



TEST REPORT ON TEXT GAME V3.10

Abstract

We partitioned our Text Game V3.10 into two parts for review and testing – map parsing and game playing. In each part, we mainly take the strategies of White Box test and Branch test, as well as Boundary Test. We decomposed responsibilities among our team members aiming to balance work load and make people test functions not developed by them. Some bugs were found and fixed during the test.

Group 10 – Dan, Yue Ben, Nianzu Ma

2013/04/29

Testing for the Map Parsing involved a series of unit tests with various valid and invalid maps. The sample maps used as well as the results for each test case can be found on our groups Git Repository but the significant errors and resolutions are listed below:

- Places, artifacts, and our actors were previously allowed to not have descriptions (or questions in the case of actors). This has been changed to check each object for a description or question.
- Places, artifacts, and our actors were parsed on the assumption of parsing an object and then parsing its descriptions. But an `ArrayIndexOutOfBoundsException` exception was mishandled if a description came before any objects were created (for place, artifact, or actor). This has been changed to either make sure the correct object has been created or to stop parsing if no valid objects exist yet.
- Invalid integers would create mishandled `NumberFormatException` exceptions. A try / catch was used each time integers were parsed to properly handle invalid integers.
- The parsing order was not checked. In the case of Places and Paths, there was a decision statement added to verify that the string data was in the right order, but this could not be easily applied to Artifacts / Keys / Lights because it would require assumptions about valid input or a major code change.
- The number of parsed object did not match the section counter. The section counter is now parsed and checked with the number of objects parsed.
- Room ID's of 0 and 1 were parsable, but now the code checks for 0 and 1 room ID's.
- Lighting entries could be negative or above 100. Since 0 and 100 mark opposite boundaries of lighting, values outside of those ranges are deemed invalid now.
- Movability of artifacts could be negative or above 1000. There is now a decision to check for valid values of movability.

The following code segment was Inspected for a reoccurring error within parsing:

```
//taken from main
while(true){
    //String filename = br.readLine();
    env.initEnvironment(filename);
    if(env.message.contains("ERROR")){
        System.out.println(env.message+"\n" +
            "Please input the correct environment file path:");
        filename = br.readLine();
        continue;
    }else{
        System.out.println(env.message+"\n");
        break;
    }
}
```

The major issue observed was after an error was detected, the new parsed data was added to the

pre-existing environment. Since there was also nothing in the Environment.java clearing the invalid data, the resolution was to create a new environment after a new filename is assigned.

- Version control is updated after testing. Before testing the map version was not checked, now versions 1.*, 2.* and 3.10 are allowed. Lower-version maps cannot have the format that's only defined for the higher-version maps, which is checked while parsing the map, mainly realized in the initEnvironment() function in the Environment class. All commands are allowed for all versions (see executeToken() function in the game class)— an alternative design is also realized but not in use now (see executeTokenLimitingVersion() function in the game class).

As for game playing, the extractToken() function (in the Token class) is proved to provide correct and formatted Token to executeToken() function (in the Game class). After inspecting and testing concrete command execution for game playing, we fix or strengthen some functions listed below:

- In the executeAnswer() function, it's added to check whether there is an Actor in the room to avoid NullPointerException.

Responsibilities Decomposition

In attached table below we roughly list the key developers for each main function cluster, and also the reviewer/tester and modifier/developer. Actually during the cascading development and version update, for almost every part, every one of we three has been involved.

Main Code Cluster	Develop	Review, Test	Modify
Map Parser (Environment class)	Cindy, Dan	Dan	Dan, Cindy
Version Control (Environment, Game)	Dan	Cindy	Cindy
extractToken (Token class)	Cindy, Dan	Cindy	
executeToken (Game class)	Cindy, Dan, Nianzu	Cindy, Dan	Cindy
executeGo&Exit (Game class)	Cindy, Dan, Nianzu	Nianzu	
executeLook (Game class)	Nianzu, Cindy, Dan	Nianzu	
executeQuit&Help (Game class)	Cindy	Cindy	
executeGet&Drop&Inve (Game class)	Dan	Cindy	
executeUse (Game class)	Nianzu, Dan, Cindy	Nianzu	
executeAsk&Answer (Game class)	Dan, Cindy	Nianzu, Cindy	Nianzu

Test Map Parsing

We have reviewed, debugged as well as tested the map parsing part of our game in the following key sub-sections: version control, parsing Places, parsing Paths, Parsing Lighting, and parsing Artifacts.

Test Version Control

Game versions 1.*, 2.* and 3.10 are accepted. For every version, all commands are allowed.

Test version number extraction from the GDF line in the map file

The version number "(int)a.(int)b" is stored in a int[2], with int[0] for "a" and int[1] for "b".

```
static void testVersion() throws IOException{
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    while(true){
        int[] d = new Environment().parseVersionNumber(br.readLine());
        if(d==null) {System.out.println(d);continue;}
        for(Integer dd: d)
            System.out.println(dd);
    }
}
```

Following test cases cover normal situations and possible exceptions.

3.10 //input 3 //output: version[0] 10 //output: version[1], thus v3.10	2. null
gdf 3.10 normal game 3 10	2 null
3.10 +3.1 -2.0 3 10	2. 2 Null //since there is a space inside

Test version control in real games

The test cases below are selected according to Equivalence Testing and Branch Testing, for every possible situation, we select certain test cases presenting the functions we have.

- Test case: version 1.0
...\\CS440\\Code\\Text_Game_V3.10\\TestTextGame\\TestMapParsing\\Version
TestSuite\\sixRooms.gdf
Game environment: GDF 1.0 The Wilderness
Input commands to wander around and find the exit.
Type HELP for instructions at any time.
Let's begin!

Result: the program accepts this game, and all commands are allowed. And for commands came up for higher versions, such as USE/GET/DROP/INVE/ASK/ANSWER, the player will get messages such as "no actor here" (for ASK) and so on.

- Test case: version 1.9, same as v1.0 actually

```
...\CS440\Code\Text_Game_V3.10\TestTextGame\TestMapParsing\Version  
TestSuite\sixRooms-fakeV1.9.gdf
```

Game environment: GDF 1.9 The Wilderness

Input commands to wander around and find the exit.

Type HELP for instructions at any time.

Let's begin!

Result: the program accepts this game as v1.9, and all commands are allowed.

- Test case: version 2.0 - the version number is fake, its content is still under version 1.0 actually, and only one path has lock-pattern.

```
...\CS440\Code\Text_Game_V3.10\TestTextGame\TestMapParsing\Version  
TestSuite\sixRooms-fakeV2.0.gdf
```

Game environment: GDF 2.0 The Wilderness

Input commands to wander around and find the exit.

Type HELP for instructions at any time.

Let's begin!

Result: the program accepts this game as version 2.0, and all paths, except the one with a lock-pattern in the map file, are assigned the default lock-pattern, which is unlocked. All commands are allowed to use.

- Test case: v1.0 containing a false Path

```
...\CS440\Code\Text_Game_V3.10\TestTextGame\TestMapParsing\Version  
TestSuite\sixRooms-falseV1.0.gdf
```

Game environment: GDF 1.0 The Wilderness

ERROR: map V1.0 does not allow this many integers for path 1!

- Test case: v1.0 containing Lighting section

```
...\CS440\Code\Text_Game_V3.10\TestTextGame\TestMapParsing\Version  
TestSuite\sixRooms-falseV1.1.gdf
```

Game environment: GDF 1.1 The Wilderness with Lighting

ERROR: map version under 2.0 doesn't allow Lighting entries!

Temporary Conclusion: for fake version numbers, we allow lower version maps with a higher fake version numbers, but not the contrary. For example, a map v1.0 containing Lighting section is not allowed. Also map v1.0 cannot have a Path with more than 3 numbers - which is the format for version 2.0 and above.

- Test case: v 3.10

```
...\CS440\Code\Text_Game_V3.10\TestTextGame\TestGamePlaying\MystiCity20_V3.10_si  
mplified.gdf
```

Game environment: GDF 3.10 MystiCity

Input commands to wander around and find the exit.

Type HELP for instructions at any time.

Let's begin!

Result: the program accepts this game, and all commands are allowed.

- Test case: version number is 3.5

...\\CS440\\Code\\Text_Game_V3.10\\TestTextGame\\TestMapParsing\\VersionTestSuite\\sixRooms-fakeV3.5.gdf

Game environment: GDF 3.5 The Wilderness

ERROR: we currently accept only map version 1.*, 2.* and 3.10

Result: the program doesn't accept this game, as we want.

Conclusion

This is only one possible realization of the version control, and it works well now, introducing more freedom of writing map file. We have also realized another way of restricting available commands based on game version, but leave it unused.

Test Places

All test cases for this section are stored in Git repository

\\CS440\\Code\\Text_Game_V3.10\\TestTextGame\\TestMapParsing\\PlacesTestSuite

misspelledPLACE.gdf:

Parsing was stopped but using alternative filenames (as prompted to do) did not change error message except to an invalid filename.

misspelledPLACESM.gdf:

Parsing was stopped but using alternative filenames (as prompted to do) did not change error message except to an invalid filename.

PLACES_no#.gdf:

Parsing was stopped but using alternative filenames (as prompted to do) did not change error message except to an invalid filename.

PLACES0.gdf:

Parsing was stopped, same problem as mentioned above

PLACEScomments.gdf:

Parsing was stopped and the error did not change with a new file name. The third Place was not reached because parsing was stopped during the second place.

PLACESdescriptions.gdf:

Parsing was completed. Results showed three descriptions (two extra) for the Ogre's Lair and no descriptions for the Troll's Bridge.

PLACESduplicateID.gdf:

Parsing was stopped. The duplicate ID was detected and the system prompted for another filename. The error message changed when a file could not be found but to nothing else.

PLACESmaxINT.gdf: - java.lang.NumberFormatException

Parsing caused an exception. The place ID number was too large for a 32 bit signed integer.

PLACESminINT.gdf: - java.lang.NumberFormatException

Parsing caused an exception. The place ID number was too large (negatively) for a 32 bit signed integer.

