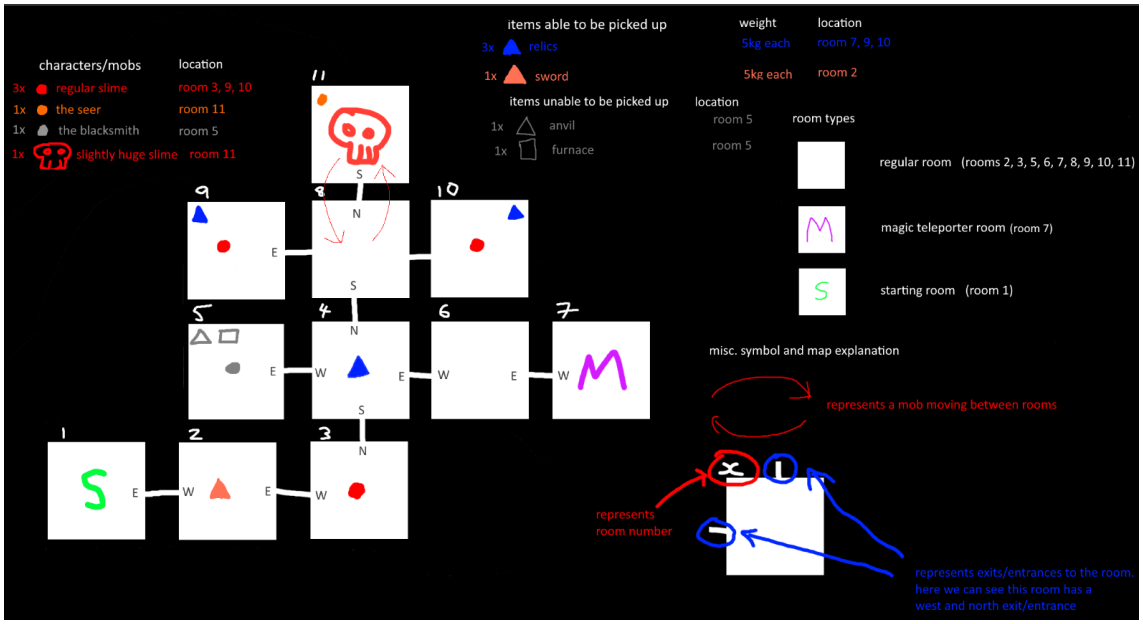
Tensura Labyrinth

Introduction

Tensura Labyrinth is a simple text-based adventure game where you are lost in a labyrinth and you must escape the labyrinth by picking up the relics and giving them to the Seer, and defeating all the slimes. Below, is a map of the game, accompanied with an explanation of the game.



You start in a room, and you can travel between rooms by using the go command to go either north, west, east, or south. In rooms, you may see items that are pickable or not pickable, and you may see NPCs that are enemies or friendly NPCs. You can interact NPCs with interact. Enemies will not attack you, but you must slay all the enemies as one of the objectives to pass the game. To slay enemies, you must pick up a Sword in one of the rooms to be able to attack enemies. There are several commands that can help you such as pickup, drop and attack to interact with items or NPCs.

Items

All items can be interacted with. They will pop up with some flavour text. Some items can be picked up.

Item	Is pickable by the player	Location (Room)	Weight (kg)	Amount	Purpose of item
Relic	Yes	7, 9, 10	5	3	All 3 relics must be given to the seer as one of the winning conditions
Sword	Yes	2	5	1	Used to defeat enemies
Furnace	No	5	30	1	Decorative item
Anvil	No	5	50	1	Decorative item

NPCs

All NPCs can be interacted with. Some NPCs can be attacked.

NPC	Is attackable by the player	Can be given items	Location (Room)	Amount	Purpose of NPC
Slime	Yes	No	3, 9, 10	3	All slimes need to be defeated as one of the win conditions
Boss Slime	Yes	No	11 or 8	1	Moves between room 11 and room 8 every 4 commands (including

					incorrect ones). The Boss Slime need to be defeated as one of the win conditions
Seer	No	Yes	11	1	Needs to be given 3 relics as one of the win conditions
Blacksmith	No	Yes	5	1	Decorative NPC

Commands

The player's and the room's inventories are array lists of items. The character list in the room is also an array list, but of characters instead. Character and NPC are used interchangeably.

