Reflection of Project (Assignment 4):

The design phase was the most crucial part in this project for me as it made things very simple to implement when completing the functionality requirements of the game. Instead of the game running through just one class, many classes were used to store, get, and update information for the various objects used within the game. Various classes are used for these different objects to adhere to the Single Responsibility Principle that emphasises on classes having one responsibility and a single reason to change, which makes it easier to test classes and organise them. For the monster class in particular, we create a separate dragon class that inherits the monster class. This is done to adhere to the Liskov Substitution principle which puts emphasis on being able to replace an inherited class by its sub class without changing the behaviour of the program. Through the dragon class we inherit all of monster and are able to add another method that is exclusive to the dragon without any issues. Using these classes also definitely adheres to the Don't Repeat Yourself principle, which highlights the use of these classes and methods to reduce the amount of code repeated and ensuring there is a minimal amount of maintenance required. I also made sure that players, locations, and items classes can all be accessed by the location class, as this was important for most location class methods that depended on these other classes.

Within the design itself there was a lot of things I found ambiguous so I made an effort to add a few things so that the game would run a little smoother. One example was the combat system that was implemented from the brief. In this situation it was stated that when a player or monster has a greater attack value + a random value from 2- 10, the damage inflicted would be players damage – monsters defence, or vice versa. In both cases, if the damage – defence was equal to 0, the combat sequence would be stuck in an infinite loop and thus in this scenario I made it so that at least 1 damage is taken to progress the game. There was also a lot of issues when dealing with creating locations by getting them from the text file. To fix these issues I decided to create a global vector called map which would store all 20 of the locations and created an ID variable for the player which would be used as an index within the map to get the location of the player. I also had to change the idea that an Item was set for a monster within its class as whenever I tried to use map[ID].getMonster()->getitem(), the value would always return null. I wasn't sure how to fix this, so instead I created a pickItem and a pickArtefact method in the main class which would take a random number and choose an item that is dropped by the monster varying with its predetermined drop rate percentage.

For the future, I would probably create a few more classes to better organise and maintain my code, as the main class of the project was overwhelmed by the end of the project. Especially for the classes that are repeated so often like getting inputs of different types, these are more helper methods and probably should be differentiated from the main game logic class. I definitely think I could've gone out of my way and spent more time to make the more reliable and reusable and not just be satisfied that the functionalities work. I would also probably spend a bit more time with planning a lot of the classes used in the main application that were obviously going to be implemented such as generating all the locations, setting up all the items and creating the player for use.