PLACESnoID.gdf: - java.lang.ArrayIndexOutOfBoundsException

Parsing caused an exception. The name of the place was disregarded as a useless line, and the description was added to a non-existent Place, which threw the exception

PLACESnoNAME.gdf:

Parsing was stopped but using alternative filenames (as prompted to do) did not change error message

PLACESoutOfOrder.gdf:

Parsing was finished. The alternate order for ID and name were parsed properly despite the blatant order reversal.

PLACESoverestimated.gdf:

Parsing was finished. The proposed third place was not searched for and never erroneously added.

PLACESroomID0.gdf:

Parsing was finished. The reserved room ID of 0 was assigned to the parsed place.

PLACESroomID1.gdf:

Parsing was finished. The reserved room ID of 1 was assigned to the parsed place.

PLACESunderestimated.gdf:

Parsing was finished. The extra unaccounted place was added without any error.

Conclusion

Parsing descriptions needs to be more rigorous. Potentially for this is to make sure the last line parsed created a place or was a description. Room ID's of 0 and 1 are allowed, but more information is needed to determine if this is a map creator problem or a parsing problem. Under or over estimating the Places follows the same rules, as the Room ID's. Even if integers outside of the possible ranges of signed integers are not allowed, they should not cause an exception and need to be changed as well. Finally, the error detected that was not associated with places involved being unable to provide a valid filename after parsing was stopped from an invalid gdf file. The only

difference noticed after the initial parsing error message is providing a incorrect filename locked the message (and state of the program) to continuously detect a non-existent file name.

Shorter version:

- Only assign descriptions if the previous line parsed was a description or a valid ID and place name line.
- A place should not be allowed to have no description (places can have no descriptions now).
- Check to make sure numbers are within integer ranges
- If parsing is stopped, allow for new files to be parse (and for a way to quit at this stage)
- More information is needed if reserved room ID's and over or under estimating nPlaces is allowable

Test Paths

All test cases for this section are stored in Git repository

`\CS440\Code\Text_Game_V3.10\TestTextGame\TestMapParsing\PathsTestSuite`

`misspelledPATH.gdf`:

Parsing was completed. No paths were detected.

`misspelledPATHSM.gdf`:

Parsing was completed. No paths were detected.

`Paths_no#.gdf`:

Parsing was completed. The paths section was detected but not having any path information did not cause an exception or stop parsing.

`PATHS0.gdf`:

Parsing was completed. Although 0 paths should have technically been parsed, the 1 path included was parsed.

`PATHSbadDestination.gdf`

Parsing was completed. The invalid path was not erroneously parsed, and was probably disregarded because it did not have enough integers.

`PATHSbadDirection.gdf`

Parsing was completed. Although it was intended that north as a direction would not be parsed, north was assigned the direction.

`PATHSbadSource.gdf`:

Parsing was stopped. The invalid Place ID was checked, and the system prompted the entry of a valid file path. But the error message only changed when a non-existent file path was entered (to No environment file found) and the program could not be moved out of this state.

PATHSdecimalID.gdf: - java.lang.NumberFormatException

Parsing caused an exception. The decimal number could not be interpreted as an integer and an exception was thrown.

PATHSmaxInt.gdf - java.lang.NumberFormatException

Parsing caused an exception. The number was outside the acceptable integer ranges and could not be interpreted as an integer.

PATHSminInt.gdf - java.lang.NumberFormatException

Parsing caused an exception. The number was outside the acceptable integer ranges and could not be interpreted as an integer.

PATHSnoDirection.gdf:

Parsing was stopped. The non-existence of direction for a path stopped the parsing, and the system prompted for a new file path. The error message only changed when a non-existent file path was entered (to No environment file found) and the program could not be moved out of this state.

PATHSoutOfOrder.gdf:

Parsing was completed. Despite blatantly reordering the direction and source, the parsing properly detected the intended information.

PATHSoverestimated.gdf:

Parsing was completed. The parsing did not detect that the number of parsed paths was not the same as indicated to be in the file.

PATHSplacesAfter.gdf:

Parsing was stopped. The path parsing was stopped when the place ID could not be verified, and the same problem as documented above occurred when prompted to enter a new file name.

PATHStooFewInts.gdf:

Parsing was completed. The parsing disregarded the path with too few integers and continued to parse the other paths.

PATHStooManyInts.gdf:

Parsing was completed. The parsing disregarded the path with too many integers and continued to parse the other paths.

PATHSunderestimated.gdf:

Parsing was completed. The parsing did not stop when it had reached the “predetermined” number of paths in the file.

PATHSworseDirection.gdf:

Parsing was stopped. The invalid direction was detected, stopped the parsing, and prompted the demand for a new file path. But the same error as mentioned above occurred with being unable to change the state of the program to anything other than No environment file found, and then nothing else from that.

Conclusion

The biggest omission was not storing Path ID's. This is most likely due to a design change that originally included PlaceOut objects for the Path functionality which eventually led to the removal of the Path class. Number formatting was the only major error strictly related to sing paths, with integers outside the ranges of acceptable integers as well as decimal numbers being parsed to integers. Being unable to reenter a file name when parsing is stopped is also another error.

Test Lighting

All test cases for this section are stored in Git repository

\CS440\Code\Text_Game_V3.10\TestTextGame\TestMapParsing\LightingTestSuite

LIGHTINGlevel0.gdf:

Parsing was completed. The indicated light level was placed in the correct room.

LIGHTINGlevel100.gdf:

Parsing was completed. The indicated light level was placed in the correct room.

LIGHTINGlevel101.gdf:

Parsing was completed. The indicated light level was above possible values, and it was disregarded upon parsing.

LIGHTINGmaxInt.gdf - java.lang.NumberFormatException

Parsing caused an exception. Number was outside the ranges of possible 32 bit signed integer values.

LIGHTINGminInt.gdf - java.lang.NumberFormatException

Parsing caused an exception. Number was outside the ranges of possible 32 bit signed integer values.

LIGHTINGnegative.gdf:

Parsing was completed. The indicated light level was below possible values, and it was disregarded upon parsing.

LIGHTINGnonInt.gdf:

Parsing was completed. The non integer place ID number as well as light level did not cause an exception as previous non integers have, and the invalid integers were ignored upon parsing them

LIGHTINGplace0.gdf:

Parsing was completed. The indicated light level was assigned to all places, as intended by the inclusion of place 0.

LIGHTINGtooFewInts.gdf:

Parsing was completed. The light entry was disregarded because there were too few integers.

LIGHTINGtooManyInts.gdf:

Parsing was completed. The light entry was disregarded because there were too many integers.

Test Artifacts

All test cases for this section are stored in Git repository

\CS440\Code\Text_Game_V3.10\TestTextGame\TestMapParsing\ArtifactTestSuite

ARTIFACTSbadPlace.gdf

Parsing was stopped. The invalid place ID was caught and the system prompted the entry of a new gdf file.

ARTIFACTSduplicateID.gdf

Parsing was stopped. The duplicate Artifact ID's were detected and the system prompted the entry of a new gdf file.

ARTIFACTSmovable-1.gdf

Parsing was completed. Movability of the artifact was set to -1.

ARTIFACTSmovable1001.gdf

Parsing was completed. Movability of the artifact was set to 1001.

ARTIFACTSnoArtifact.gdf

Parsing was stopped. The non existent artifact for a particular description was detected and the system prompted the entry of a new gdf file.

ARTIFACTSnoDescription.gdf

Parsing was stopped. The non existent description for a particular artifact was detected and the system prompted the entry of a new gdf file.

ARTIFACTSoverestimated.gdf

Parsing was stopped. The mismatch between reported and actual artifacts was detected and the system prompted the entry of a new gdf file.

ARTIFACTStab.gdf

Parsing was completed. The tab in the name was artifact name was removed.

ARTIFACTStooFewInts.gdf

Parsing was stopped. The shortage of integers was detected and the system prompted the entry of a new gdf file.

ARTIFACTStooManyInts.gdf

Parsing was completed. The extra integer was assumed as part of the artifact name.

ARTIFACTSunderestimated.gdf

Parsing was stopped. The mismatch between reported and actual artifacts was detected and the system prompted the entry of a new gdf file.

Conclusion

Only two code changes need to be implemented following the results. There needs to be a movability check so invalid numbers are not assigned. And names should not contain tabs.

Test Game Playing

We divided game playing into subsections as token extraction, execute commands overview, and concrete exetutions for each commands. White box testing and branch testing, as well as some boundary testing are the main techniques we use.

Test maps are available in the Git repository \CS440\Code\Text_Game_V3.10\TestTextGame\TestGamePlaying\.

Test Token Extraction

The token extraction function is realized in the class “token”. Its function is to extract tokens from user inputs, ready for providing commands for execution.



Figure 1 extract token (UML activity diagram) final states omitted

Using code review, we ensure the function is case-insensitive, also “\t” is replaces by a single white space, and extra white spaces are tolerated and removed from token. A test function presented below is used to test this function. It reads user input from a line and outputs corresponding Token, or error message.

```
static void testTokenExtraction() throws IOException{
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    while(true){
        String userInput = br.readLine();
        Token token = Token.extractToken(userInput);
        if(token == null){
            System.out.println("ERROR: Invalid Token."); //Command=Optional=null
            continue;
        }
        System.out.println("Command: "+token.command+", Optional: "+token.optional);
    }
}
```

- Verify Direction Validation

Using code inspection, we ensure the isValidDirection function can work correctly to test whether an input string is a valid predefined direction. All the directions (including abbreviations) are spelled correctly and stored in the enum Direction, and the isValidDirection function checks the input string against all directions.

```
public enum Direction {
    N, NORTH, S, SOUTH, E, EAST, W, WEST,
    U, UP, D, DOWN,
    NE, NORTHEAST, NW, NORTHWEST, SE, SOUTHEAST, SW, SOUTHWEST,
    NNE, NORTHNORTHEAST, NNW, NORTHNORTHWEST,
    SSE, SOUTHSOUTHEAST, SSW, SOUTHSOUTHWEST,
    ENE, EASTNORTHEAST, ESE, EASTSOUTHEAST,
    WNW, WESTNORTHWEST, WSW, WESTSOUTHWEST,
}
```




```
static boolean isValidDirection(String input) {
    for (Direction d : Direction.values())
        if (d.name().equalsIgnoreCase(input.trim()))
            return true;
    return false;
}
```

- Test Direction Abbreviation (token.abbreviateDir())

Using static code review, we ensure the correctness of the function abbreviateDir(), which return the short capital abbreviation for valid direction. Also later we’ll partially test this function together with the overall token extraction function and all other game playing functions.

