CPSC 2720 Practical Software Development

Retrograde

Team Ashley

Tyrell Martens, Landon Constantin, Tyler Hippard, Truman Long

November 5, 2021

TABLE OF CONTENTS

Introduction……2
Project Management……3
Team Organization……...3
Risk Management……4
Software Design……6
Sequence Diagrams……10
Class Description……..12

INTRODUCTION

You find yourself on a spaceship, it's dark, cold, and lonely. You're afraid to see what lurks within, but must press on. As you progress, your once-plentiful oxygen supply becomes scarce. Where is this ship's crew? How do you escape? How will you survive? This text-based adventure game challenges your problem-solving and cipher-cracking skills as you navigate your way through an unfamiliar area via suffocating and dilapidated corridors, with the only indication of progress being strange hints left by the crew as to their fate. Your first few minutes into Retrograde will have you immersed in an unrecognizable room, just getting your bearings after waking up. Who are you? Why are you here? What happened? As you ponder these questions, you hear a faint voice through a communication device on the floor…

Through completing engaging puzzles, the player will progress from room-to-room, gathering all information that they can to overcome different obstacles and piece together the story of what happened here on this spaceship. The player must collect any potentially useful resources and manage their limited oxygen supply, with unknown dangers and hazards that could be lurking behind every door.

With the assistance of a damaged Artificial Intelligence named "MAC-GFN" through the communication device, the player can receive critical information on possible paths to take, helpful instructions if the player feels stuck, objective reminders, and overall moral support.

MAC-GFN's scanners show there may still be signs of life aboard. Do you risk your life to save them, when they might not even be alive? Even if you do find them, will they help? Or will they be a burden on the ship's struggling life support? Creative re-allocation of limited resources may be necessary to survive.

The player's main goal is to survive the trials that the spaceship holds and reach the bridge to repair MAC-GFN. Whether you solve the mystery of the spaceship and discover the underlying reason for the ship's disastrous state or not is up to you.

PROJECT MANAGEMENT

TEAM ROLES

Team Member	**Design - Draft**	**Design - Final**	**Implementation - Basic**	**Implementation - Final**
Truman	Design Lead	QA Lead	Reporting Lead	Phase Lead
Landon	QA Lead	Design Lead	Phase Lead	Reporting Lead
Tyler	Reporting Lead	Phase Lead	Design Lead	QA Lead
Tyrell	Phase Lead	Reporting Lead	QA Lead	Design Lead

Team Role Responsibilities

• Phase Lead:

- Understand what needs to be accomplished in the phase.
- Ask questions/clarification from instructor(s) on behalf of the group.
- Identify (in collaboration with the team) tasks to be done and distribute them.
- Follow up with team members about assigned tasks.
- Identify problems and lead group discussions about how to solve them.
- Make sure the team meets deadlines.

• Design Lead:

- Propose and collect design ideas from the group.
- Assure (in collaboration with the team) that the software design is following SOLID+DRY principles.
- Create or maintain the UML diagrams so they reflect the current design of the software.

• 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.
- Identifies test cases.

• 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

Design Risks

- Design needs to be adjusted during implementation.
    - Review the design to see where it could have been improved.
    - If it is too late, continue with implementation and note where the design could have been improved.
    - The team should be sure the project will compile and meet the minimum requirements (Create a Minimum Viable Product) before adding anything deemed extra.
    - If it is not too late, change the appropriate implementation to correct the design flaw.

- Communication and Role Risk (Roles of the individuals are unclear and confusion arises)
    - Create a respected group environment where tasks are clear and each individual is in charge of their own contributions.
    - Tasks of individuals are communicated by the phase leader at the time.

- Lack of C++ Language Experience
    - Ask group members questions.
    - If other group members are uncertain on a topic, research C++ online or reach out to Professors/TAs for assistance.

- Impending Deadlines
    - Implement classes in order of importance based on a priority hierarchy from highest to lowest.
        - Game
        - PlayerHandler
        - Room
        - Puzzles
        - NPC
        - Non-Puzzle Objects (Interactable)
        - Story Plot Objects (Non-functional, but provide extra story details)
    - The team will discuss possibilities of simplifying the code to make sure the system compiles and runs to its most basic abilities, including sacrificing extra modularity for hard-coded interactions and NPC sub-classes.

