Design Document

Version 1 - October 3, 2024

<u>Castle Adventure Game</u>

CPSC-2720

Team members: Areeb, Bobby, James

Repo Manager: Areeb

Researchers/Maintainers: Areeb, Bobby, James

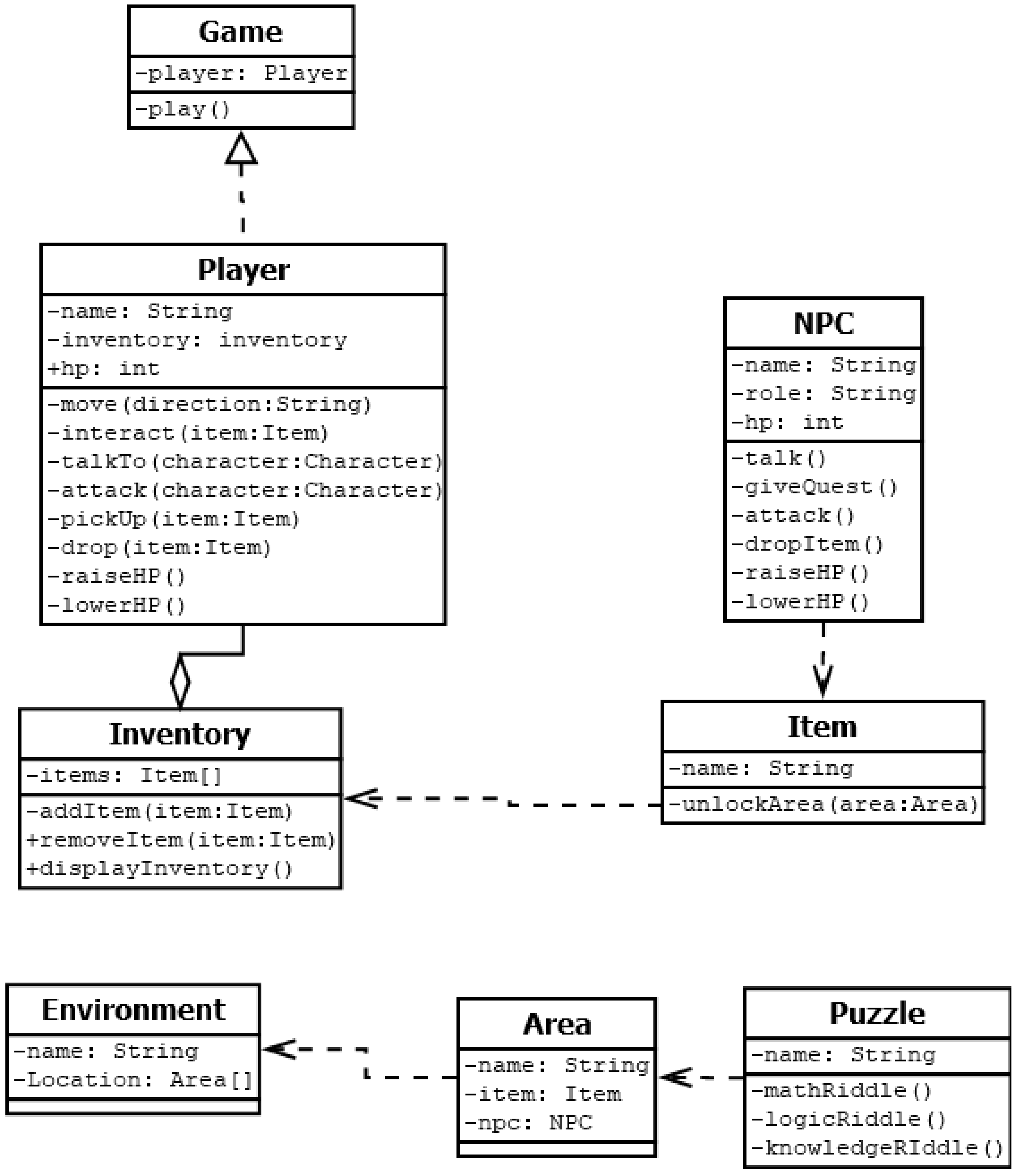
Reporter: Ibrahim, Deen

1. Introduction

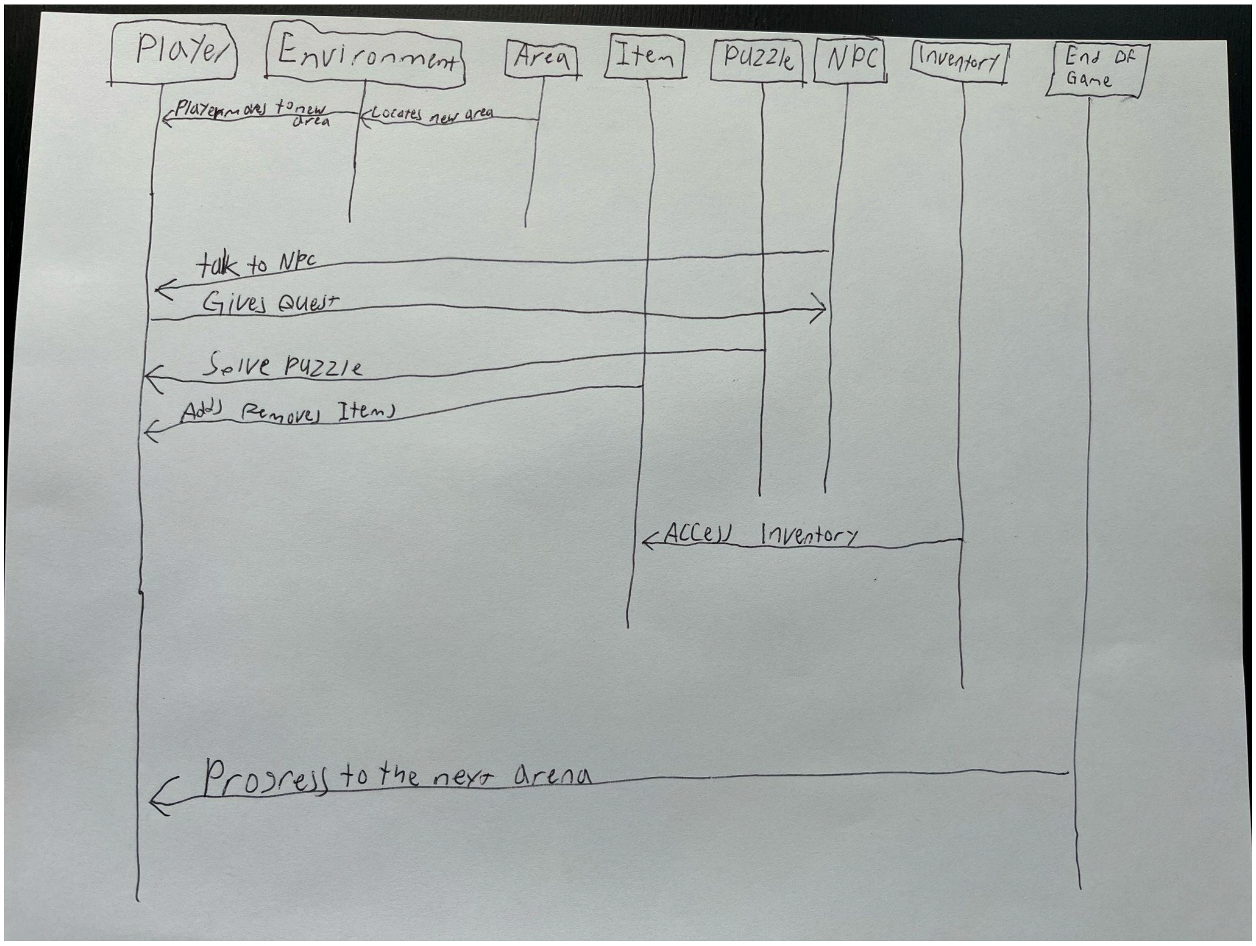
- **Timeline:** Due October 20, 2024
- **Project overview and object of this document:**

This project is about a text-based game in C++ which is going to be castle themed and set in a castle area where there will be different areas/rooms in the castle where the player will have to fight NPCs and when fighting the NPCs. They will give the player a puzzle that they have to solve and pick up items after beating the NPCs and also pick up items in the rooms. The objective of this document is to show the high level design of the program with different types of diagrams.