- Test Token Extraction

In the table below are some test cases as invalid tokens. In each cell in the table there are two lines, the 1st line is the test input, and the 2nd line is the output.

Invalid token test cases - misspelling	Invalid tokens – special characters
goe ERROR: Invalid Token.	 ERROR: Invalid Token. //input: tab
e xit ERROR: Invalid Token.	 ERROR: Invalid Token. //input: white space
 ERROR: Invalid Token. //input: Enter	// ERROR: Invalid Token.

In the table below, each two lines make one test case: the 1st line is the test input, and the 2nd line is the output.

1 st line//test input 2 nd line//output token, all capitalized	
look Command: LOOK, Optional: null get Command: GET, Optional: null drop Command: DROP, Optional: null use Command: USE, Optional: null inve Command: INVE, Optional: null inventory Command: INVENTORY, Optional: null Help Command: HELP, Optional: null QUIT Command: QUIT, Optional: null exit Command: EXIT, Optional: null	Go southeast Command: GO, Optional: SE //input: 3 spaces before "Go" and 3 after look southwest Command: LOOK, Optional: SW //input: look Direction look here Command: LOOK, Optional: HERE look artifact Command: LOOK, Optional: ARTIFACT look Command: LOOK, Optional: null //input: " "+ "look" + "\t" LOOK SOUTHSOUTHWEST Command: LOOK, Optional: SSW //input: "\t" + "LOOK" + " " + "SOUTHSOUTHWEST" get Command: GET, Optional: null // input: "GET" + a single white space
go ERROR: Invalid Token. go e Command: GO, Optional: E go west Command: GO, Optional: W go artifact ERROR: Invalid Token.	use any artifact Command: USE, Optional: ANY ARTIFACT drop any artifact in the inventory Command: DROP, Optional: ANY ARTIFACT IN THE INVENTORY get artifacts in the place Command: GET, Optional: ARTIFACTS IN THE PLACE
ask Command: ASK, Optional: null ask question ERROR: Invalid Token. answer Command: ANSWER, Optional: null answer myAnswer Command: ANSWER, Optional: MYANSWER answer a long answer containing numbers like (1) Command: ANSWER, Optional: A LONG ANSWER CONTAINING NUMBERS LIKE (1)	

All test cases are selected to cover all branches in this extractToken() function.

Conclusion

All test results are the same as we expect and we want, but the game cannot deal with situation where an Artifact's name is the same as a Direction.

Test Execute Commands (executeToken())

The whole executeToken() function is roughly presented in the activity diagram below. By code review, we ensure it goes to different branches correctly. Each branch will be tested separately. While testing the overall function and each branch, we take the Branch Testing strategy to cover every possible branch. Also for some branches, there are more detailed activity diagrams attached in concrete sections.



An alternative design – limit commands according to game version

We actually have an alternative design as a backup solution for game playing, where we don't allow certain commands in lower game versions.

```
//For different game version, certain commands are limited
void executeTokenLimitingVersion(Environment env, Token token){
    message = "";
    if(token.command.equalsIgnoreCase("HELP")) {this.executeHelp();return;}
    if(token.command.equalsIgnoreCase("QUIT")) {this.executeQuit();return;}
    if(token.command.equalsIgnoreCase("EXIT")) {this.executeExit(env,token);return;}
    if(token.command.equalsIgnoreCase("GO")) {this.executeGo(env,token);return;}
    if(token.command.equalsIgnoreCase("LOOK")) {this.executeLook(env,token);return;}
    if(env.version[0]<2){
        message = "Invalid Token in game version " +env.version[0]+ "." +env.version[1]+", "+
            "Type correct commands or HELP for instructions.\n";
        return;
    }
    if(token.command.equalsIgnoreCase("INVE") || token.command.equalsIgnoreCase("INVENTORY"))
        {this.executeInventory();return;}
    if(token.command.equalsIgnoreCase("DROP")) {this.executeDrop(env, token);return;}
    if(token.command.equalsIgnoreCase("GET")) {this.executeGet(env, token);return;}
}
```



```

        if(token.command.equalsIgnoreCase("USE")) {this.executeUse(env, token);return;}
        if(env.version[0]<3){
            message = "Invalid Token in game version " +env.version[0]+"."+env.version[1]+" "+
                "Type correct commands or HELP for instructions.\n";
            return;
        }
        if(token.command.equalsIgnoreCase("ASK")) {this.executeAsk();return;}
        if(token.command.equalsIgnoreCase("ANSWER")) this.executeAnswer(env, token);
    }
}

```

Test Commands Help and Quit

As long as the game is started, it doesn't matter when we execute the "HELP" command, nor the "QUIT" command. So for this test, we just input "help" and "quit" after the game begins, using the standard game entrance in the "main" class.

Note: case-insensitivity and white spaces and so on in user input are tested more systemically in testTokenExtraction.

Test Command HELP

- Enter the game as if playing, through "main" function in the "main" class.
- Input: help/HELP/Help
- Expected result: the program returns hints for game
- Test result: the program returned message as bellow

Valid inputs: "HELP", "QUIT", "EXIT", "GO" Direction, "LOOK" [Direction/"HERE"/Object], "INVENTORY", "GET"/"DROP"/"USE" [Object].
 Valid Directions: N,NORTH, S,SOUTH, E,EAST, W,WEST, U,UP, D,DOWN, NE,NORTHEAST, NW,NORTHWEST, SE,SOUTHEAST, SW,SOUTHWEST, NNE,NORTHNORTHEAST, NNW,NORTHNORTHWEST, SSE,SOUTHSOUTHEAST, SSW,SOUTHSOUTHWEST, ENE,EASTNORTHEAST, ESE,EASTSOUTHEAST, WNW,WESTNORTHWEST, WSW,WEISTSOUTHWEST.

- Conclusion: HELP command is executed correctly.

Test Command QUIT

- Enter the game as if playing, through "main" function in the "main" class.
- Input: quit/QUIT/Quit
- Expected result: the program ends the game
- Test result: Game quitted, and the program returned message as bellow

Game quit. Thanks for coming.

- Conclusion: QUIP command is executed correctly.
- Suggestion: confirmation might be asked for before ending the game.

Test Commands Inve/Inventory

In the runtime, items in the player's inventory are stored in a global variable "inventory" in the game class. By inspection and debugging, it's shown the execution of command Inve/Inventory outputs the exact items in the variable inventory.

```

void executeInventory() {
    if(inventory.size() == 0)
        message = "*No artifacts in your inventory.\n";
}

```

```

else{
    message = "*Artifacts in your inventory:\n";
    for(int i = 0; i < inventory.size(); i++)
        message += inventory.get(i).name + "\n";
}
}

```

- Input: inve/inventory
- Expected result: the program returns correct message letting the player know what are the items in his inventory

1) Test case: empty inventory.

In the map "MystiCity20_V3.10_simplified.gdf", type "inve" right after the game begins. (Using any correct map of current versions, the inventory should be empty right after the map is loaded and the game begins.)

By debugging we see the real-time content in the inventory as below:

inventory	ArrayList<E> (id=25)
elementData	Object[10] (id=37)
modCount	0
size	0

[]

System output (the first line is input test command):

```

inve
*No artifacts in your inventory.

```

2) Test case: non-empty inventory.

In the map "MystiCity20_V3.10_simplified.gdf", type "inventory" after picking up the Leather bag in the Entrance hall and the Brass lantern in the Pool of Enchantment, no "Drop" or "Use" commands ever called.

By debugging we see what's in the inventory as follows:

inventory	ArrayList<E> (id=25)
elementData	Object[10] (id=37)
[0]	Light (id=42)
description	ArrayList<E> (id=48)
ID	1
lightActive	false
lightLevel	25
movability	1
name	"Brass lantern" (id=49)
placeID	-1
type	2
usedEver	true
value	30
[1]	Artifact (id=45)
description	ArrayList<E> (id=46)
ID	8
movability	1
name	"Leather bag" (id=47)
placeID	-1
type	0
usedEver	true
value	20

[Light@c0b3d1, Artifact@15d63da]

Test result: the program returned message as bellow (the first line is input test command):

```
inventory
*Artifacts in your inventory:
Brass lantern
Leather bag
```

Conclusion

The executeInventory() function is functioning well, presenting the real-time content in the inventory.

Test Command Ask and Answer

By code review, we ensure Ask and Answer commands are only allowed in game version 3.10, and this is realized in the executeToken() function in the game class.

As long as there is an actor in some place, the player has to answer the actor's riddle correctly before GOing to other places or EXITing the game – going back to previous place is allowed. Once the riddled is figured out, the actor will disappear; if the player's answer is wrong, he will be sent back to the starting place of this map.

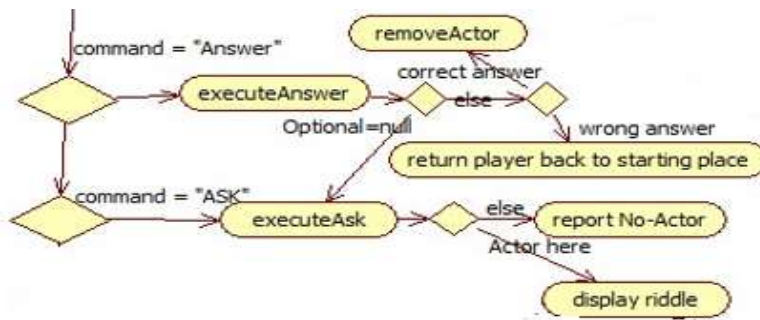
The ASK command comes with no parameter, and its function is to ask for description of the riddle in the current place, if there is an actor here holding a riddle.

```
void executeAsk() {
    if(currentPlace.actor != null){
        message = "*Actor " + currentPlace.actor.name+ " is here. " +
            "Type \"ANSWER yourAnswer\" to solve his riddle before moving on:\n";
        for(int i=0;i<currentPlace.actor.question.size();i++){
            message += currentPlace.actor.question.get(i)+"\n";
        }
    }else message = "*No actor here. You can move on.\n";
}
```

While the ANSWER command can come with or without parameter: if it comes alone, it is equal to the ASK command; else it should have only one parameter - the player's answer (as a string) to the riddle. String comparison is used to check the correctness of the answer, case-insensitive.

```
void executeAnswer(Environment env, Token token) {
    if(currentPlace.actor == null){
        message = "*No actor here. You can move on.\n";
    }else{
        if(token.optional == null){
            executeAsk();
        }else if(token.optional.equalsIgnoreCase(this.currentPlace.actor.answer)){//answer correctly
            message = "Correct! You can move on now.\n";
            this.currentPlace.actor = null;//remove the actor once his riddle correctly answered
        }else{
            this.currentPlace = env.places.get(0);
            message = "Sorry you failed. Now you are back at the starting place - "+currentPlace.name+".\n";
        }
    }
}
```

The activity diagram below shows the flow of executing these two commands.



