Item**. They share basic attributes name and description, as well as basic item related commands. The two subclasses were separated to give them different functionality.

**Room Class**

Contains basic attributes related to the room, as well as a puzzle object, monster object, and an item object if applicable. Has a function that retrieves the r\_Item object if one is in that room.

**OptionsMenu Class**

Contains an array list of all objects that have been used thus far. Has functionality to retrieve file names, write the current state to a text file, read a saved state from a text file, as well as simple exit and resume functions.

**Inventory**

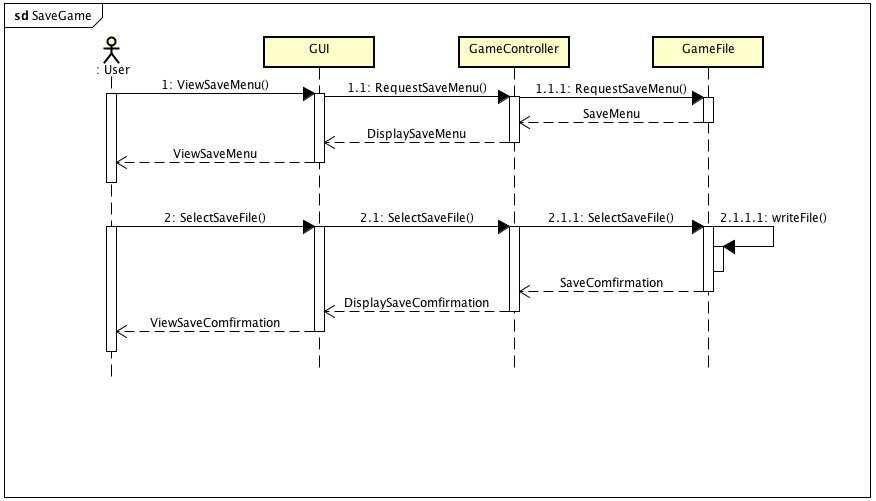
Contains an array list of item objects. An attribute that keeps track of which item object is currently equipped.