- Concerns with team members. What to do if a team member is not responding or contributing in ways they should be.
    - Reach out to team member via email, discord, or phone number if possible.
    - If a team member is not volunteering for tasks the group may assign tasks to them
    - If a team member is lacking in skills for the task they are assigned to it is their responsibility to learn on their own from online sources, course instructors, or help from other team members.
    - If the non-respondent team member does not respond or contribute after a reasonable amount of time, the team may contact Dr. Anvik to discuss options.
    - It is possible for a team member to have unexpected family or personal issues that may cause them to miss team deadlines. If this is the case, it is that team members responsibility to inform the team what is going on. From there, the team can decide how to proceed and what actions need to be taken to complete the project.

DEVELOPMENT PROCESS

CODE REVIEW PROCESS

- Each member will use branches when assigned their implementation task.
- Other members of the group will review the code from a completed branch.
    - They will be able to detect errors a lot better than the creator could (fresh eyes).
- If there is a merge conflict, the appropriate member will resolve it with the aid of the group if needed.

COMMUNICATION TOOLS

- Using a discord server to communicate important documents, ideas, deadlines, ask questions, and organize information through various text channels.
- Using tags during the implementation around where an error is to communicate what the person is thinking, or TODO to denote where some implementation is not yet completed.
    - Ex: /*ERROR: I think there is a Seg fault here because memory on vector is not added before accessed */
    - Ex: /* TODO: This function should also calculate x when doing y */