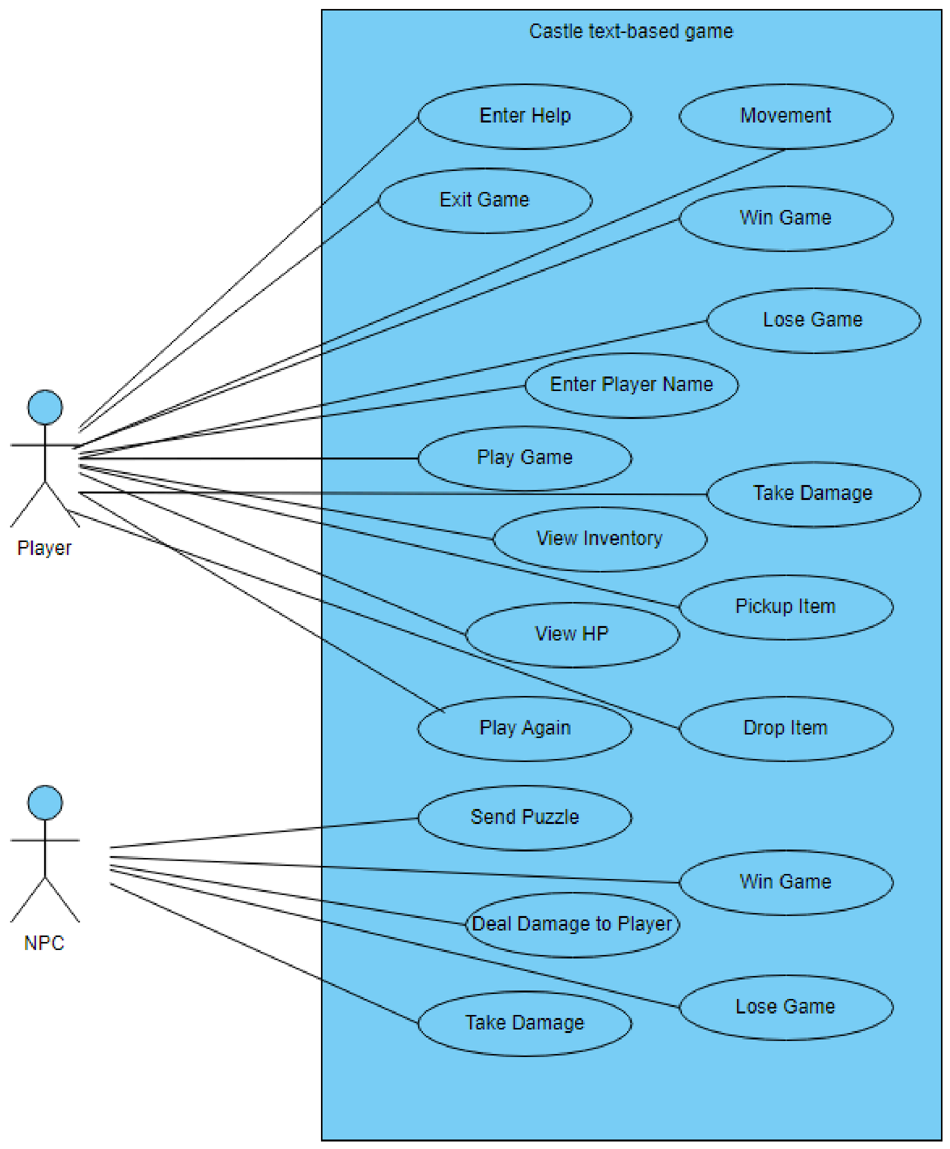
- **UML Class Diagram:**



- **Sequence Diagram:**

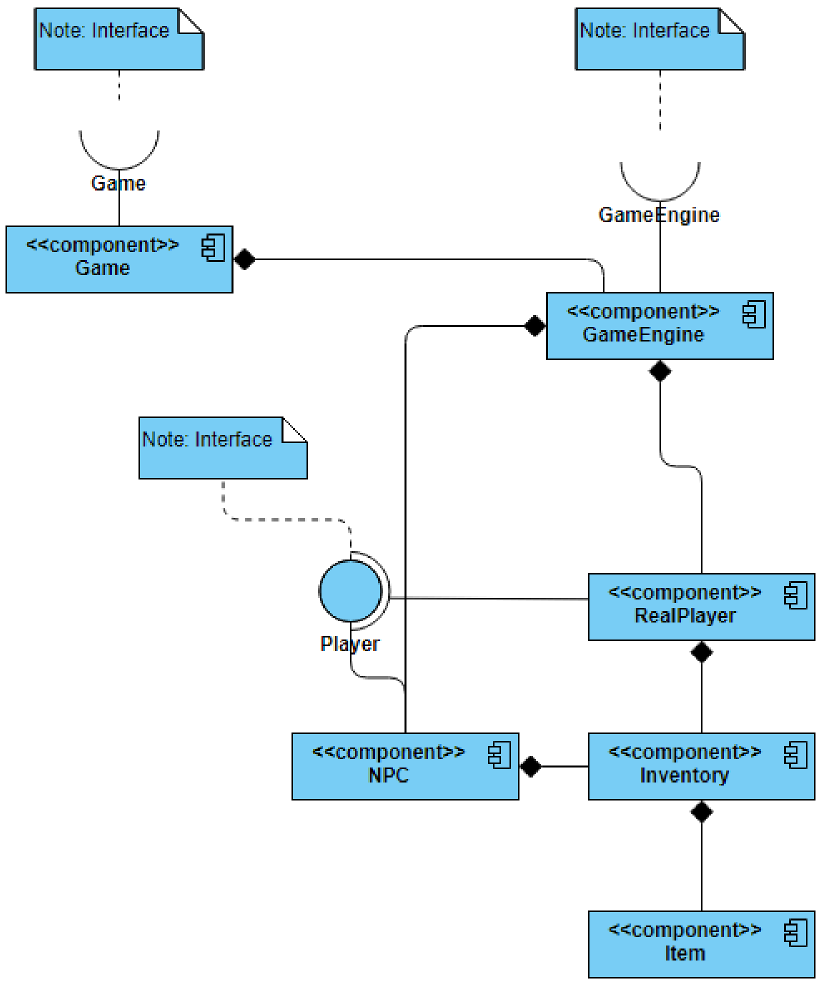


- ## Use Case Diagram:



2. Component Design

- Component Diagram:

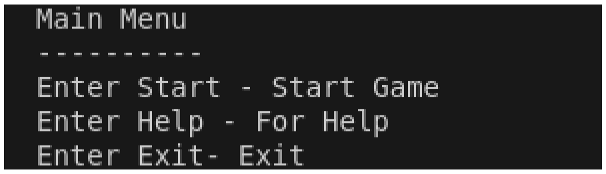


All the .h are the interfaces, and the .cpp are the files that implement all the attributes and functions defined in the .h files. The lines with the filled in square defines composition and the square that is not filled in defines aggregation. Inventory.cpp cannot exist without NPC.cpp and/or RealPlayer.cpp. Item.cpp cannot exist without Inventory.cpp. GameEngine.h cannot exist without NPC.cpp and/or RealPlayer.cpp. None of the .h files and .cpp files cannot exit with Game.h.