**2.4.3 Dynamic Model**

**2.4.3.1 Sequence Diagrams**

**3.1.1 Command Menu Feature**

1.1.1 Save Game Diagram

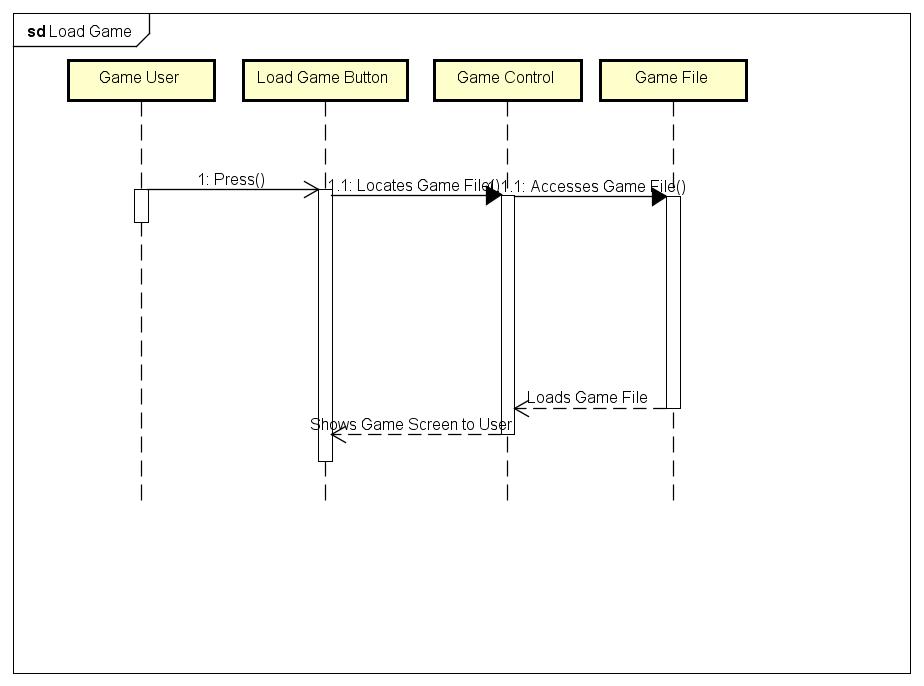


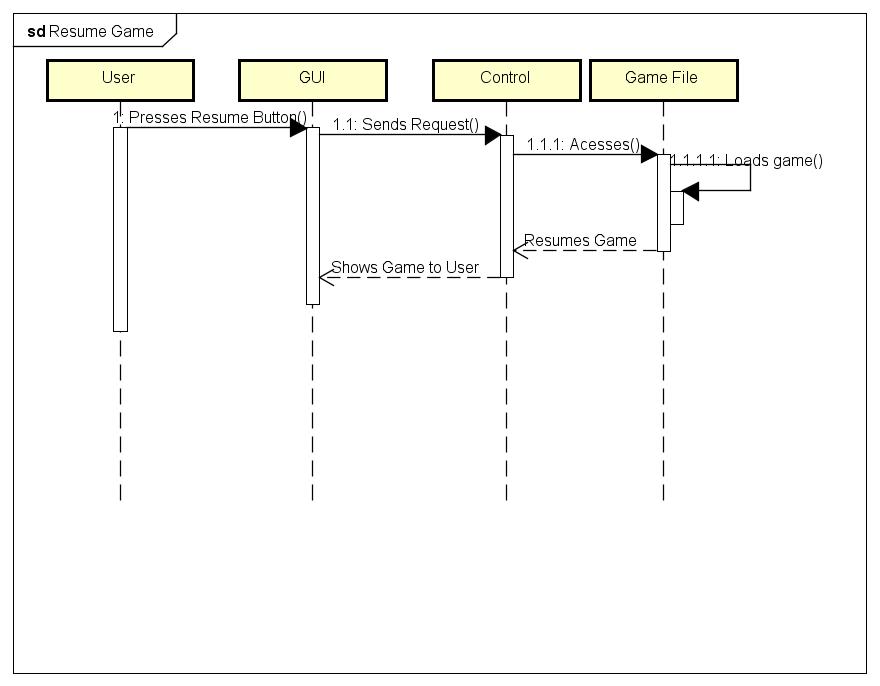
1.1.2 Exit Game Diagram 

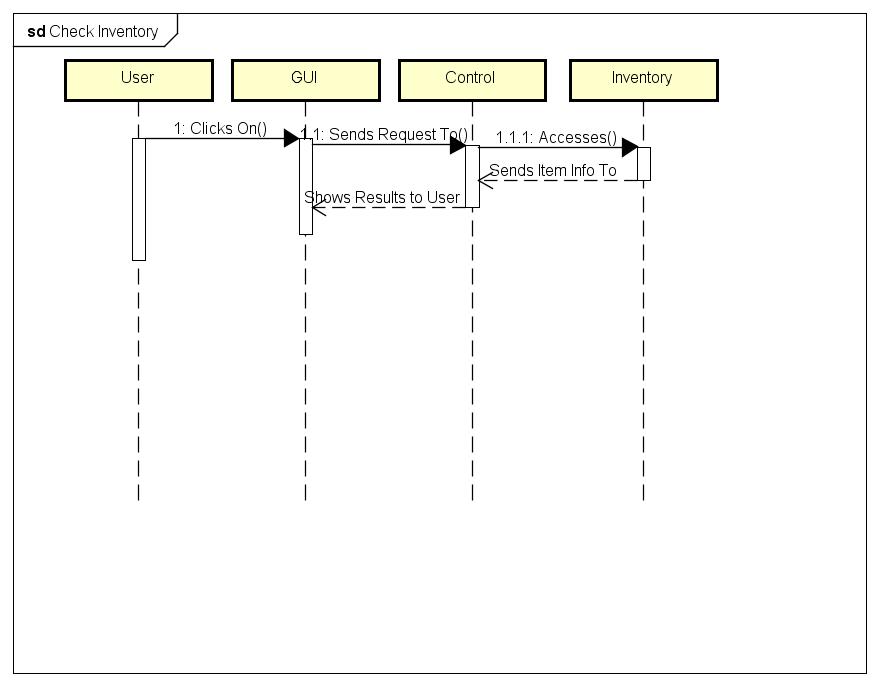