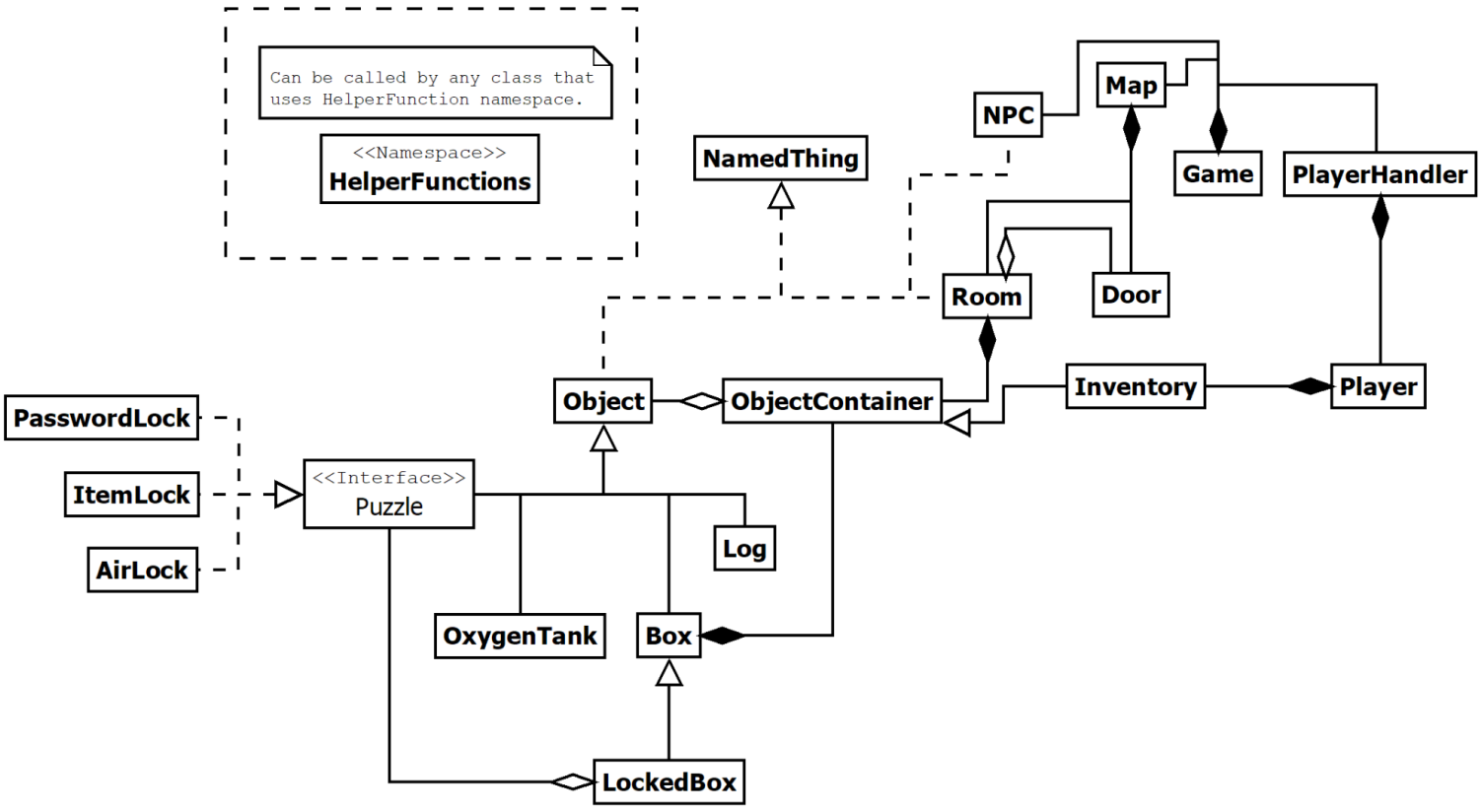
CHANGE MANAGEMENT

- The team will use GitLab's built in issue tracking system to organize issues, changes, suggestions, etc.
- Various tags on the repository will be added for concerns such as possible improvements, performance concerns, story suggestions, bug reports, user experience tweaks, and more.
- If changes are to be made to a significant portion of the code, a meeting will be held to communicate the pros and cons of the change and ask for feedback regarding the change to ensure that each group member is aware of the change before proceeding.