- **Component Responsibility Outline:** All the .h files will declare the classes that will be associated with them, functions and/or virtual functions, constructors, destructors and/or virtual destructors. “Game.h” will have the attributes of starting the game like initiating other classes. “GameEngine.h” will have attributes and functions of the environment, rooms, puzzles, etc. “Player.h” can either be an NPC (computer player) or a RealPlayer and will have attributes and functions of the player’s name, health, inventory, items, and what puzzles they have solved and/or lost. “Inventory.h” will have attributes and functions for adding/removing items from a player’s inventory, and displaying player’s inventory when they type for it in the command line. “Item.h” will have attributes and functions that contain the item’s name, description, and how it interacts with players when they pick it up and/or drop it. All the .cpp files will implement all the classes of the .h files and implement the attributes and functions. “Game.cpp” will have the functionality of how the game starts and is responsible for starting the game. “GameEngine.cpp” will have the functionality of the environment, rooms, puzzles, etc. “RealPlayer.cpp” contains all the actions to the real player who is playing the game, movement, connection to rooms, handling inventories when the player picks an item up and/or drops an item, and managing how it interacts with an NPC (computer player). “NPC.cpp” implements how the NPC interacts with players, puzzles that the NPC gives to players, attacking, shooting, or giving items to a player once it loses to the real player. “Inventory.cpp” implements how items are being added, removed, changed, and displayed. “Item.cpp” implements the logic of how items are created, and what each item does to the player (for example, if it regenerates the player’s health, gives access to another room/area, etc.).

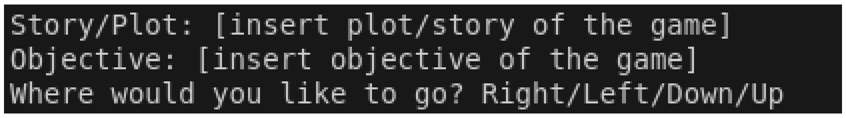
- **Interface Prototype:**

1: Player runs the Makefile (running the game)



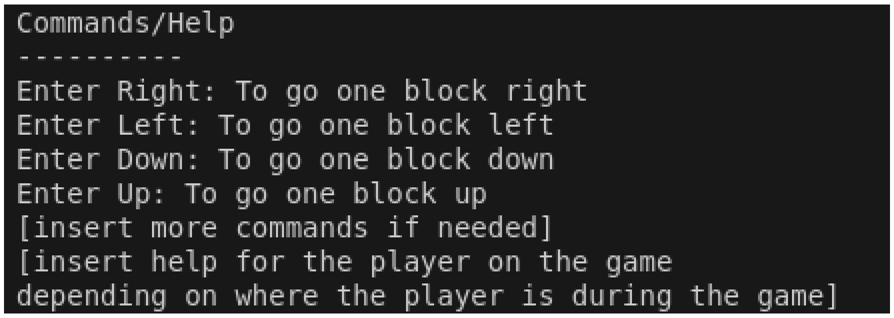
```
 Main Menu
 ----------
 Enter Start - Start Game
 Enter Help - For Help
 Enter Exit- Exit
```

2: Player entered start



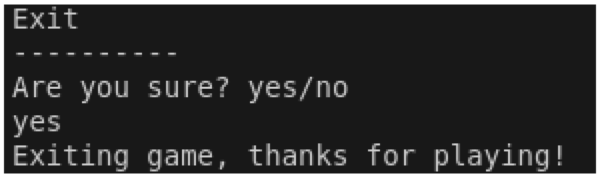
```
Story/Plot: [insert plot/story of the game]
Objective: [insert objective of the game]
Where would you like to go? Right/Left/Down/Up
```

3: Player entered help



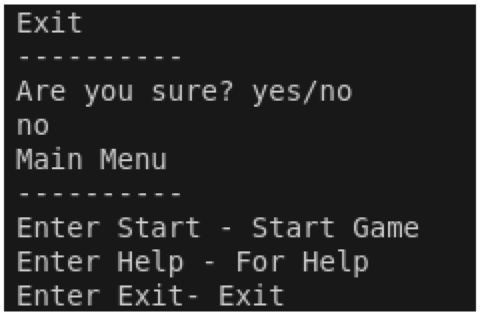
```
Commands/Help
----------
Enter Right: To go one block right
Enter Left: To go one block left
Enter Down: To go one block down
Enter Up: To go one block up
[insert more commands if needed]
[insert help for the player on the game
depending on where the player is during the game]
```

4: Player entered exit and entered yes



```
Exit
----------
Are you sure? yes/no
yes
Exiting game, thanks for playing!
```