1.1.3 View Map Diagram



1.1.4 Load Game Diagram



1.1.5 Resume Game Diagram 

1.1.6 Check Inventory Diagram

**3.1.2 Item Feature**

1.2.1 Use Item Diagram



1.2.2 Pick Up Item Diagram



1.2.3 Un-Equip Item Diagram



1.2.4 Examine Item Diagram



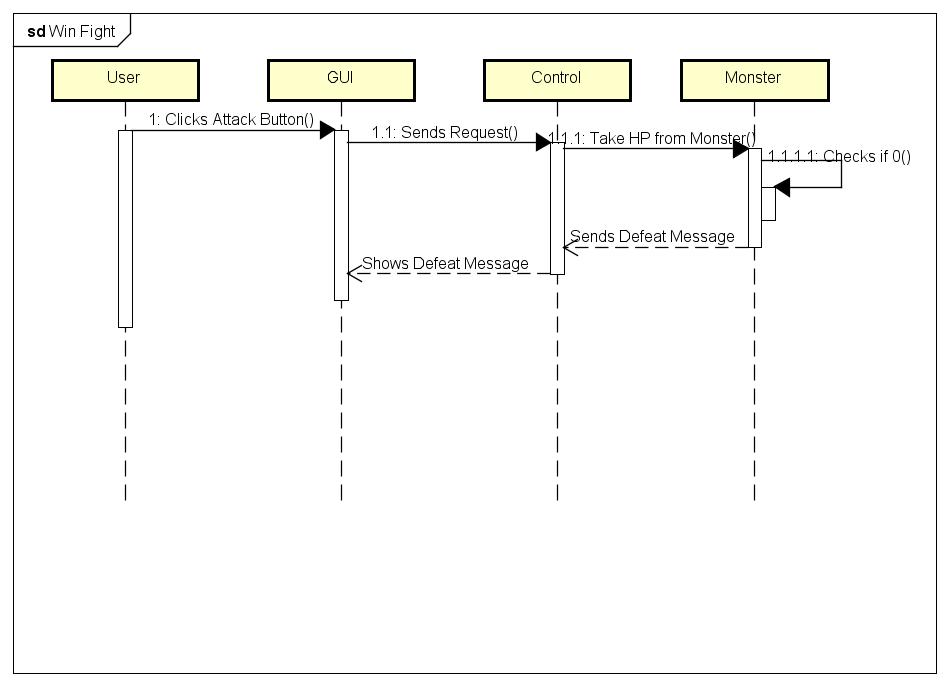
1.2.5 Equip Item Diagram



1.2.6 Drop Item Diagram



**3.1.3 Combat Feature**

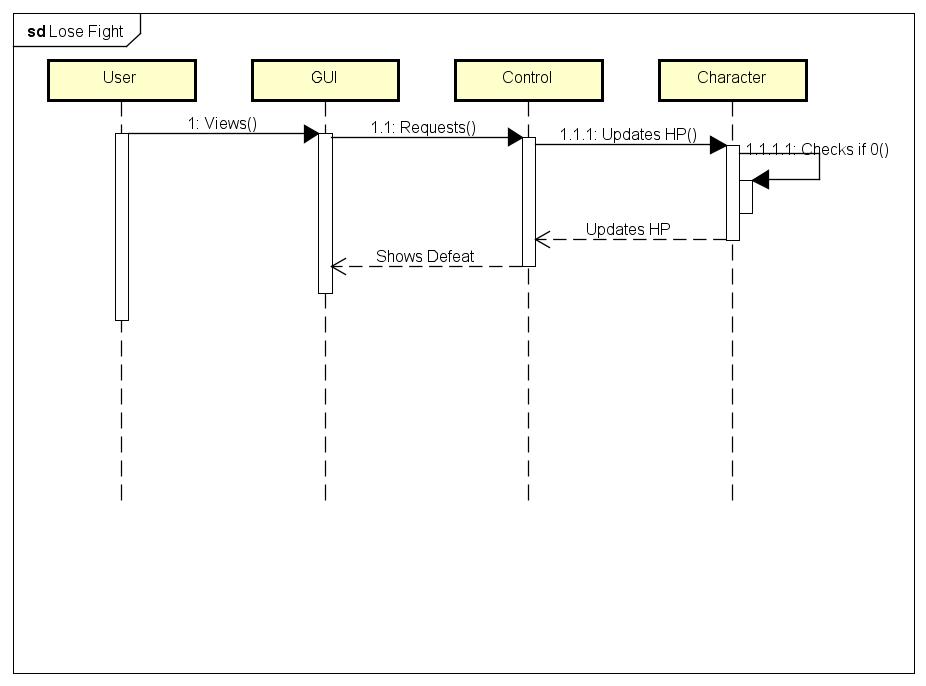
1.3.1 Win Fight Diagram 

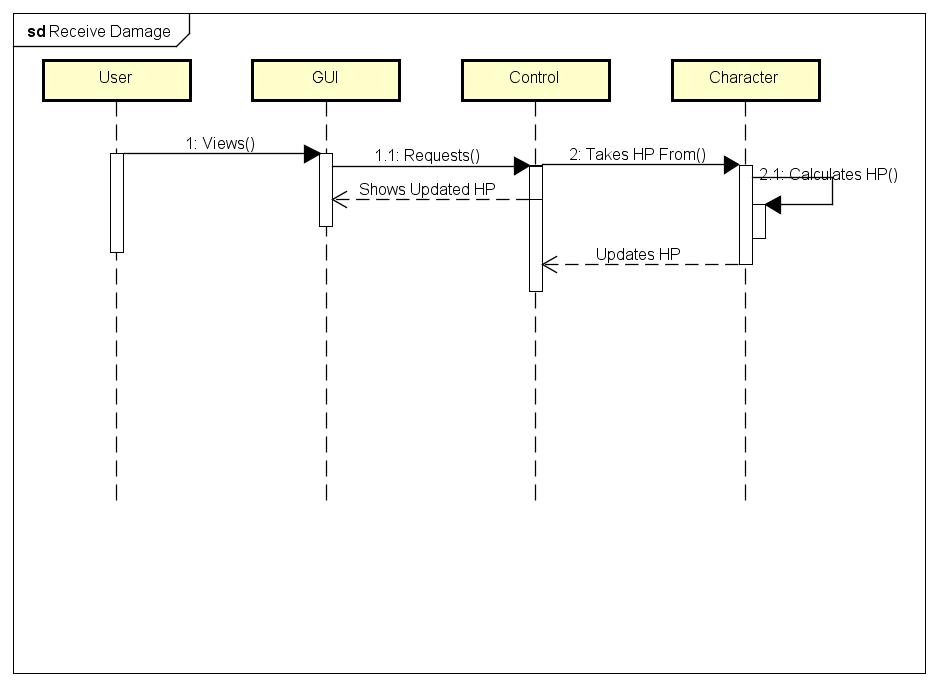