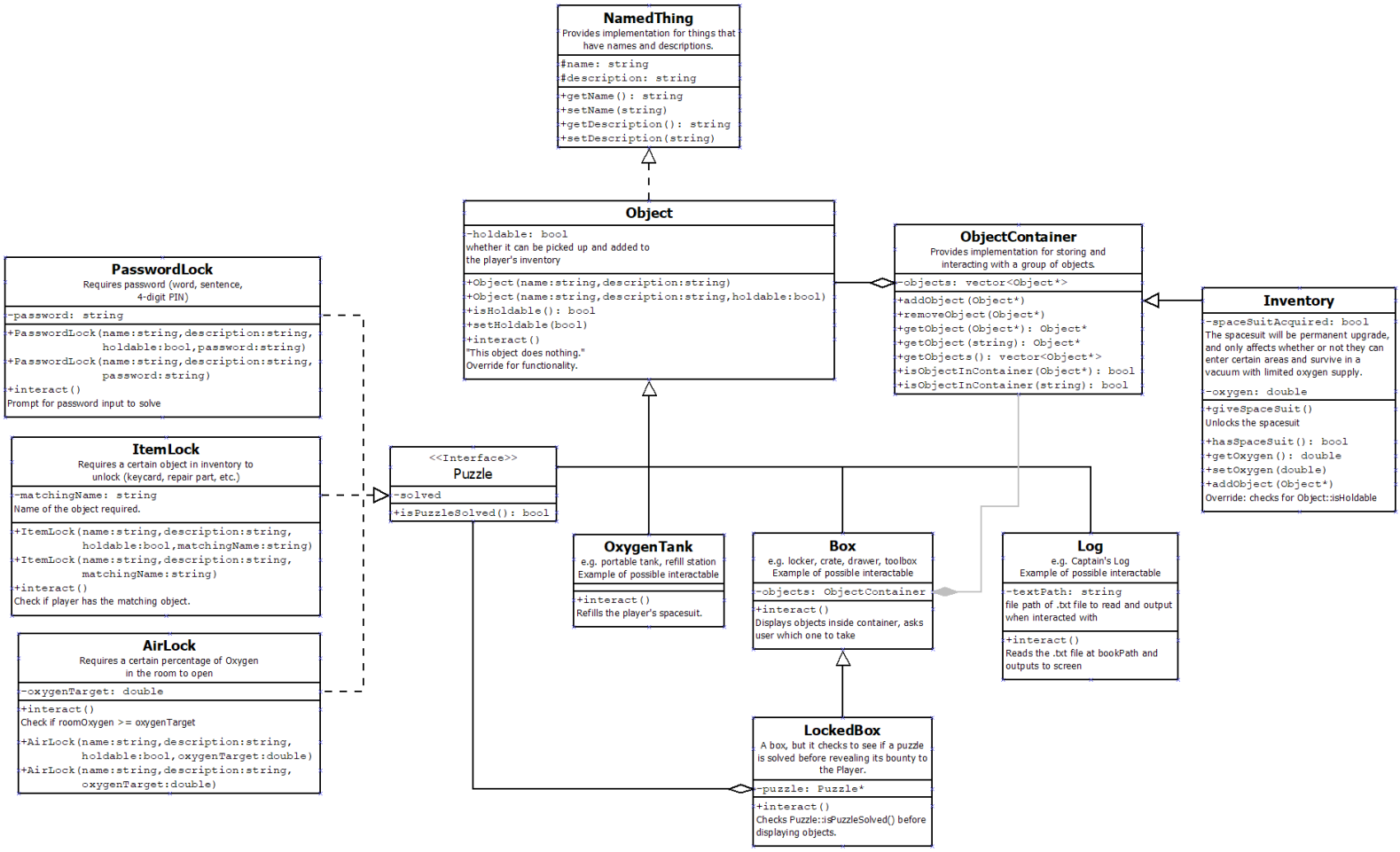
SOFTWARE DESIGN

DESIGN – CLASS DIAGRAMS

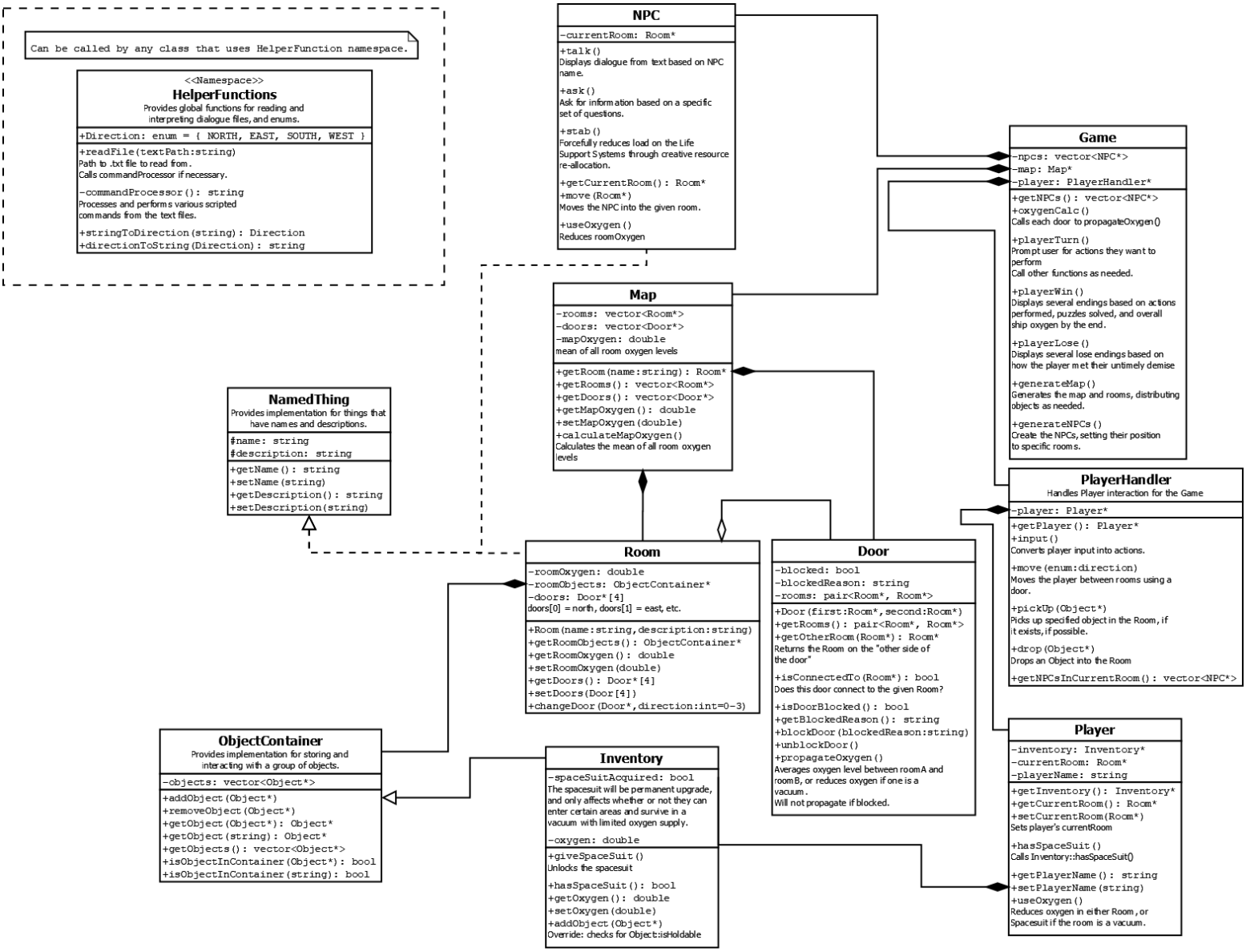
Small Overview Diagram



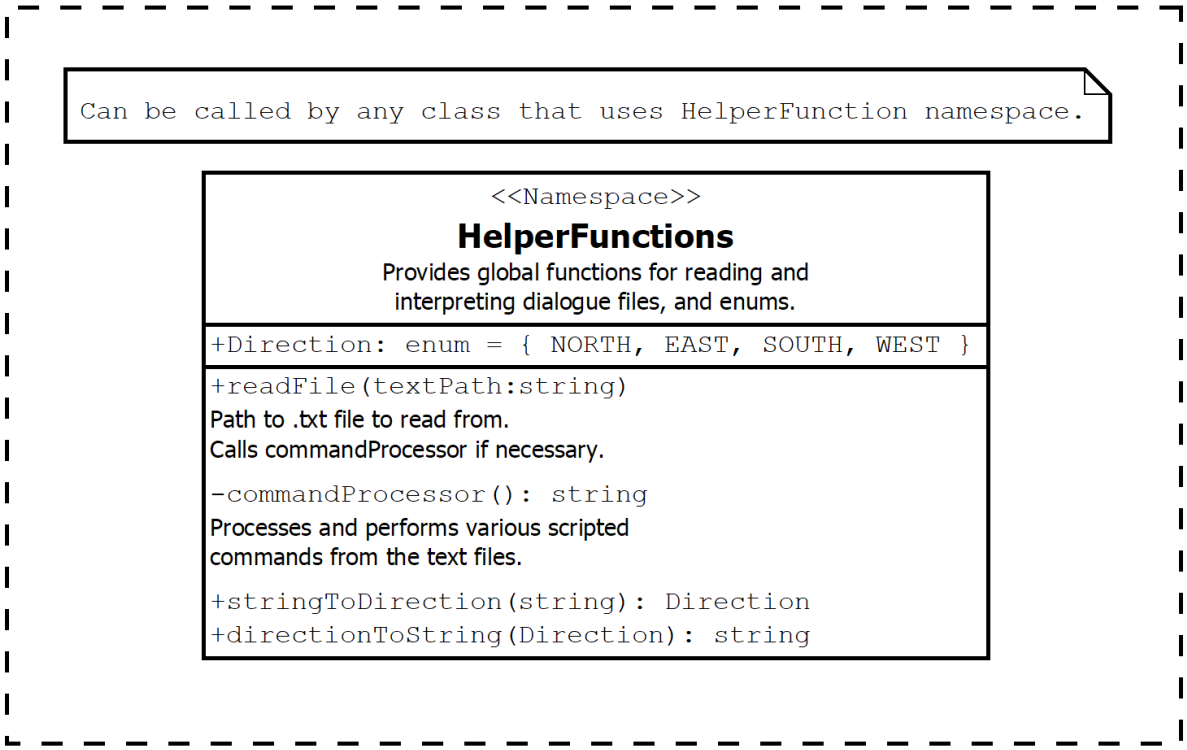
Object Classes



Player, NPC, and Environment Classes

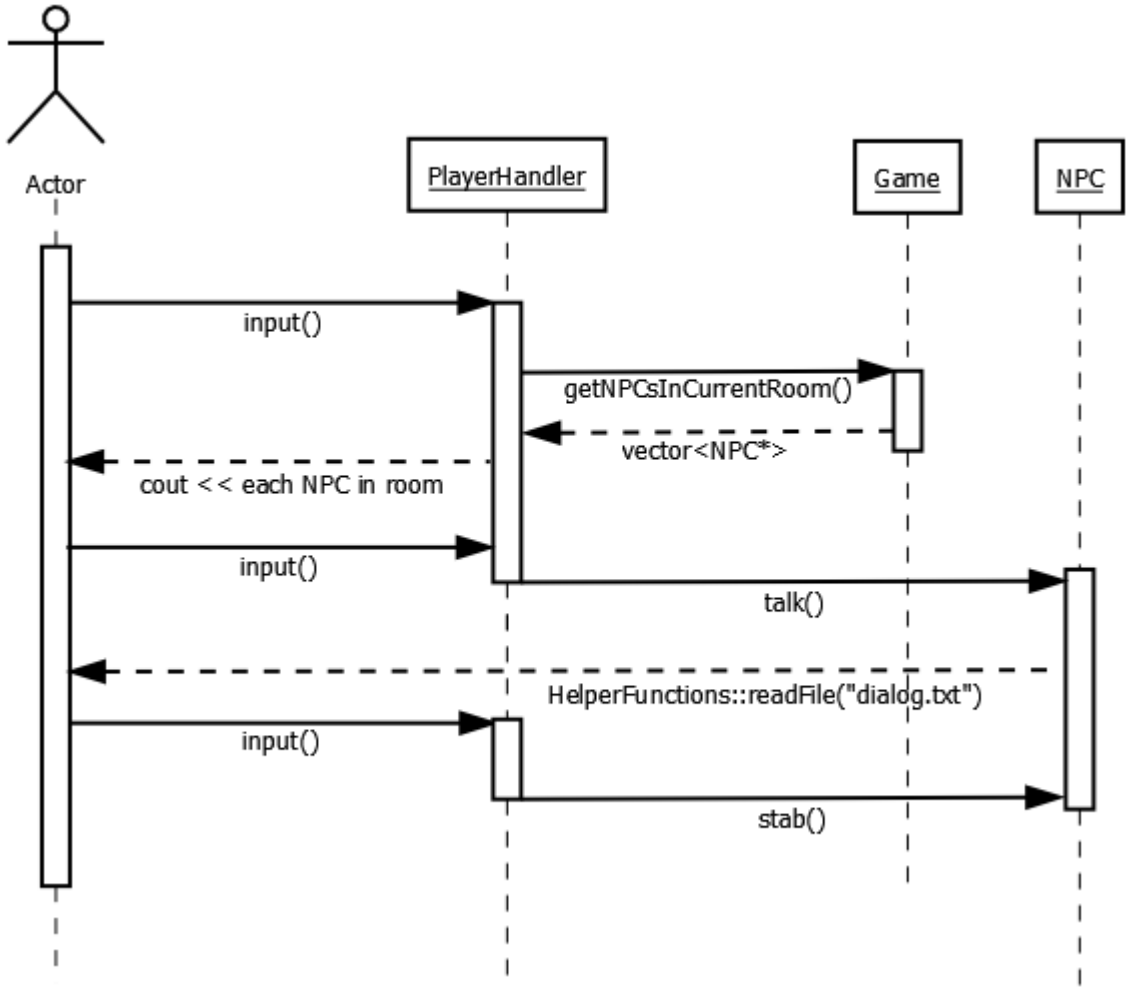


Namespace Class

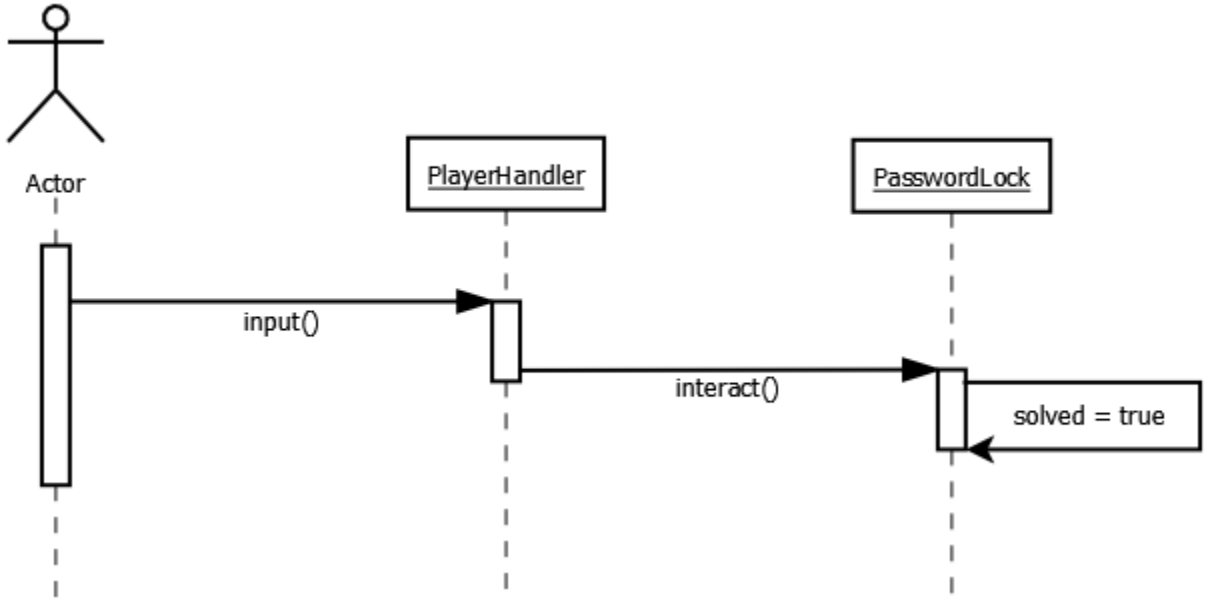


SEQUENCE DIAGRAMS

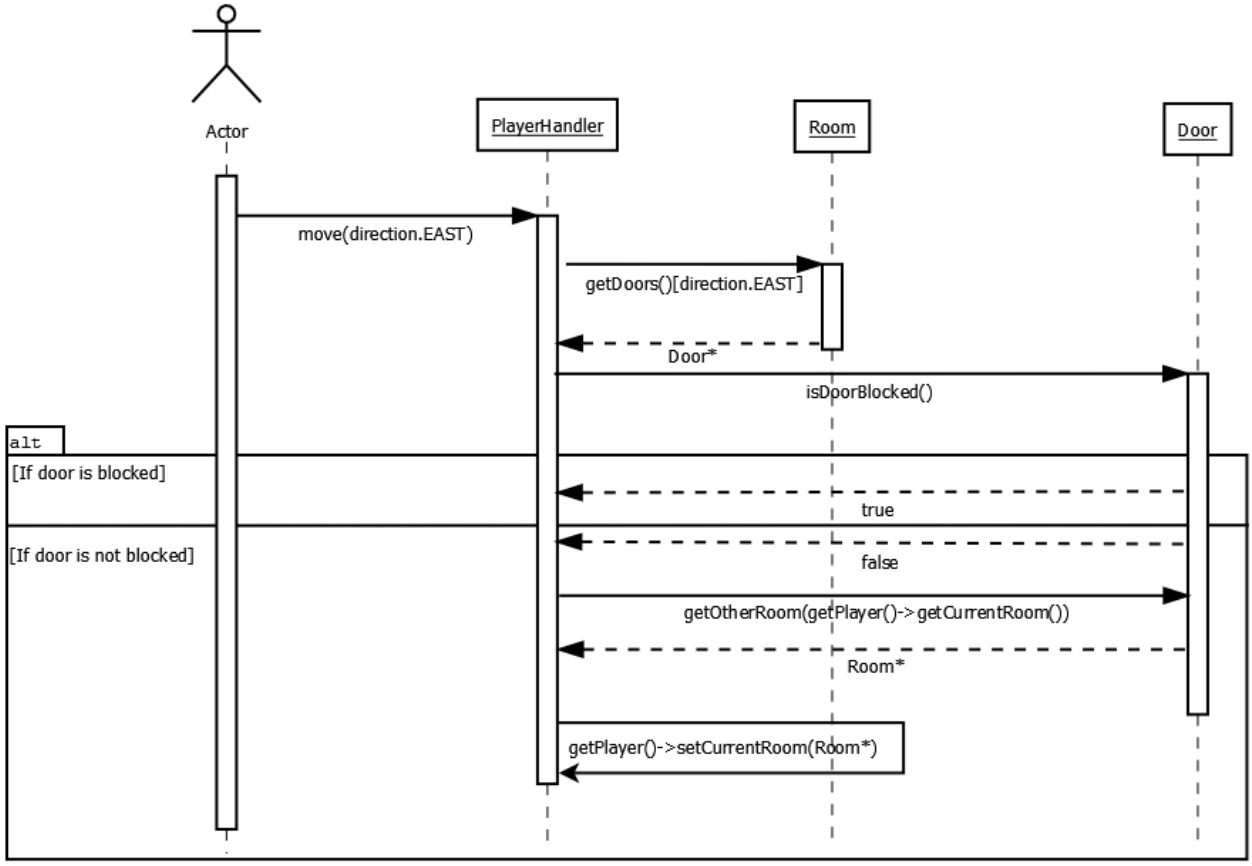
Interaction with NPC



Interaction with Object



Interaction with Environment



CLASS DESCRIPTIONS

NamedThing is a non-pure virtual class providing a name and description for each Object, NPC, Room and Map classes in the game, as well as implemented getters/setters for those attributes. Each of these classes should not need to override this class' behaviour.

Player class represents the user's player in the game. They have an inventory to hold objects, and will keep track of the room they are currently in. This class also handles the player's name, and expenditure of either their spacesuit or the room's oxygen for each action.