5: Player entered exit and entered no



```
Exit
----------
Are you sure? yes/no
no
Main Menu
----------
Enter Start - Start Game
Enter Help - For Help
Enter Exit- Exit
```

3. Class/Object Design

Player: Represents the main playable character, who will interact with the system

Attributes:

Name – String – (Player name)

Inventory – Inventory (Shows the items that the player has)

Hp – Health Points (Shows Player’s health)

Methods:

move (direction: String) – Allows the players to move in any direction

interact (item: Item) – interaction with an item

talkTo (Character – Character) – initiate chat with other characters

attack (Character – Character) – allows the player to attack enemy

pickup (item: Item): Enables the player to collect items

drop (item: Item): Drops item from inventory

raiseHP(): int – Increases the player's health points.

lowerHP(): int – Decreases the player's health points.

NPC (non player Character) - Represents the character the player can interact with

Attributes:

Name: String – The npc’s name

Role: string – The npc’s roles in the game

Hp – Health Points (Shows NPC’s health)

Methods:

Talk() – Engages with the player

giveQuest() - Assign a quest to the player

attack() – Attacks the player

dropItem() – Drops an items may be useful for player

raiseHP(): int – Increases the player's health points.

lowerHP(): int – Decreases the player's health points.

Inventory – All items that player can have

Attributes:

Items: Item[] – A list of items the player has collected

Methods:

addItem(item: Item) – Adds an item to the player's inventory.

removeItem(item: Item) – Removes items from the player's inventory.

displayInventory() – Displays the list of items in the player’s inventory.

Item: Represents objects in the game world that the player can interact with or use.

Attributes:

name: String – The item’s name.

Methods:

unlockArea(area: Area) – Unlocks access to a previously closed area.

Environment: Represents the whole game world, which has different areas that the player can explore.

Attributes:

name: String – The name of the environment.

Location: Area[] – A list of areas within the environment.

Area: Represents a specific location within the environment where player can go

Attributes:

name: String –area's name.

item: Item –item available in the area.

npc: NPC –NPC present in the area.

Puzzle: A type of problem the player must solve in the game. Each puzzle has different types of problem statements.

Attributes:

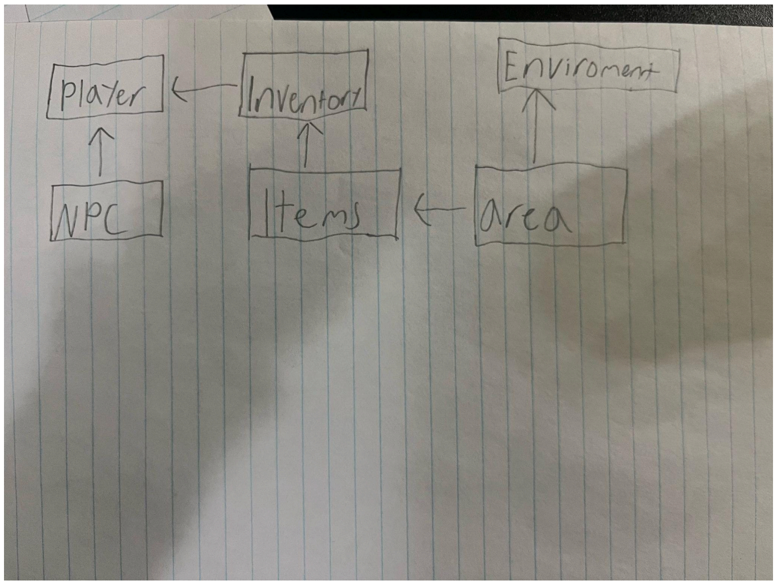
name: String – The name of the puzzle.