1.3.2 Surrender Hint Diagram

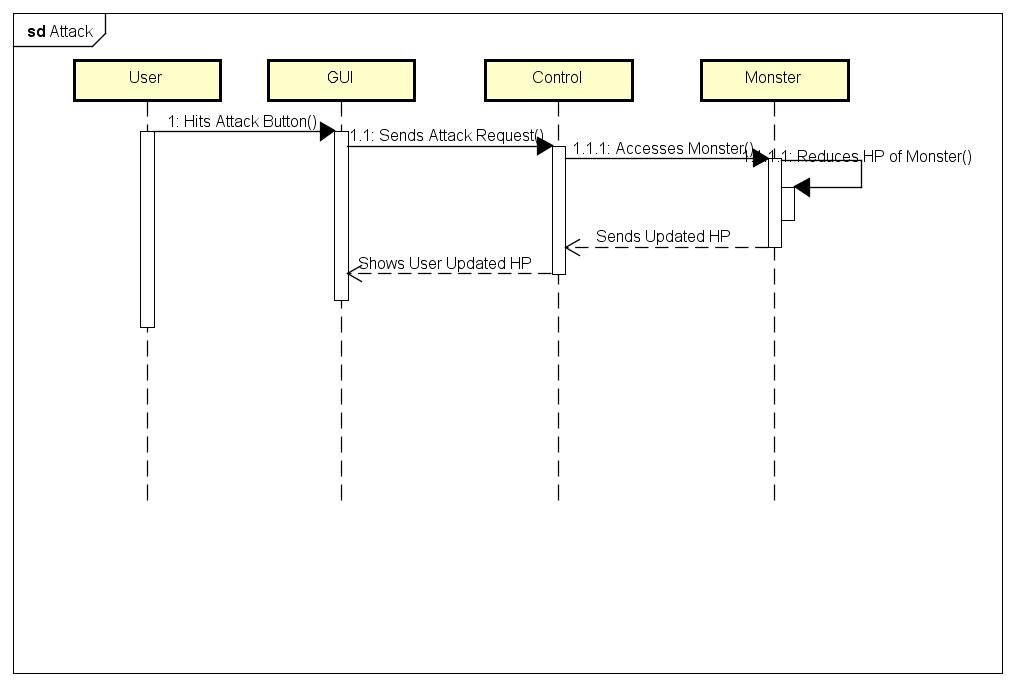


1.3.3 Escape Fight Diagram

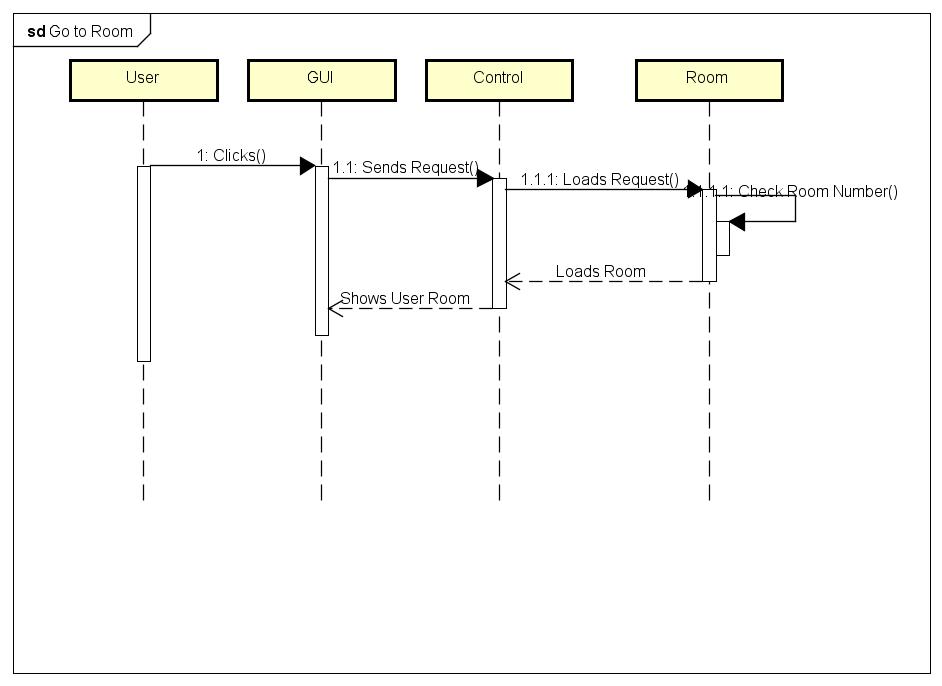
1.3.4 Surrender Fight Diagram

1.3.5 Lose Fight Diagram

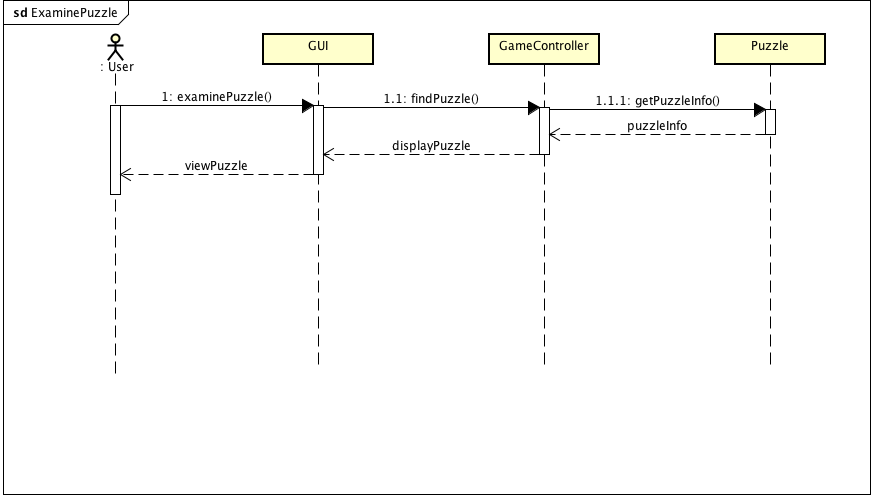