By code review, the functions look good to go. Then we do testing as shown next.

- Test environment: MystiCity20_V3.10_simplified.gdf

There are 2 actors¹ in this map:

16 106 Ogre g

**What always ends everything?*

5000 13 Riddler wrong clock

**Twice every single day, I am correct.*

**But I am wrong otherwise.*

**What am I?*

We do tests only on the more complicated one in Room 13. Every test case is tested **independently** in **separated runs**. They all happen when the first time the player comes into Room 13... ("Go N" command below lets the player goes from room ID 12 to Room ID 13)

go n

**Actor Riddler is here. Type "ANSWER yourAnswer" to solve his riddle before moving on:*

Twice every single day, I am correct.

But I am wrong otherwise.

What am I?

- Test cases and results:

1) ask

ask

**Actor Riddler is here. Type "ANSWER yourAnswer" to solve his riddle before moving on:*

Twice every single day, I am correct.

But I am wrong otherwise.

What am I?

Test result: correct, as expected, displaying the riddle (actor's name included).

2) Go to previous place without solving the riddle

go s

**This is Entrance Hall.*

Test result: correct as expected. Successfully come back to previous place.

3) Go to a new place without solving the riddle

go e

**Actor Riddler is here. Type "ANSWER yourAnswer" to solve his riddle before moving*

Twice every single day, I am correct.

But I am wrong otherwise.

What am I?

¹ Actor format:

ID placeID name long_answer_with_spaces

**Riddle description*

Test result: correct as expected. Going to another place than the previous one is not allowed without solving the riddle.

4) answer

```
answer
*Actor Riddler is here. Type "ANSWER yourAnswer" to solve his riddle before moving on:
Twice every single day, I am correct.
But I am wrong otherwise.
What am I?
```

Test result: correct, as expected, same as executing command ASK.

5) wrong answer 1

```
answer wrong answer
Sorry you failed. Now you are back at the starting place - Entrance Hall.

look
*This is Entrance Hall.
```

Test result: correct, as expected, transported to the starting place.

6) Wrong answer 2

```
answer wrong clock
Sorry you failed. Now you are back at the starting place - Entrance Hall.
```

Test result: correct as expected, but showing the vulnerability of current design and implementation – only exact same string can be accepted (case insensitive but white spaces in the middle are not allowed).

7) Correct answer 1

```
answer Wrong Clock
Correct! You can move on now.

go e
It is too dark here. Try adjusting the lights or returning to your previous room.
```

Test result: correct as expected. After solving the riddle, executing “GO another_Direction” command as normal.

Continue on this test run: go to any other place and come back to Room ID 13 again...

```
go s
*This is Entrance Hall.
You are standing in the entrance hall of the great six-room dungeon
There are doors to the east and north, and a stairway leading down
The main exit ( from the game ) is to the west

go n
*This is Ogre's Lair.
You have entered the Ogre's Lair! Better leave before he wakes up . . .
There are doors to the south and the east
```

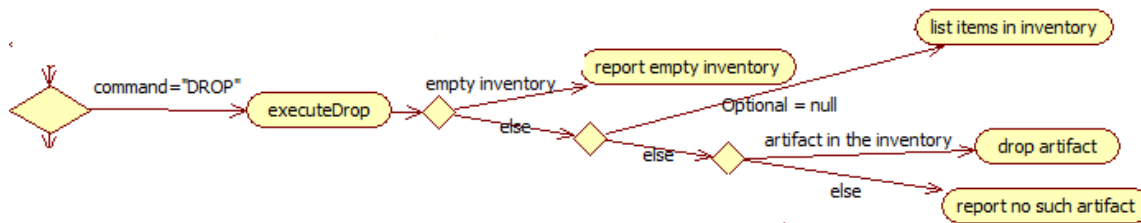
Test result: correct as expected. After solving the riddle, the actor disappears from the place.

Conclusion

During the test we found and fixed a bug for not checking whether there is an Actor in the room when executing Answer. After that the command is correctly implemented as designed, but vulnerable since checking correct answer is not flexible.

Test Command Drop

The DROP command discard specified artifact from inventory. The activity diagram below shows the detailed functioning of executing the command.



By code review, the functions look good to go. It first checks the number of artifacts in the inventory; then check if an artifact is specified; finally it checks if the specified artifact exists in the inventory. Following each check there is proper action.

Now we do testing as shown next.

- Test environment: MystiCity20_V3.10_simplified.gdf

- Test cases and results:

8) DROP (empty inventory)

Right after the game starts, the inventory is empty. Type “drop”.

```
drop
There is nothing in your inventory to drop.
```

Test result: correct as expected, displaying the proper message.

9) DROP (non-empty inventory)

After the game starts, Go East to the Pool of Enhancement and pick up the Brass Lantern there, thus the inventory is not empty. Type “inve” to see items in the inventory, then type “drop”.

```
inve
*Artifacts in your inventory:
Brass lantern
```

```
drop
Artifacts to drop:
Brass lantern
```

Test result: correct as expected, displaying the artifacts in the inventory.

10) DROP Artifact_NOT_in_the_inventory (non-empty inventory)

Following prior test 2), type “DROP something not existing”.

```
DROP something not existing
SOMETHING NOT EXISTING is not in your inventory, type INVENTORY/INVE to see what is in your inventory.
```

Test result: correct as expected. Report no such item in the inventory, also give suggestions.

11) DROP Artifact_in_the_inventory (non-empty inventory)

Following prior test 3), type “Drop brass lantern”. Also execute “inve” and “look here” commands before and after executing the “Drop brass lantern” command, showing the change of artifacts in the inventory and in the place.

```

inve
*Artifacts in your inventory:
Brass lantern

look here
There is no artifact in this room.

```

```

Drop brass lantern
Dropped BRASS LANTERN on the ground, will stay in Pool of Enchantment.

```

```

inve
*No artifacts in your inventory.

look here
*Artifacts in this room:
Brass lantern

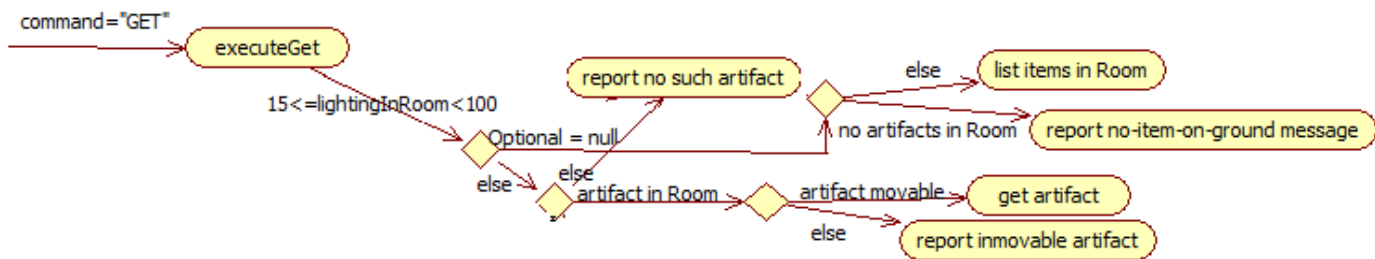
```

Test result: correct as expected, the specified artifact is removed from the inventory and left in the place.

Conclusion

The DROP is designed and implemented properly.

Test command Get



Inspect Lighting restriction

By statically reviewing the code, we ensure the lighting restrictions on executing the GET command: if lighting is not supporting, the program will not allow the player to GET artifacts, and it will respond corresponding messages.

```

void executeGet(Environment env, Token token){
    if((carryingLightValue + currentPlace.lightLevel + currentPlace.lightValueOnGround) < 15){
        message += "Too dark to see what is on the ground. Try turning on a light.\n";
        return;
    }
    if((carryingLightValue + currentPlace.lightLevel + currentPlace.lightValueOnGround) >= 100){
        message += "Too bright here to see what is on the ground. Try turning off a light.\n";
        return;
    }
}

```

1) Test Case: get (there is artifact in the place, light level is 0 here)

After the game starts, go N to Place ID 13. Type "get".

```

get
Too dark to see what is on the ground. Try turning on a light.

```

Test result: correct as expected, displaying corresponding message.

Above is an example test case for where light level is lower than 15. By inspection we've guaranteed the correctness of lighting restrictions, also we have similar detailed tests on lighting restrictions when testing the command LOOK, thus no more test cases listed here regarding lighting.

Test GET in different situations

Test Environment: MystiCity20_V3.10_simplified.gdf

The following tests are **independent** and done in **different runs** of the same game – all in a room where lighting is between 15 (included) and 100 (excluded).

1) Test Case: get (no artifact in the place)

After the game starts, go E then go Down to the Potions Lab. Type “get”.

```
look here
There is no artifact in this room.
```

```
get
There is nothing here.
```

Test result: correct as expected, displaying corresponding message.

2) Test Case: get Artifact (no artifact in the place)

After the game starts, go E then go Down to the Potions Lab. Type “get an artifact”.

```
look here
There is no artifact in this room.
```

```
get an artifact
There is nothing here.
```

Test result: correct as expected, displaying corresponding message which is same as last test case.

3) Test Case: get (there is artifact in the place)

Right after the game starts, type “get” in the starting place (ID 12).

```
look here
*Artifacts in this room:
Leather bag
```

```
get
*Artifacts in the room:
Leather bag
```

Test result: correct as expected, displaying artifacts in the room.