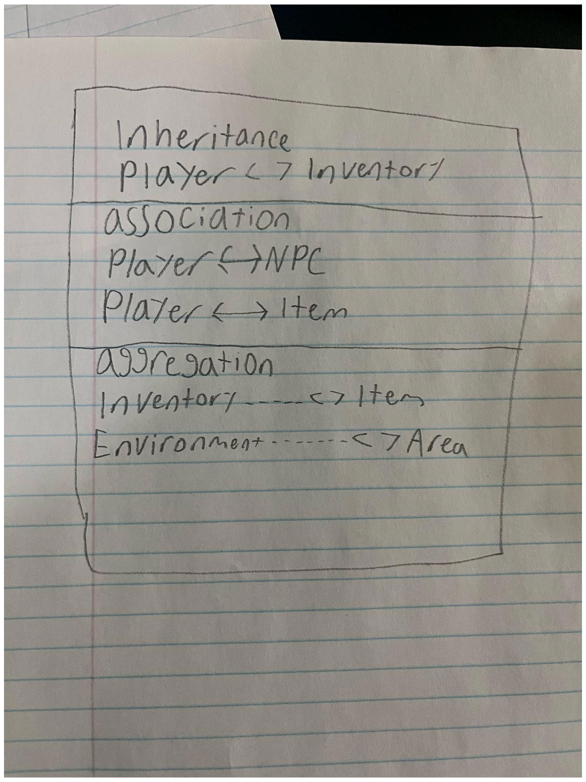
Methods:

mathRiddle() – puzzle of math problems

logicRiddle() – logic-based problem.

knowledgeRiddle() – puzzle testing player's knowledge.