1.3.6 Receive Damage Diagram

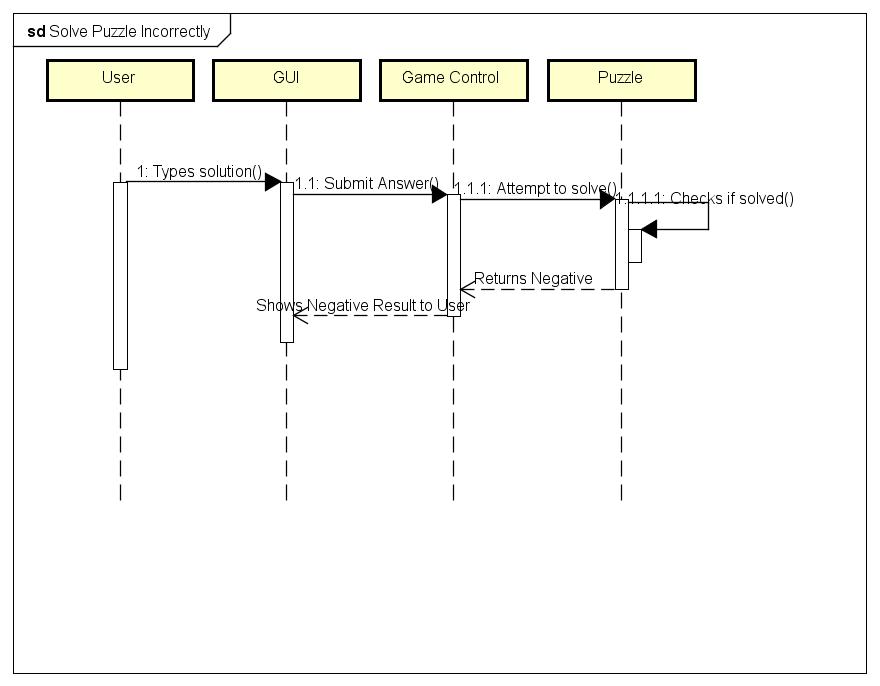
1.3.7 Attack Monster Diagram 

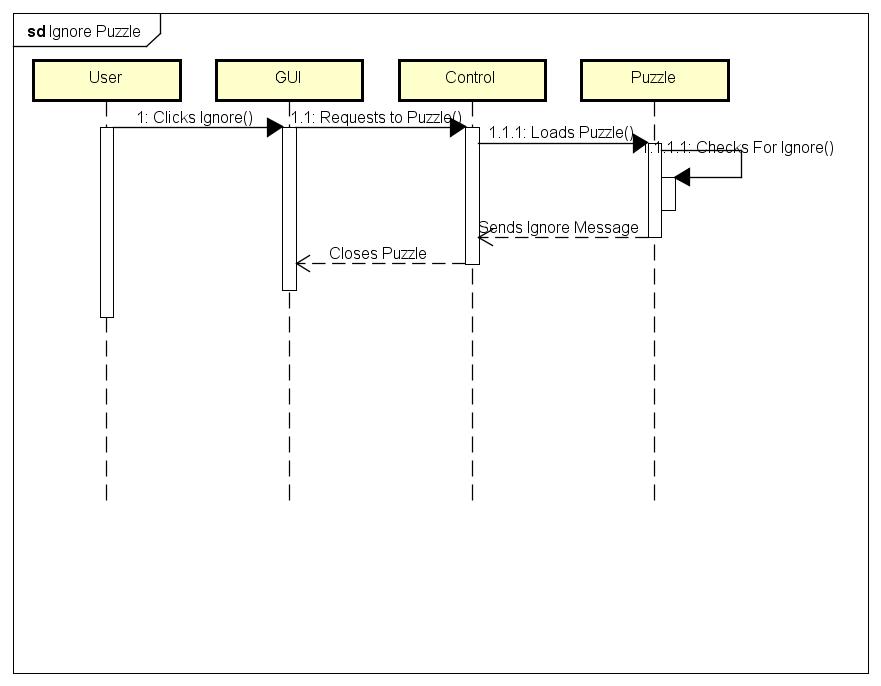
**3.1.4 Navigation Feature**

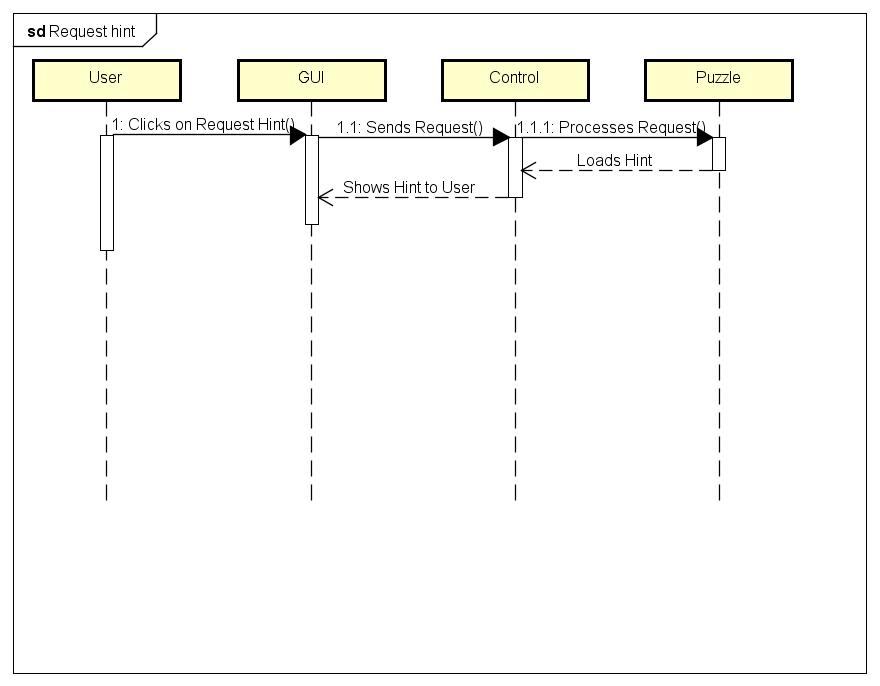
1.4.1 Go to Room Diagram