4) Test case: get Artifact that exists in the place (there is artifact in the place)

Right after the game starts, type “Get leather bag” in the starting place (ID 12). Also use “look here” and “inve” commands before and after, to see the change of artifacts in the place and in the inventory.

```
look here
*Artifacts in this room:
Leather bag
```

```
inve
*No artifacts in your inventory.
```

```
Get leather bag
Picked LEATHER BAG up, it is now in your inventory.
```

```
look here
There is no artifact in this room.
```

```
inve
*Artifacts in your inventory:
Leather bag
```


Test result: correct as expected, specified artifact is moved from the ground to the inventory.

5) Test case: get Artifact that doesn't exist in the place (there is artifact in the place)

Right after the game starts, type "GET something" in the starting place (ID 12).

```
GET something
```

```
No SOMETHING in this place, type GET or LOOK HERE to see what is in your current location.
```

Test result: correct as expected, report no-such-item-here message, and give suggestions.

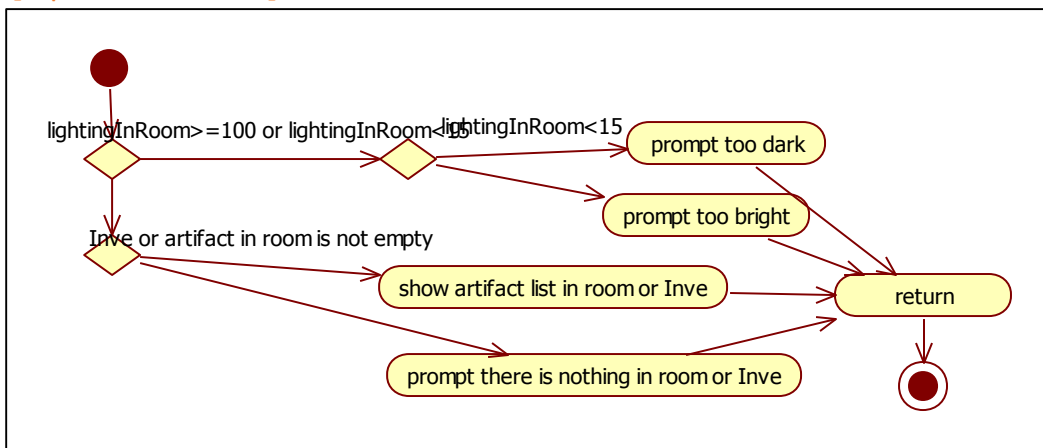
Conclusion on command GET

It's implemented correctly as designed.

Test command Use

The following test use white box test. The test cases chosen are according to branch testing.

Test "USE [option = null]" command



Activity diagram for use [option=null]

Using test map "MystiCity20_V3.10_s_Lighting.gdf" where LIGHTING section as follows:

	LIGHTING	3
1	13 100	// Ogre's Lair
2	101 14	// C1
3	106 0	// C6

1) Test Case: After the game starts, go N to Place ID 13, type "use".

```
use
It is too bright to see artifacts on the ground. Try turning a light off.
```

Test result: correct as expected, displaying corresponding message.

2) Test case: go to Room106, type "use".

```
use
It is too dark to see artifacts on the ground. Try turning a light on.
```

Test result: correct as expected, displaying corresponding message.

3) Test case: In entrance, type “use”

```
use
*No artifacts in your inventory.
*Artifacts in the room:
Leather bag
```

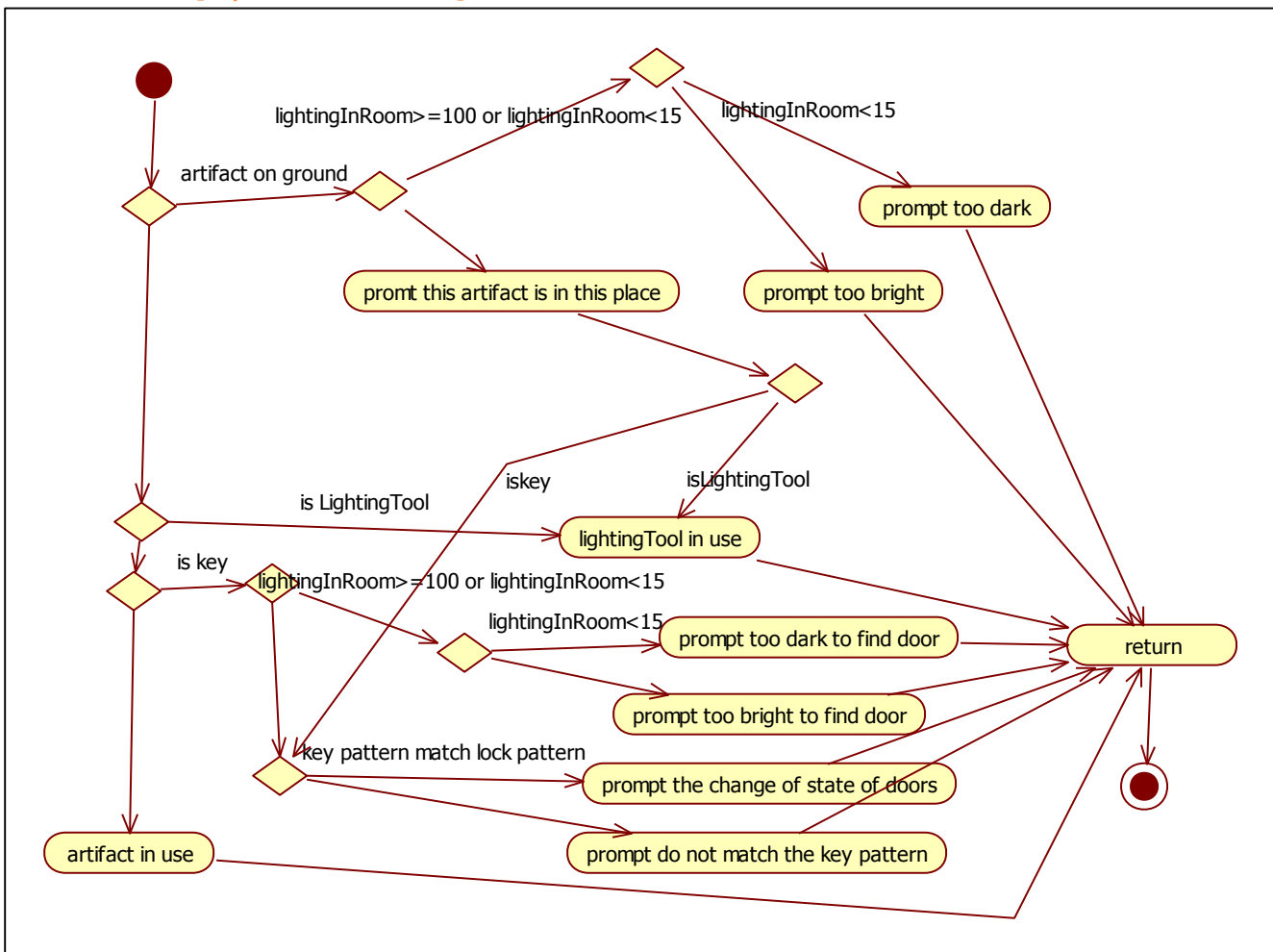
Test result: correct as expected, displaying corresponding message.

4) Test case: In Potions Lab, type “use”

```
use
*No artifacts in your inventory.
*No artifacts in the room.
```

Test result: correct as expected, displaying corresponding message.

Test “USE [option != null]” command



Activity diagram for use [option != null]

5) Test case: : After the game starts, “get Leather bag”, “go north”, “drop leather bag”, “use leather bag”

Using test map “MystiCity20_V3.10_s_Lighting.gdf”

```
use leather bag
It is too bright to use any artifacts on the ground. Try turning a light off.
```

Test result: correct as expected, displaying corresponding message.

6) Test Case: After the game starts, “get Leather bag”, “go north”, “drop leather bag”, “use leather bag”

Using test map “MystiCity20_V3.10_s_Lighting1.gdf” where LIGHTING section as follows:

```
LIGHTING      3

1      13      12      // Ogre's Lair
2    101 14  //C1
3      106      0      // C6
```

```
use leather bag
It is too dark to use any artifacts on the ground. Try turning a light on.
```

Test result: correct as expected, displaying corresponding message.

7) Test case: The user can use the key on the ground, so we drop the key on the ground in the Entrance Hall. type “use key” .

```
look here
*Artifacts in this room:
Leather bag
Brass key

use brass key
BRASS KEY is in this place.
  Path in the direction D to Potions Storeroom gets locked.

use brass key
BRASS KEY is in this place.
  Path in the direction D to Potions Storeroom gets unlocked.
```

Test result: As expected, the user can use the key on the ground as long as he could see the key in this place.

8) Test case: we also tested drop the golden key on the ground of portions lab, and use the golden key. It doesn't match the pathway from portions lab to other place.

```
use golden key
GOLDEN KEY is in your inventory.
No lock matched that key pattern.
```

Test result: as expected, it does not match the pathway from portions lab to other place.

9) Test case: With brass lantern on the ground of Entrance Hall, type “use brass lantern”

```
go w
*This is Entrance Hall.
You are standing in the entrance hall of the great six-room dungeon
There are doors to the east and north, and a stairway leading down
The main exit ( from the game ) is to the west

drop brass lantern
Dropped BRASS LANTERN on the ground, will stay in Entrance Hall.

look here
*Artifacts in this room:
Leather bag
Brass lantern

use brass lantern
BRASS LANTERN is in this place.
It is now on.
```

Test result: as expected, user can use the lighting tool on the ground, as long as the player can see it.

10) Test case: use light in the inventory.

This test case takes place in Entrance Hall, type “use brass lantern”

```
inve
*Artifacts in your inventory:
Brass lantern

use brass lantern
BRASS LANTERN is in your inventory.
It is now off.
```

Test result: as expected, user can use the lighting tool in the inventory.

11) Test case: with a key in the inventory, go to dark room--- Room 106, and type “use [key]”

```
inve
*Artifacts in your inventory:
Brass key

use brass key
BRASS KEY is in your inventory.
It is too dark to find a door for BRASS KEY. Try turning a light on.
```

Test result: if the room light level is below 15, user cannot use the key as expected.