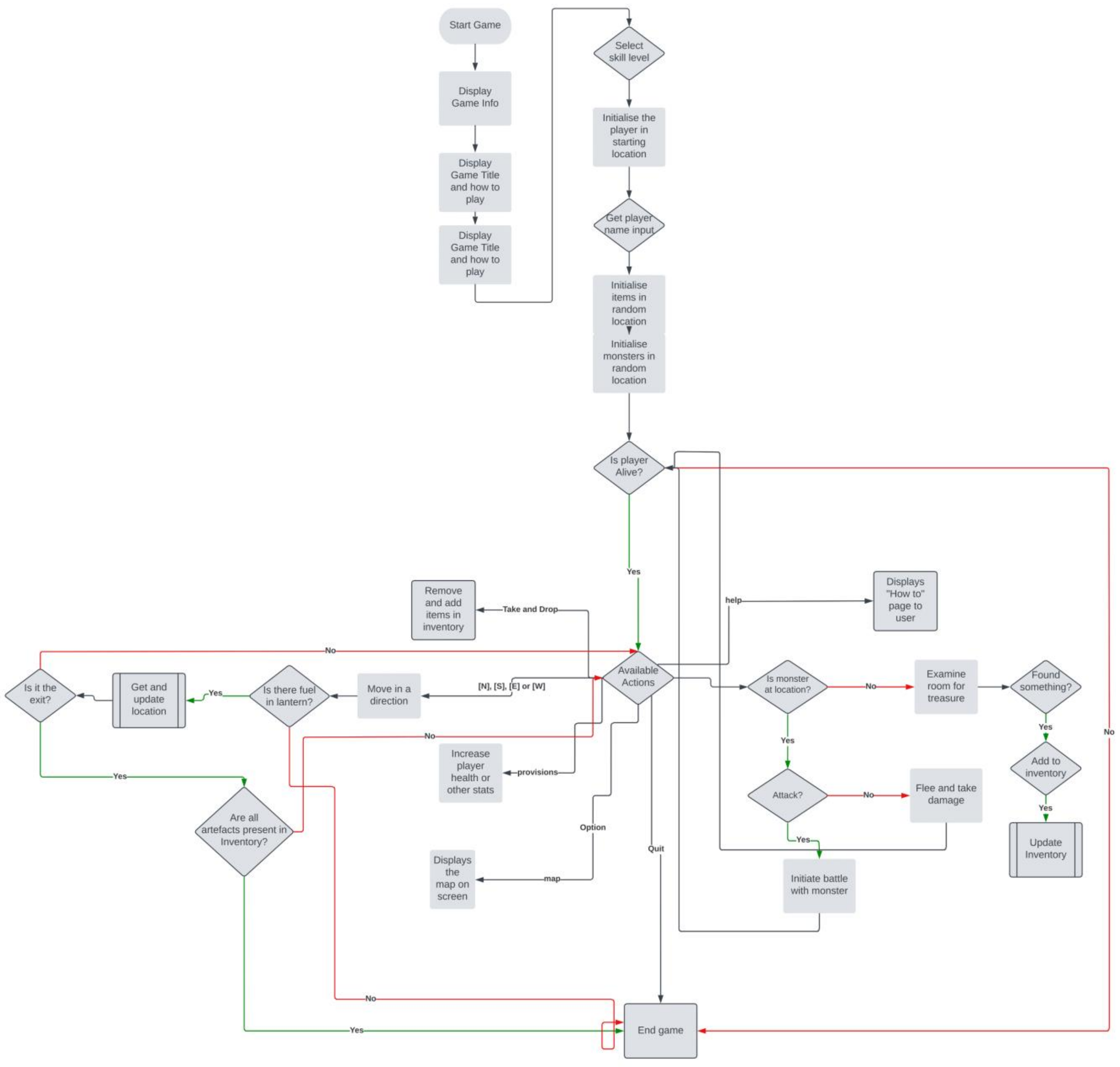
Assignment 2 file is below for reference:

Assignment 2: Preparing the Treasure Caverns of Doom (Part A) – Project Documentation

Name: Raunak Koirala

Student No. 32509987

Flowchart Design:


Start Game
Display Game Info
Display Game Title and how to play
Display Game Title and how to play
Select skill level
Initialise the player in starting location
Get player name input
Initialise items in random location
Initialise monsters in random location
Is player Alive?
Yes
No
Available Actions
help
Displays "How to" page to user
Take and Drop
Remove and add items in inventory
[N], [S], [E] or [W]
Move in a direction
Is there fuel in lantern?
Yes
No
Get and update location
Is it the exit?
No
Yes
Are all artefacts present in Inventory?
Yes
No
provisions
Increase player health or other stats
Option
map
Displays the map on screen
Quit
Is monster at location?
No
Yes
Examine room for treasure
Found something?
Yes
Add to inventory
Yes
Update Inventory
Attack?
No
Yes
Flee and take damage
Initiate battle with monster
End game

Project Test Plan:

Class	Function	Description	Expected Result	Actual Result
Player, Monster, Dragon	setHealth	Mutator method to initialise health for actor	Implemented within constructor, should set value given in brief	As expected
Player, Monster, Dragon	setAttack	Initialise attack	Set value in brief	As expected
Player, Monster, Dragon	setDefence	Initialise defence	Set value in brief	As expected
Player, Monster, Dragon	setDamage	Initialise damage	Set value in brief	As expected
Player	setLuck	Initialise luck	Set value in brief	As expected
Monster, Dragon	setItem	Initialise random item	Provide random artefact to monster	As expected
Monster, Dragon	setLevel	Initialise level	Set value in brief	As expected
Player	setName	Initialise player name based on input	Set name with given input	As expected
Monster, Dragon, Item, Location	setName	Initialise name	Set name given in brief	As expected
Player, Monster, Dragon	getHealth	Returns health of monsters or player	Return expected value from getDetails section during all tests	As expected
Player, Monster, Dragon	getAttack	Returns attack of monsters or player	Return expected value from getDetails	As expected
Player, Monster, Dragon	getDefence	Returns defence of monsters or player	Return expected value from getDetails	As expected
Player	getLuck	Returns luck of player	Return expected value from getDetails	As expected

Monster, Dragon	getItem	Returns artefacts of monsters	Return expected value from getDetails	As expected
Monster, Dragon	getLevel	Returns monsterlvel	Return expected value from getDetails	As expected
Player Monster, Dragon, Item, Location	getName	Returns name of object	Return expected value from getDetails	As expected
Player	updateInventory	Used to add or remove items from inventory	Create and add items in a player's inventory	Does add to inventory however another function might be needed to remove from inventory later on or added onto this method
Player	getInventory	Return players inventory items in list	Converted to string in another function and called in getDetails	Returns as vector so another function is required
Player, Monster, Dragon	updateHealth	Update the players health based on artifacts and items	Add or remove values based on given function parameter	As expected
Player	updateAttack	Update the players attack	Add or remove values based on given function parameter	As expected
Player	updateDefence	Update the players defence	Add or remove values based on given function parameter	As expected
Player	updateDamage	Update the players damage	Add or remove values based on given function parameter	As expected
Player	updateLuck	Update the players luck	Add or remove values based on given function parameter	As expected