**3.1.5 Puzzle Feature**

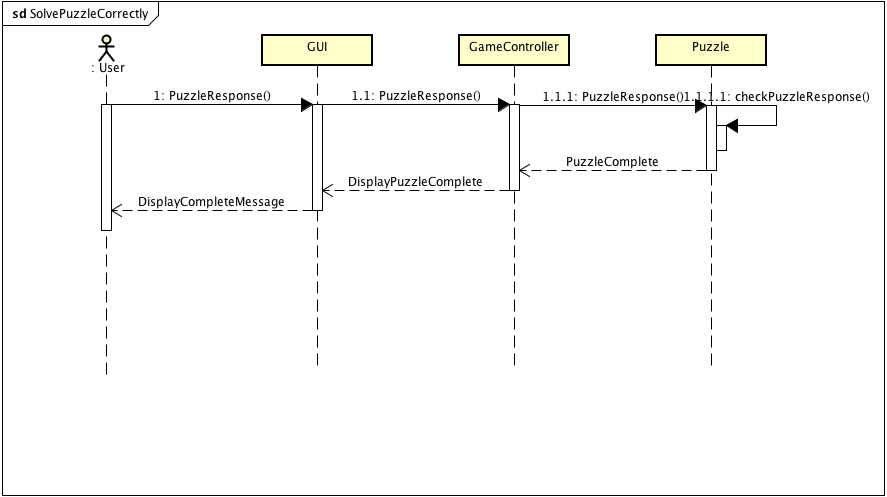
1.5.1 Examine Puzzle Diagram 

1.5.2 Solve Puzzle Incorrectly Diagram

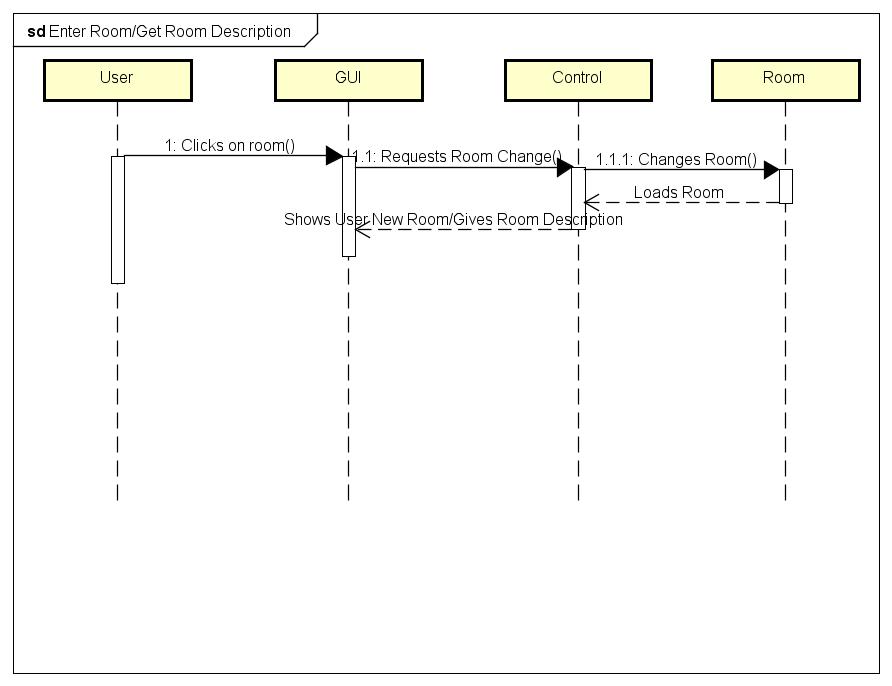
1.5.3 Ignore Puzzle Diagram 

3.5.4 Request Hint Diagram