Command	Command description	Implementation
go [exit]	(default command) Goes to an adjacent room from the current room. The possible exits are north, west, east, and south, depending on if the room has these exits. Example: *go east*	A room stores exits (north, east, west, south) to other rooms using a hashmap. The go command utilises these exits to know which rooms to go to.
quit	(default command) Quits the game.	Stops the *play()* method when the quit command is called
exit	(default command) Quits the game.	Stops the *play()* method when the exit command is called
help	(default command) Prints out all possible commands of the game.	The help command prints the possible command words that have commands linked to them.
back	Goes to the previous room. If no previous room, then it will not work.	Stores the each room that has been traversed inside an array list called *roomHistory.* If back is called, sets the *currentRoom* to the last entry in *roomHistory.*
pickup [item]	Allows players to pick up pickable items. The only items that can be picked up are relics and swords. Only items in the current room can be picked up. Example: *pickup Sword*	If target item is pickable and exists in the current room, adds it to the player's inventory, and removes the target item from the current room.
drop [item]	Allows players to drop items from their inventory if the items exist. Example: *drop Sword*	If target item exists in the player's inventory, removes it from the player's inventory, and adds the target item to the current room. The logic is stored in the *dropItem()* method. The logic is stored in the *pickUp()* method.
viewInv	Allows players to view their inventory.	Prints out the player's current inventory. The logic is stored in the *viewInv()* method.
viewRoom	Allows players to view the room to see the room's items or characters.	Prints out the items and characters inside the current room. The logic is stored in the *viewRoom()* method.
viewMap	Prints out the minimap, and the current location of the player. The O represent the room and the X represents the player.	The map is designed like ASCII art. The O representing a room, is stored in an array. Every room has an *int roomNumber*. The current room's number is then used to locate the O in the array to change it to an X, as the player is currently there. The logic is stored in the *viewMap()* method.
interact [NPC or item]	Allows players to interact with an NPC or an item. All NPCs and items can be interacted with. Example: *interact Slime/interact Sword*	Interacting with NPCs and items returns and prints their attribute *string description.* The logic is stored in the *interact()* method.
give [NPC] [item]	Allows players to give NPCs items. Only seers can accept items, and the only items they can accept are relics. Example: *give Seer Relic*	If the target item exists in the player's inventory and can be given to the NPC, the target item will be removed from the player's inventory, and then added

		to the NPC's inventory. The logic is stored in the *give()* method.
attack [NPC]	Allows players to attack enemies if the player has a sword in their inventory. Only enemy NPCs, which are slimes and boss slimes, can be attacked. Example: *attack Slime*	If the player has a weapon (Sword) in their inventory, and the target NPC is an enemy, then the target NPC will be removed from the room's character list which means that the player has "defeated" the enemy. The logic is stored in the *attack()* method.
objectives	Views the current objectives. Once an objective is completed, it will not show up on the objectives list.	There are two objectives: Defeating all slimes and giving 3 relics to the seer. This is achieved by using the *viewObjectives()* method with the *objectives* command. The objectives that have not been completed will be printed out. If they have been completed, they will not print out.

Base and challenge tasks

Each task bullet point is mapped according to the task bullet points in the coursework task list.