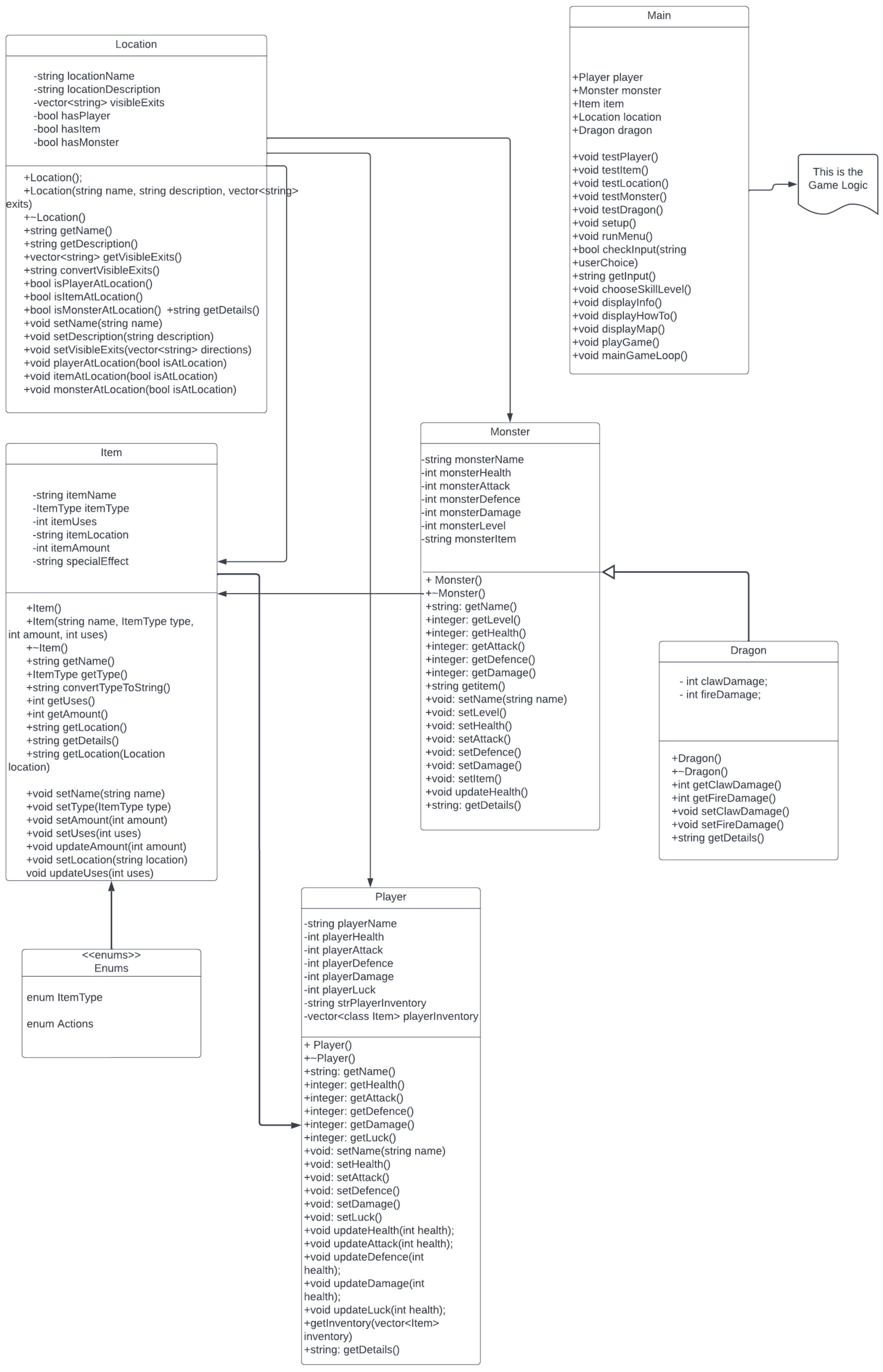
Player, Monster, Dragon, Item, Location	getDetails	Displays all data in an organised manner in string form	Called numerous times to check accessor methods in classes	As expected
Player	convertInventoryToString	Convert vector into string for user display	Called in getDetails whilst testing to show inventory	As expected
Item	getType	Returns item type	Converted and later called in getDetailswhile testing	As expected
Item	convertTypeToString	Converts type to string from enum	Called in getDetails to display item type	As expected
Item	getUses	Returns no. of uses of item	Called with getDetails and updated with another function	As expected
Item	getAmount	Returns amount of item	Called with getDetails	As expected
Item	getLocation	Returns location of item	Called with getDetails	As expected
Item	setType	Sets type of item	Set type provided	As expected
Item	setAmount	Sets amount of item	Set amount provided	As expected
Item	setLocation	Sets random location in which item is stored	Set location provided	As expected
Item	setUses	Sets no. of uses of item	Set uses provided	As expected
Item	updateAmount	When multiple items are found function adds them to item amount	Applied to item to check addition based on given parameter	As expected
Item	updateUses	Reduces uses when item is used	Applied to item with multiple uses to check subtraction	As expected

Location	getDescription	Returns description of location	Called with getDetails	As expected
Location	getVisibleExits	Returns exits as vector	Later converted and tested within getDetails	As expected
Location	convertVisibleExits	Converts to string to display to user	Displays string visible exits to user using getDetails	As expected
Location	playerAtLocation	Checks if player in location and returns true or false	Called with getDetails	As expected
Location	itemAtLocation	Checks if item in location and returns true or false	Called with getDetails	As expected
Location	monsterAtLocation	Checks if monster in location and returns true or false	Called with getDetails	As expected
Location	setDescription	Initialises location description	Set description given during testing	As expected
Location	setVisibleExits	Sets exits visible from location	Set exits given during testing to update list	As expected
Dragon	getClawDamage	Returns claw damage	Tested in getDetails	As expected
Dragon	getFireDamage	Returns fire damage	Tested in getDetails	As expected
Dragon	setClawDamage	Sets claw damage	Tested in getDetails	As expected
Dragon	setFireDamage	Sets fire damage	Tested in getDetails	As expected
Monster	Monstor constructor	Initialises the monster based stats and item it carries based on given name	Initialise values for class tested with getDetails	As expected
Player	Player constructor	Sets taken name from user and	Initialise values for class tested with getDetails	As expected