12) Test case: type “use [key]” we tested the situation is the room light level that is bigger than 100(included), user cannot use key too.

13) Test case: with Brass Key in the inventory, because it has keyPattern 11, and mastercode 2, so expectedly it should open the doors that of [1100, 1199]. In the map file, the path from Entrance Hall to Potions Storeroom has lock pattern of 1101, so it should be toggled by this brass key.

```

look
*This is Entrance Hall.
You are standing in the entrance hall of the great six-room dungeon
There are doors to the east and north, and a stairway leading down
The main exit ( from the game ) is to the west
*Tips on surroundings:
You can go N, it's Ogre's Lair.
An Exit is to the W.
You can go E, it's Pool of Enchantment.
It's Potions Storeroom to the D but locked.
Other Directions are not available.

use brass key
BRASS KEY is in your inventory.
  Path in the direction D to Potions Storeroom gets unlocked.

use brass key
BRASS KEY is in your inventory.
  Path in the direction D to Potions Storeroom gets locked.

```

14) Test case: Golden key whose key pattern is 20, master code is 2, could toggle the doors whose lock pattern is with the range [2000,2099]. Expectedly, it could not toggle the door from Potions lab to Potions storeroom with lock pattern 1102.

```

look
*This is Potions Lab.
There is a cauldron of thick green goop here,
bubbling slowly over a cool blue flame.
Doors lead to the west and east.
*Tips on surroundings:
It's Potions Storeroom to the W but locked.
You can go E, it's Room 106.
Other Directions are not available.

use golden key
GOLDEN KEY is in your inventory.
No lock matched that key pattern.

```

15) Test case: Use other artifact which is not lighting tool or key in the place where light level is [15, 99]

After game start, get "leather bag" in the first place, and use it.

```

inve
*Artifacts in your inventory:
Leather bag

use leather bag
LEATHER BAG is in your inventory.
Now it's in use.

```

Test result: as expected, the artifact is in use.

16) Test case: use Artifact that doesn't exist in the inventory and not in current place

Right after the game starts, type "use something" in the starting place (ID 12).

```

look here
*Artifacts in this room:
Leather bag

inve
*No artifacts in your inventory.

use something
Sorry, no SOMETHING found in your inventory or in this place.

```

Test result: correct as expected, prompt there is not such artifact in the inventory or in current place.

Inspect Lighting restriction on “USE” command

By statically reviewing the code, we ensure the lighting restrictions on “USE” command: if lighting is not supporting, the program will not allow the player to overview the artifact list in current place by typing “USE”, and it will respond corresponding messages.

```

void executeUse(Environment env, Token token){
    if(token.optional == null){//no artifact name specified
        message = "Please choose one artifact in your inventory or at this place:\n";
        this.executeInventory();

        if(!checkLighting()){
            if((carryingLightValue + currentPlace.lightLevel + currentPlace.lightValueOnGround) < 15)
                message = "It is too dark to see artifacts on the ground. Try turning a light on.\n";
            else
                message = "It is too bright to see artifacts on the ground. Try turning a light off.\n";
            return;
        }
    }
}

```

Inspect “USE [key]” command in different situations

By statically reviewing the code, we ensure the lighting restrictions on “USE [key]” command: if lighting is not supporting, the program will not allow the player to use the key in current place and it will respond corresponding messages.

```

for(int i=0; i<this.inventory.size(); i++){
    if(token.optional.equalsIgnoreCase(this.inventory.get(i).name)){//in his inve
        message = token.optional + " is in your inventory.\n";
        //action of using key
        if(this.inventory.get(i) instanceof Key){
            if((carryingLightValue + currentPlace.lightLevel + currentPlace.lightValueOnGround) < 15){
                message += "It is too dark to find a door for " + token.optional + ". Try turning a light on.\n";
                return;
            }
            else if((carryingLightValue + currentPlace.lightLevel + currentPlace.lightValueOnGround) >= 100){
                message += "It is too bright to find a door for " + token.optional + ". Try turning a light off.\n";
                return;
            }
            this.useKey((Key) this.inventory.get(i), env);
            return;
        }
    }
}

```

By statically reviewing the code, we ensure that the user can also use the key in current place, which is not in his inventory. The command of “use [key]” could toggle the doors if they match the key pattern. If the lock Patterns of some doors match the key pattern, the program will prompt user of the information of the change of state of doors. If none of the doors match the key pattern, the program will prompt user so.

```

//execute USE Key at current Place
//unlock or lock all matching paths
void useKey(Key key, Environment env){
    boolean foundLock = false;
}

```

```

for(int j=0; j<this.currentPlace.outGoing.size(); j++){
    if(this.checkKeyAgainstLock(key.keyPattern,
        key.masterCode,
        this.currentPlace.outGoing.get(j).lockPattern)){
        foundLock = true;
        if(this.currentPlace.outGoing.get(j).isLocked == false){
            this.currentPlace.outGoing.get(j).isLocked = true;
            if(currentPlace.outGoing.get(j).outNeighborID==1)
                message += " Path in the direction "+currentPlace.outGoing.get(j).direction+" to "+
                    "EXIT" + " gets locked.\n";
            else
                message += " Path in the direction "+currentPlace.outGoing.get(j).direction+" to "+
                    "EXIT" + " gets locked.\n";
            Place.findPlaceByID(env.places,currentPlace.outGoing.get(j).outNeighborID).name
                + " gets locked.\n";
        }else{
            this.currentPlace.outGoing.get(j).isLocked = false;
            if(currentPlace.outGoing.get(j).outNeighborID==1)
                message += " Path in the direction "+currentPlace.outGoing.get(j).direction+" to "+
                    "EXIT" + " gets unlocked.\n";
            else
                message += " Path in the direction "+currentPlace.outGoing.get(j).direction+" to "+
                    "EXIT" + " gets unlocked.\n";
            Place.findPlaceByID(env.places,currentPlace.outGoing.get(j).outNeighborID).name
                + " gets unlocked.\n";
        }
        message += "The key unlocked every locked door and locked every unlocked
        door.\n";
    }
}
if(!foundLock)
    message += "No lock matched that key pattern.\n";
}

```

```

//use artifact in the place
for(int i = 0; i < currentPlace.roomArtifacts.size(); i++){
    if(token.optional.equalsIgnoreCase(currentPlace.roomArtifacts.get(i).name)){//in the room
        //allow the use of used artifacts even without lighting supporting
        //if(currentPlace.roomArtifacts.get(i).usedEver){
        //    this.useArtifactOnGround(currentPlace.roomArtifacts.get(i),env);
        //    return;
        //}
        //check lighting in the room before using unused artifacts

        message = token.optional + " is in this place.\n";
        if(currentPlace.roomArtifacts.get(i) instanceof Key)
            useKey((Key)currentPlace.roomArtifacts.get(i), env);
        else if(currentPlace.roomArtifacts.get(i) instanceof Light)
            useLight((Light)currentPlace.roomArtifacts.get(i), false);
        else
            ;//to be later defined with other artifacts

        //this.useArtifactOnGround(currentPlace.roomArtifacts.get(i),env);
        currentPlace.roomArtifacts.get(i).usedEver = true;//duplicate, realized in useArtifact
        return;
    }
}

```

Test “USE [light]” command in different situations

By statically reviewing the code, we ensure that the user can use the light in our inventory in any cases, but use the light on the ground requires light level [15, 99]. The command of “use [light]” could toggle the light, which could be either in the current place or at user’s inventory.

```

        //action of using light
        else if(this.inventory.get(i) instanceof Light){
            this.useLight((Light)this.inventory.get(i), true);
            return;
        }
        //action of using other artifacts
        message += "Now it's in use.";
        return;
    }
}

if((carryingLightValue
    + currentPlace.lightLevel + currentPlace.lightValueOnGround) < 15){
    message += "It is too dark to use any artifacts on the ground." +
        " Try turning a light on.\n";
    return;
}else if((carryingLightValue
    + currentPlace.lightLevel + currentPlace.lightValueOnGround) >= 100){
    message += "It is too bright to use any artifacts on the ground." +
        " Try turning a light off.\n";
    return;
}
}

```

```

//use artifact in the place
for(int i = 0; i < currentPlace.roomArtifacts.size(); i++){
    if(token.optional.equalsIgnoreCase(currentPlace.roomArtifacts.get(i).name)){//in the room
        //allow the use of used artifacts even without lighting supporting
        //if(currentPlace.roomArtifacts.get(i).usedEver){
        //    this.useArtifactOnGround(currentPlace.roomArtifacts.get(i),env);
        //    return;
        //}
        //check lighting in the room before using unused artifacts

        message = token.optional + " is in this place.\n";
        if(currentPlace.roomArtifacts.get(i) instanceof Key)
            useKey((Key)currentPlace.roomArtifacts.get(i), env);
        else if(currentPlace.roomArtifacts.get(i) instanceof Light)
            useLight((Light)currentPlace.roomArtifacts.get(i), false);
        else
            ;//to be later defined with other artifacts

        //this.useArtifactOnGround(currentPlace.roomArtifacts.get(i),env);
        currentPlace.roomArtifacts.get(i).usedEver = true;//duplicate, realized in useArtifact
        return;
    }
}
}