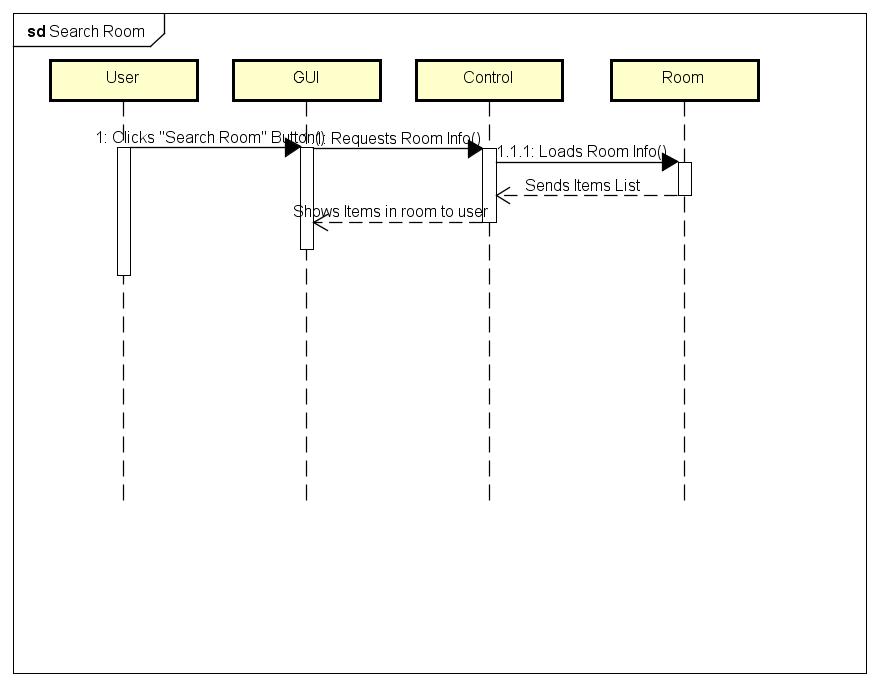
1.5.5 Solve Puzzle Correctly Diagram



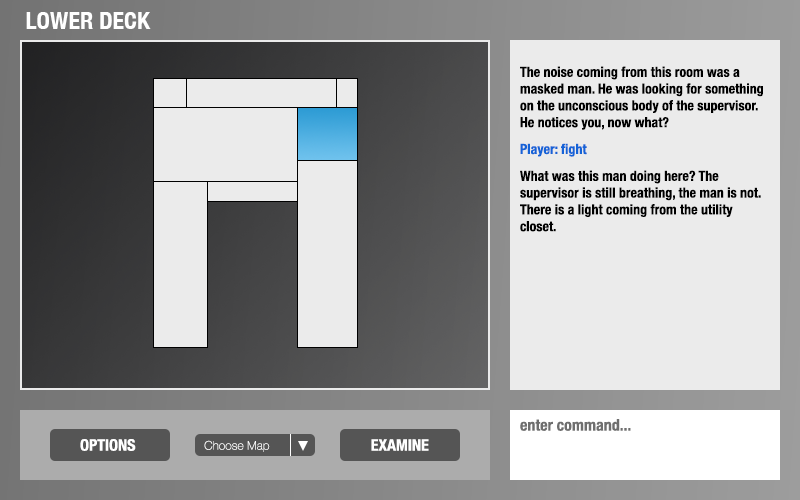
**3.1.6 Room Feature**

1.6.1 Enter/Get Room Description Diagram

1.6.2 Search Room Diagram



**2.4.4 GUI Mockup**

****