		randomises stats		
Item	Item constructor	Takes name, type, amount and uses and sets value of location randomly	Initialise values for class tested with getDetails	As expected
Dragon	Dragon constructor	Haven't added any parameters yet however is responsible for special skills and damage	Initialise values for class tested with getDetails	As inhertitance is not implemented the entire class wouldn't properly display in getDetails thus only the attacks are displayed
Location	Location constructor	Initialises name, description and exits	Initialise values for class tested with getDetails	As expected
Main	Main()	Runs the main logic of code	Used to test and run all other function	As expected
Main	runMenu()	Display main menu	Show brief sample menu	As expected
Main	getInput()	Gives user input as string	Tested while testing functions which use this method	As expected
#Scenario	Function	Description	Expected Result	Actual Result
Selecting options	checkInput	Checks all input to make sure a valid command is entered.	Ensures that player is asked to re-enter if letter isn't valid, quit if "Q" is given and do the respective action when appropriate command is entered. Different letters	When "A" is entered, the system does execute the testAction() class for testing the combat within the game so works as expected.

			are entered to test this.	When “Q” is entered the game thanks the player for playing and quits
Ending game conditions	isAlive	Ensures player is alive after combat is completed	When player loses the combat, they are no longer alive and quits application	The algorithm implemented did not produce the output desired as application did not exit with player losing, the function needs to be changed to a Boolean to check result easier
Combat	testAction()	Does a combat sequence as required by the brief	Set values for both monster and character to simulate update in health and the winner of a battle	As expected, the battle continues till the monster or player loses all health

UML Diagram of project:



Description of “How to Play”:

You enter the Treasure Caverns with a map, your sword, a lantern, and a backpack with some provisions.

To achieve your objective, you must explore the vast and dangerous caves to vanquish the Dragon and retrieve Balthazar’s Spell Tome as well as four other artifacts guarded by minions and return them to the village.

Use your map and your lantern to navigate this dark and treacherous cave and make sure to use provisions and potions to keep yourself alive. You will only be able to remain alive while your lantern has oil else the darkness will swallow you. While journeying through the cave, upgrade your armour and make use of already collected artifacts to aid you in battle.

Look out for and pick up lost gems or gold dropped by fallen adventurers to take back to the village and trade with the villagers.

Controls:

Move using commands: [N], [S], [E], [W], for the direction you want to take

You may attack a monster using command: [A]ttack

You may flee from combat using command: [F]lee

You may stop to look for items and treasures using command: e[X]amine

You can pick up things and remove things from your inventory using command: [T]ake and [D]rop