```

```

void useLight(Light light, boolean inInventory){
    if(light.lightActive){
        if(inInventory)
            carryingLightValue -= light.lightLevel;
        else
            currentPlace.lightValueOnGround -= light.lightLevel;
        light.lightActive = false;
        message += "It is now off.\n";
    }
    else{
        if(inInventory)
            carryingLightValue += light.lightLevel;
        else
            currentPlace.lightValueOnGround += light.lightLevel;
        light.lightActive = true;
        message += "It is now on.\n";
    }
}
}

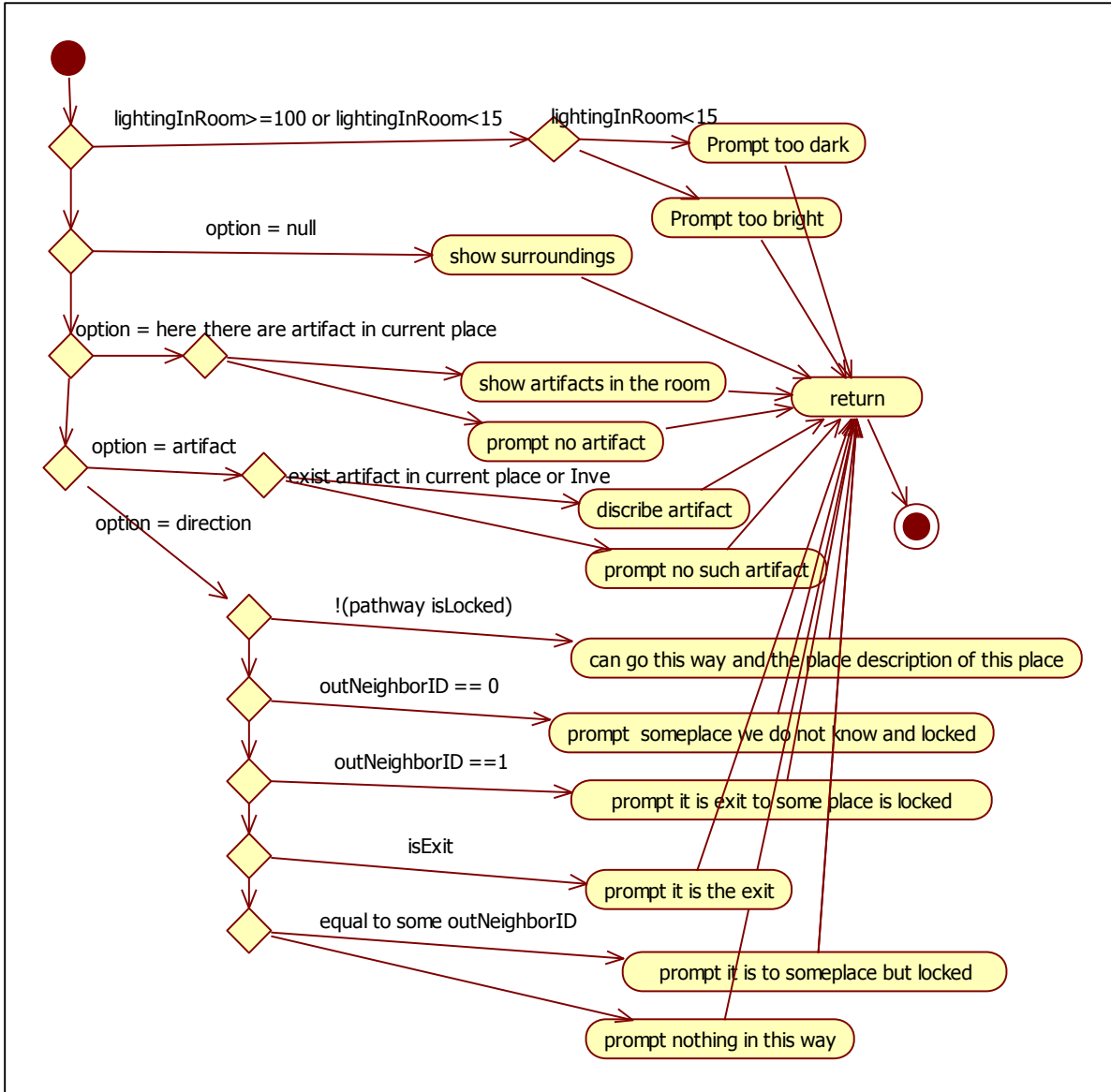
```


Conclusion on command USE

It's implemented correctly as designed.

Test Commands look

The following test use white box test. The tests chosen are according to branch testing.



Activity graph for execute look method

Note: case-insensitivity and white spaces and so on in user input are tested more systemically in testTokenExtraction.

Refer to the activity diagram in testExecuteCommands document to see the flow chart of executing the command LOOK.

Test Command look, look here, and look [direction], look [object]

- Input: look, look here, look [direction], look [object]

- Expected result:

Expected result can be divided into three cases depending on the light level of the place where the player input LOOK, LOOK [direction], LOOK [object] and LOOK HERE COMMAND:

Firstly, when the light level x satisfies the expression $15 \leq x \leq 99$, the program will return correct message letting the play know the following information:

1. If input LOOK COMMAND, the information would be:

- The name of this place.
- Brief description of this place.
- Tips on the surroundings, leading user to other places.

2. If input LOOK HERE COMMAND, the information would be:

2.1 The artifact name in this room.

2.2 There is no artifact in this room.

3. If input LOOK [direction], the information would be:

3.1 There is nothing in this direction, you cannot go this way.

3.2 You can go this direction and the name of place in this direction

3.3 The name of place in this direction, but it is locked.

4. If input LOOK [object], the information would be:

4.1 If the [object] is at current place, the program will give the description of the [object].

4.2 If the [object] is in user's inventory, the program will give the description of the [object].

4.3 If the [object] is neither in user's inventory nor in current place, it will prompt that there is no this [object] in this room or user's inventory.

Secondly, when the light level in current place is 100, input LOOK, LOOK [direction], LOOK [object] or LOOK HERE, the program will prompt "Too bright to see things here. Try adjusting the lights or going to your previous room."

Thirdly, when the light level in current place is below 15(excluded 15), input LOOK, LOOK [direction], LOOK [objects] or LOOK HERE, the program will prompt "Too dark to see things here. Try adjusting the lights or going to your previous room."

Design LIGHTING section of the map "MystiCity20_V3.10_s_Lighting.gdf" as follows:

```
LIGHTING      3
1    13  100 // Ogre's Lair
2    101 14  //C1
3    106 0   // C6
```

1) Test case: In Ogre's Lair

Type "look", "look here", "look e", "look leather bag"

Test result: as expected, the program returned message as bellow:

```
look
Too bright to see things here. Try adjusting the lights or going to your previous room.
```

2) Test case: In Room 101

Type "look", "look here", "look e", "look leather bag"

Test result: as expected, the program returned message as bellow:

```
look
Too dark to see things here. Try adjusting the lights or going to your previous room.
```

3) Test case: In Entrance Hall, type "look"

Test result: as expected, the program returned message as bellow:

```
look
*This is Entrance Hall.
You are standing in the entrance hall of the great six-room dungeon
There are doors to the east and north, and a stairway leading down
The main exit ( from the game ) is to the west
*Tips on surroundings:
You can go N, it's Ogre's Lair.
An Exit is to the W.
You can go E, it's Pool of Enchantment.
It's Potions Storeroom to the D but locked.
Other Directions are not available.
```

4) Test case: In Entrance Hall, type "look here"

Test result: as expected, the program returned message as bellow:

```
look here
*Artifacts in this room:
Leather bag
```

5) Test case: In entrance Hall, "get leather bag", type "look here"

Test result: as expected, the program returned message as bellow:

```
get leather bag
Picked LEATHER BAG up, it is now in your inventory.

look here
There is no artifact in this room.
```

6) Test case: get "Leather bag", go to Pool of Enchantment, type "look Leather bag"

Test result: as expected, the program returned message as bellow:

```
look Leather bag
This large leather bag looks like it would hold a lot.
```

7) Test case: In Entrance Hall, no "look brass lantern in Inventory", type "look Brass lantern"

Test result: as expected, the program returned message as bellow:

```
look brass lantern
No BRASS LANTERN in the room or your inventory. Try other artifacts.
```

8) Test case: In Entrance Hall, type "look e"

Test result: as expected, the program returned message as bellow:

```
look e
You can go E, it's Pool of Enchantment.
```

9) Test case: In Entrance Hall, type "look s"

Test result: as expected, the program returned message as bellow:

```
look s
Nothing is to the S. Try other directions.
```

10) Test case: In Entrance Hall, type “look d”

Test result: as expected, the program returned message as bellow:

```
look d
It's Potions Storeroom to the D but locked.
```

11) Test case: In Entrance Hall, type “look w”

Test result: as expected, the program prompt information as below:

```
look w
An Exit is to the W.
```

To test some loop in go, we need to change the destination ID in the map to test if this loop works as we expected.

As shown below, we change the destination ID of path 10 to 0, path 8 to -1, path 2 to 23

1	23	N	13	0	// TS to OL
2	13	E	23	1201	// OL to TS
3	13	S	12	0	// OL to EH
4	12	N	13	0	// EH to OL
5	22	N	-23	1202	// PE to TS, locked
6	23	S	-22	1301	// TS to PE, locked
7	12	W	1	0	// EH to Exit
8	12	E	-1	0	// EH to PE
9	22	W	12	0	// PE to EH
10	12	D	0	1101	// EH to PS, locked
11	11	U	-12	1103	// PS to EH, locked
12	22	D	21	0	// PE to PL

The test case below use the map “MystiCity20_V3.10_s_Go.gdf”

12) Test case: In Entrance Hall, type “look e”

Test result: as expected, the program prompt information as below:

```
look e
It's EXIT to the E but locked.
```

13) Test case: In Entrance Hall, type “look d”

Test result: as expected, the program prompt information as below:

```
look d
It's some WE-DON'T-KNOW-WHAT place to the D but locked.
```

- Conclusion: “LOOK”, “LOOK HERE”, “LOOK [direction]”, “LOOK [object]” command is executed correctly.