PlayerHandler class will handle the players actions. It serves as the human interface for the game, handling user-input to perform in-game actions, such as moving, picking up or dropping objects, and interacting with NPCs and objects.

NPC class represents the non-playable characters in the game. This class handles their current room, movement, and expenditure of room oxygen. The player will be able to talk with them, ask questions, and stab them. If an NPC is stabbed, the oxygen of the room will deplete less quickly.

Object class describes all objects in the game. An object is something that exists in the environment. Some objects may be picked up by the player and added to their inventory. All objects can be interacted with, with some providing special behaviour, and some doing absolutely nothing besides creating user happiness..

OxygenTank class represents a type of object that replenishes a player's suit oxygen when used.

Box class represents an object that has objects inside them. It uses ObjectContainer for the majority of its behaviour, although users will not be able to insert an item into a Box.

LockedBox describes an object that is essentially the same as a regular Box, except that it requires some linked puzzle to be solved before the user can access its contents.

Log class embodies a type of object that is centered around the plot of the game. This will read a text file and output it to the screen. Provides overall background information on the story.

Puzzle class represents the interface for all the puzzles in the game. When a puzzle is completed, it will make sure the player does not need to re-solve the puzzle in the future with a solved state flag.

PasswordLock represents a password/passphrase puzzle in the game. It will ask the user to input a string and compare it to the expected string, which can be nearly any text or number combination we wish to create.

ItemLock represents a puzzle where the user must have a specific object in their inventory. If they have the item, the puzzle is solved. This class compares the object's name to the name of the specific object the puzzle needs.

AirLock represents a puzzle where the user must have a certain amount of oxygen in the room in order to complete the puzzle. The oxygen in the room must be greater than or equal to the oxygen target in order for the puzzle to be solved.

Room class represents a room in the game. It has 4 doors, its own oxygen value and an ObjectContainer. This will be the main form of how the player interacts with the environment, such as picking up/dropping items from/into a room, or moving between rooms via doors.

Door class represents a door that is between two rooms. It can be blocked and has a reason as to why the door is blocked if needed.. It will have the two rooms which it is connected to, which players can use to switch room.s Doors can be unblocked or blocked depending on the situation. If a door is not blocked between rooms, the oxygen in the rooms will propagate into the other room.

Map class represents the map in the game. It stores all rooms and doors in the game, as well as an average oxygen level for the whole map.It will have functions to calculate the average oxygen level in the map.

Game class represents the state of the game. This class will generate NPCs in their initial positions as well as the map that the player will traverse throughout the duration of the game. It will determine if the player has successfully completed the win condition of the game or has failed. This class also handles the turn flow of the game, triggering oxygen usage/propagation, and handing off control to the PlayerHandler class for user actions.