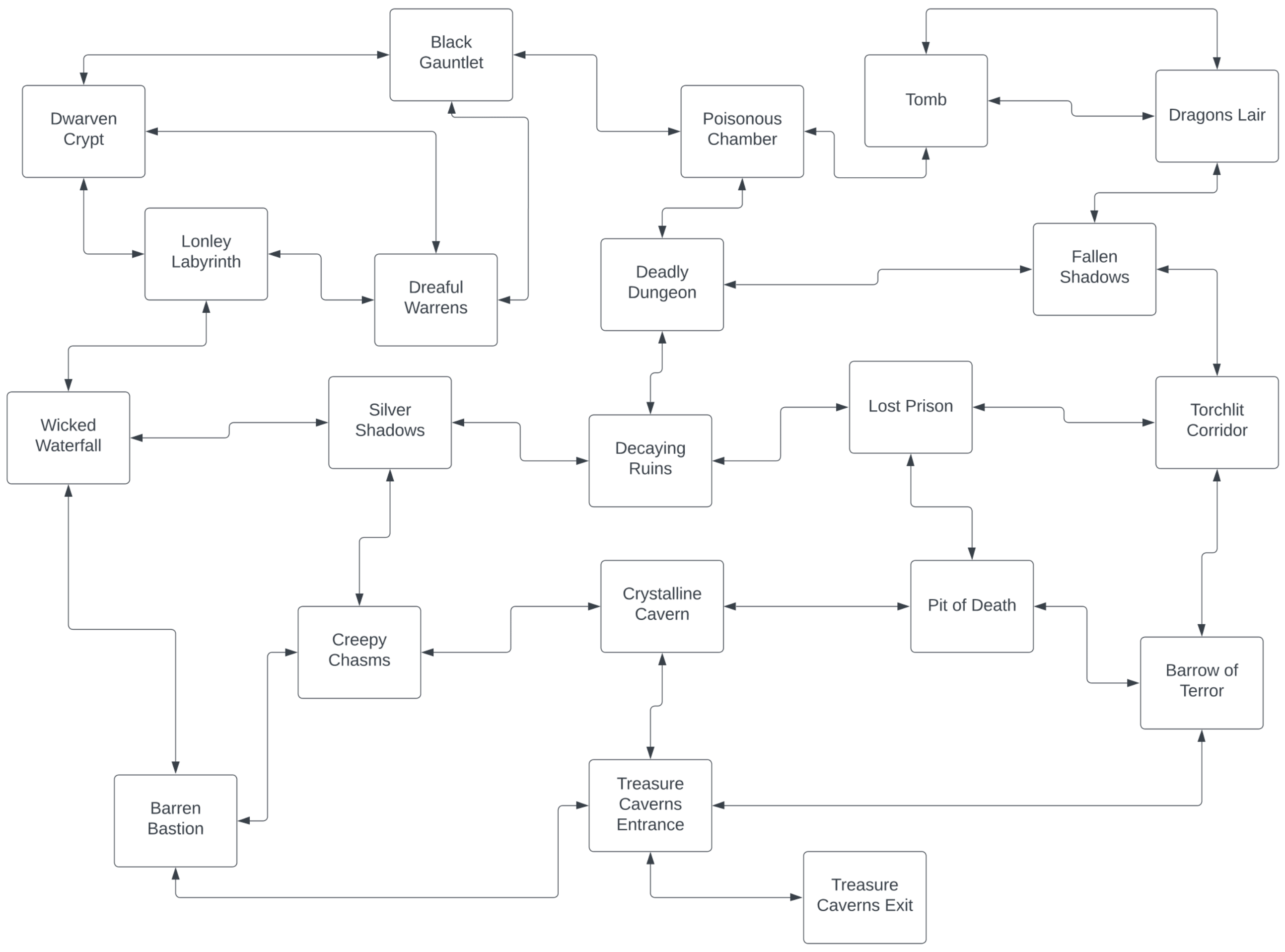
You may use your provisions using command: [P]rovisions

You may bring up the map using command: [M]ap

You may bring up this help menu using command: [H]elp

You may exit the game at any time using command: [Q]uit

Map of the Caverns with 20 caves:


Black Gauntlet
Tomb
Dragons Lair
Dwarven Crypt
Poisonous Chamber
Lonley Labyrinth
Dreaful Warrens
Deadly Dungeon
Fallen Shadows
Wicked Waterfall
Silver Shadows
Decaying Ruins
Lost Prison
Torchlit Corridor
Crystalline Cavern
Pit of Death
Creepy Chasms
Barrow of Terror
Barren Bastion
Treasure Caverns Entrance
Treasure Caverns Exit