- **Room implementation:** Rooms are objects that the player can traverse through. 11 rooms, according to the map presented above, were implemented by using the Room class. The *createGameMap()* method in the game class creates and initialises the rooms.
- **Items, inventories, and picking/dropping items:** Items are objects in the game implemented using the Item class. To see all the items, there is a table above. An Inventory class has an attribute called inventory, which is an array list of items, which represents an inventory holding items. A Player class is created to hold player information. Both the Player class and the Room class use the Inventory class, to create the player's inventory and the room's inventory. This allows players and rooms to store items in their inventory. Players can use the pickup and drop commands to pick up and drop items.
- **Item weights and inventory weight limits:** Items have a weight attribute, called *int weight*. The inventory has a weight limit, called *int weightLimit*. The Player class uses Inventory class's *getWeightLimit()* and *setWeightLimit()* to get and set the weight limit. *setWeightLimit()* is set as *static final int WEIGHT_LIMIT = 15* which means that players can only hold up to 15kg of items. There are 4 pickable items in the game: 3 are relics, and 1 is a sword. They all weigh 5kg each. This means that the player can only carry 3 items at once. Inside the *int getCurrentInventoryWeight()* method, I use for-each loops to check every item's weight in a player's inventory, and then add them up together to get the current total of the items' weights in the inventory. If *player.getCurrentInventoryWeight() + roomItem.getWeight() > player.getWeightLimit()*, then the player cannot pick the item up.
- **Winning conditions:** There are two conditions to win. The first condition is that the player must pick up the 3 relics and give them to the seer. The player must first pick up the relics scattered across the map. Then, the player must use the give command to give the seer 3 relics. The *boolean isSeerRelics()* method uses a for-each loop to check the seer's inventory. It will return true if the seer has 3 relics in their inventory, otherwise it will return false. The other winning condition is that the the player must pick up a sword, and then slay the enemies which are scattered across the map. This can be achieved by using the pickup, attack, and go commands. The *boolean isSlimesDead()* has a for-each loop that checks through the *roomList*, and then another for-each loop to check through the *characters,* to see if the slimes exist. If the slimes don't exist, then *boolean isSlimesDead()* returns true. Otherwise, it will return false. Once *boolean isSeerRelics()* and *boolean isSlimesDead()* return true, the *boolean isWin()* method will return true, and the *play()* method will end.
- **The back command:** When the go command is called, the current room is added to the array list *roomHistory* and then the current room becomes the next room. When the back command is called, the last entered item in *roomHistory* becomes the current room.
- **Adding at least 3 new commands (excluding the back command):** 9 new commands have been added. The commands table above shows how they have been implemented.
- **Adding at least 3 characters:** The character class is used to initialise characters. All characters are in a table above. One of the characters move, which is the Boss Slime. They move between room 11 and room 8 by removing them from one room and then adding them to the opposite room. The *moveBossSlime()* method has a counter that goes from 0 to 4 and backwards. This imitates a turn, and

every 4th turn the boss slime moves. A turn works by typing a command in, including incorrect ones. Rooms can store characters using an array list of characters called *characters.*

- **Extending to the parser to recognise three-word commands:** The three-word command was implemented by adding onto the previous framework. By mirroring how the second word is added, inside the command class, I added *getThirdWord(), boolean hasThirdWord()* and allowed the constructor to accept 3 command words. The parser class can accept three words and the scanner will check for a third word now.
- **Magic teleporter room:** The logic is implemented inside the go command. When you try to go to the room which is the magic teleporter room, there is a random class that Java provides, and it is used to generate a random room from 1-11 using modulus. (%10, because the roomlist is 0-indexed). The current room is set to the random room number provided.

Code quality considerations

I have tried my best to reduce coupling where possible. I have introduced classes such as item, inventory, player, character, and minimap which are abstractions and abstracts away the information to these classes. As expected from the names of the classes, the item class manages items and any information such as weight, name, and Booleans to see what kind of item it is such as relic, or weapon. The interactions with the items differ depending on the type. The inventory class manages the array list of items. The character, player, and room class use the inventory class. The character class stores information regarding characters, including the character inventory. There are Booleans inside the character class that can determine what type of character this is. The player class stores information regarding the player, including the player inventory. The minimap class stores the logic to print out the minimap.

Whilst more abstractions do increase coupling, these abstractions have helped me to get a highly cohesive and responsibly driven design to the code. I separated inventory to one class, because the player, room, and character all use an inventory. The room class was separate already. I could've avoided creating a player class as it only has an inventory, but I decided to because 2 other classes use an inventory, so it made sense to me to make a player class to handle its inventory. In addition, if there were to be more features introduced to the game, the player class seems logical, in case if features such as health, player names, and other logic were to be introduced.

All classes have private modifiers to their attributes, to prevent misuses of the class attributes. Every class has relevant getters and setters to receive information about attributes and set attributes to certain values. Alongside with class abstraction, this can help improve maintainability.

Walkthrough

This is one combination of commands to win the game.

"go east, pickup Sword, go east, attack Slime, go north, pickup Relic, go north, attack BossSlime , go east, attack Slime, pickup Relic, go west, go west, attack Slime, drop Sword, pickup Relic, go east, go north, give Seer Relic, give Seer Relic, give Seer Relic"

1. Pickup sword in room 2.
2. Defeat the slimes in room 3, 9, 10, and 11.
3. Drop the sword.
4. Pick up the 3 relics in rooms 4, 9, and 10.
5. Go to room 11 and give the 3 relics to the seer.
6. Congratulations! You have won.

As long as you defeat all the slimes, and give the 3 relics to the seer, the order of the tasks do not matter.