Inheritance

Player <> Inventory

association

Player <—> NPC

Player <—> Item

aggregation

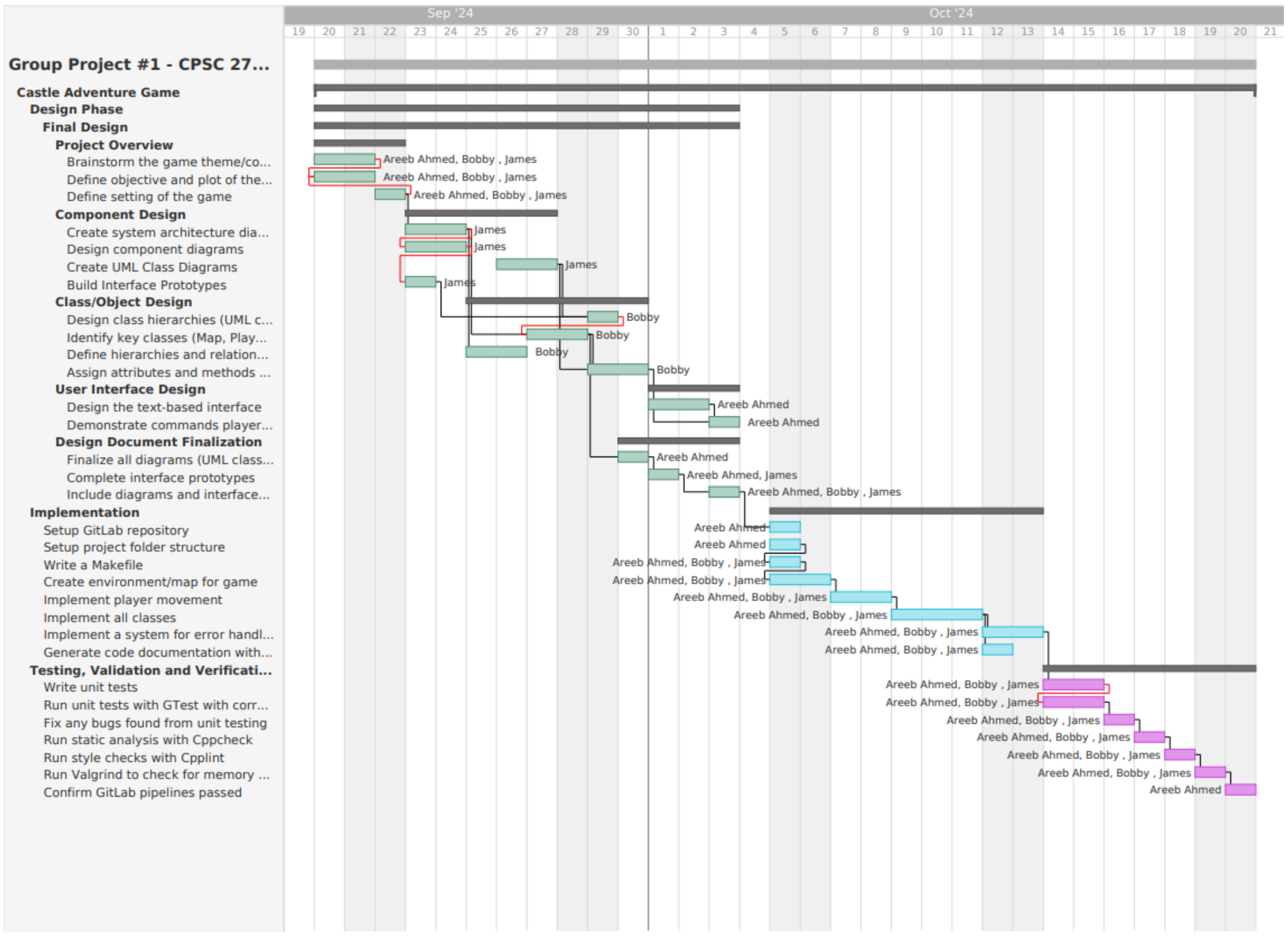
Inventory ----<> Item

Environment ------<> Area

4. User Interface Design

Since this is a CLI adventure game, all the interactions that the player makes will be made by the commands entered in the command line. Based on the commands entered by the player will determine what the game will do, actions that are required, and how it will react to it. For example, if the player mistypes something (capital and lowercase does not matter, so if the player types “Help” instead of “help” or vice versa, the game will recognize it) or types something that is not recognized by the game, then it'll notify the player that the command isn’t recognized and gives the player another chance to type it again. Also, when the player is moving in the rooms, entering/exiting rooms, interacting with items, interacting with NPCs, solving puzzles, it will all be shown in the command line so the player can know what is happening and if they are doing the right things. Therefore, all communication like instructions, plot/story description of the game, objective of the game, if the player has lost/won, and a help command if the player needs help with a command or what commands they can enter will be through the command line.

5. Plan Of Implementation


Sep '24
Oct '24
Group Project #1 - CPSC 27...
Castle Adventure Game
Design Phase
Final Design
Project Overview
Brainstorm the game theme/co...
Define objective and plot of the...
Define setting of the game
Component Design
Create system architecture dia...
Design component diagrams
Create UML Class Diagrams
Build Interface Prototypes
Class/Object Design
Design class hierarchies (UML c...
Identify key classes (Map, Play...
Define hierarchies and relation...
Assign attributes and methods ...
User Interface Design
Design the text-based interface
Demonstrate commands player...
Design Document Finalization
Finalize all diagrams (UML class...
Complete interface prototypes
Include diagrams and interface...
Implementation
Setup GitLab repository
Setup project folder structure
Write a Makefile
Create environment/map for game
Implement player movement
Implement all classes
Implement a system for error handl...
Generate code documentation with...
Testing, Validation and Verificati...
Write unit tests
Run unit tests with GTest with corr...
Fix any bugs found from unit testing
Run static analysis with Cppcheck
Run style checks with Cpplint
Run Valgrind to check for memory ...
Confirm GitLab pipelines passed
Areeb Ahmed, Bobby , James
Areeb Ahmed, Bobby , James
Areeb Ahmed, Bobby , James
James
James
James
James
Bobby
Bobby
Bobby
Bobby
Areeb Ahmed
Areeb Ahmed
Areeb Ahmed
Areeb Ahmed, James
Areeb Ahmed, Bobby , James
Areeb Ahmed
Areeb Ahmed
Areeb Ahmed, Bobby , James
Areeb Ahmed, Bobby , James
Areeb Ahmed, Bobby , James
Areeb Ahmed, Bobby , James
Areeb Ahmed, Bobby , James
Areeb Ahmed, Bobby , James
Areeb Ahmed, Bobby , James
Areeb Ahmed, Bobby , James
Areeb Ahmed, Bobby , James
Areeb Ahmed, Bobby , James
Areeb Ahmed, Bobby , James
Areeb Ahmed, Bobby , James
Areeb Ahmed