Reviewing command LOOK

(1) In the first part in the executeLook method, by inspection and debugging, it's shown that when invoke this method, it first check the light level of current place. By inspection and debugging, when light level is not between 15 and 99 (include 15 and

99), the program will prompt that it is too dark or too bright to see. Thus proving the correctness of LOOK, LOOK HERE, LOOK [direction], LOOK [object] when the light level is not within [15, 99].

```
/*execute the LOOK command, may or may not have direction/object*/
//valid input: LOOK [Direction,"HERE",object]
void executeLook(Environment env, Token token){
    // if lightLevel<15 or >= 100 user can not see
    if((carryingLightValue + currentPlace.lightValueOnGround + currentPlace.lightLevel) < 15 ||
        (carryingLightValue + currentPlace.lightValueOnGround + currentPlace.lightLevel) >= 100){
        message = "Too ";
        if((carryingLightValue + currentPlace.lightValueOnGround + currentPlace.lightLevel) < 15)
            message += "dark";
        else
            message += "bright";
        message += " to see things here. Try adjusting the lights or going to your previous room.\n";
        return;
    }
```

(2) By inspection and debugging, when light level is between 15 and 99, it's shown that the execution of command "look" outputs the information according to different situation as we expected.

```
String opt = token.optional;
if(opt == null){ //LOOK
    message = "This is "+this.currentPlace.name+".\n";
    for(int i=0;i<this.currentPlace.description.size();i++){
        message += this.currentPlace.description.get(i)+"\n";
    }
    message += "*Tips on surroundings: \n";
    //describe all the surroundings
    for(int i=0; i<currentPlace.outGoing.size();i++){
        PlaceOut tempPO = currentPlace.outGoing.get(i);
        String tempDir = Token.abbreviateDir(tempPO.direction);
        if(tempPO.isLocked){
            if(tempPO.outNeighborID == 0){
                message += "It's some WE-DON'T-KNOW-WHAT place "
                    + "to the " +tempDir+" but locked.\n";
            }else if(tempPO.outNeighborID == 1){
                message += "It's EXIT to the "+tempDir+" but locked.\n";
            }else{
                message += "It's "+Place.findPlaceByID(env.places, tempPO.outNeighborID).name
                    + " to the "+tempDir+ " but locked.\n";
            }
        }else if(tempPO.isExit){
            message += "An Exit is to the "+tempDir+".\n";
        }else{
            message += "You can go "+tempDir+", "+
                " it's "+Place.findPlaceByID(env.places, tempPO.outNeighborID).name+".\n";
        }
    }
    message += "Other Directions are not available.\n";
    return;
}
```

(3) By inspection and debugging, when light level is between 15 and 99, it's shown that the execution of command "look [direction]" outputs the information according to different situation as we expected.

```

}else if(Token.isValidDirection(opt)){ //LOOK Direction
    opt = Token.abbreviateDir(opt); //upper case for output
    PlaceOut tempPO = currentPlace.findPlaceOutByDirection(opt);
    if(tempPO != null){
        //describe things in that direction
        if(tempPO.isLocked){
            if(tempPO.outNeighborID == 0){
                message += "It's some WE-DON'T-KNOW-WHAT place "
                    + "to the " + opt + " but locked.\n";
            }else if(tempPO.outNeighborID == 1){
                message += "It's EXIT to the " + opt + " but locked.\n";
            }else{
                message += "It's " + Place.findPlaceByID(env.places, tempPO.outNeighborID).name
                    + " to the " + opt + " but locked.\n";
            }
        }else if(tempPO.isExit){
            message += "An Exit is to the " + opt + ".\n";
        }else{
            message += "You can go " + opt + ", "
                + "it's " + Place.findPlaceByID(env.places, tempPO.outNeighborID).name + ".\n";
        }
    }else{
        message = "Nothing is to the " + opt + ". Try other directions.\n";
    }
    return;
}

```

(4) By inspection and debugging, when light level is between 15 and 99, it's shown that the execution of command "look here" outputs the information of current place and brief description of the place. Thus prove the correctness of "look here" command.

```

//action of LOOK HERE command
if(opt.equalsIgnoreCase("HERE")){
    if(this.currentPlace.roomArtifacts.isEmpty()){
        message = "There is no artifact in this room.\n";
    }else{
        message = "*Artifacts in this room:\n";
        for(int i=0;i<this.currentPlace.roomArtifacts.size();i++){
            message += this.currentPlace.roomArtifacts.get(i).name+"\n";
        }
    }
    return;
}

```

(5) By inspection and debugging, when light level is between 15 and 99, it's shown that the execution of command "look [object]" outputs the description of artifact if the artifact is in current place or player's inventory and outputs there is not such artifact in current place and the player's inventory if this is the case. Thus, prove the correctness of "look [object]" command.

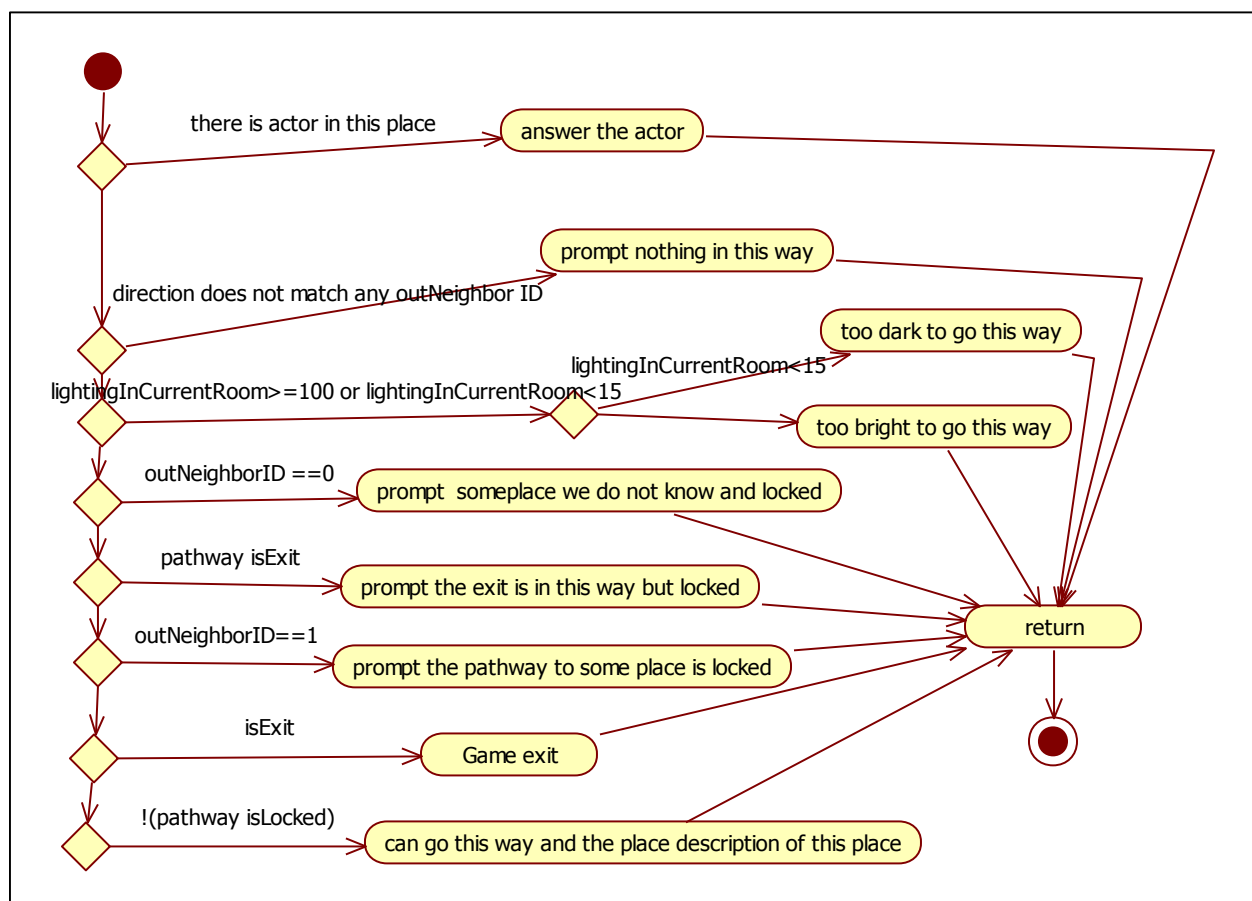
```

//action of LOOK[Object Name]command
for(int i=0;i<this.currentPlace.roomArtifacts.size();i++){//look at item in the room
    if(opt.equalsIgnoreCase(this.currentPlace.roomArtifacts.get(i).name)){
        for(int j=0; j<this.currentPlace.roomArtifacts.get(i).description.size();j++){
            message = currentPlace.roomArtifacts.get(i).description.get(j)+"\n";
        }
        return;
    }
}
for(int i=0;i<this.inventory.size();i++){//look at item in the inve
    if(opt.equalsIgnoreCase(this.inventory.get(i).name)){
        for(int j=0; j<this.inventory.get(i).description.size();j++){
            message = inventory.get(i).description.get(j)+"\n";
        }
        return;
    }
}
//no such item
message = "No "+opt+" in the room or your inventory. Try other artifacts.\n";

```

Test command Go

By reviewing the code, we ensure that in different situation, after type “go [direction]” command, the program could give the player information about whether the player could go through this direction. And if the light level is not in the range [15, 99], the program will prompt the player about either too dark or too bright in that place after the player go to specific direction. If the room is too dark or too bright, the player can only go back to the previous room, even if other direction is not locked.



Activity diagram for GO [direction] command

The following test use white box test. The tests chosen are according to branch testing.

1) Test case: In the Entrance Hall, type “go S”, which is not suggested by the system

Using map “MystiCity20_V3.10_simplified.gdf”

```
look
*This is Entrance Hall.
You are standing in the entrance hall of the great six-room dungeon
There are doors to the east and north, and a stairway leading down
The main exit ( from the game ) is to the west
*Tips on surroundings:
You can go N, it's Ogre's Lair.
An Exit is to the W.
You can go E, it's Pool of Enchantment.
It's Potions Storeroom to the D but locked.
Other Directions are not available.

go s
Cannot go Direction S, nothing there. Try other directions.
```

Test result: as expected, the program prompts the player that there is nothing there.

To test some loop in go, we need to change the destination ID in the map to test if this loop works as we expected.

As shown below, we change the destination ID of path 10 to 0, path 8 to -1, path 2 to 23

1	23	N	13	0	// TS to OL
2	13	E	23	1201	// OL to TS
3	13	S	12	0	// OL to EH
4	12	N	13	0	// EH to OL
5	22	N	-23	1202	// PE to TS, locked
6	23	S	-22	1301	// TS to PE, locked
7	12	W	1	0	// EH to Exit
8	12	E	-1	0	// EH to PE
9	22	W	12	0	// PE to EH
10	12	D	0	1101	// EH to PS, locked
11	11	U	-12	1103	// PS to EH, locked
12	22	D	21	0	// PE to PL

The test case below use the map “MystiCity20_V3.10_s_Go.gdf”

2) Test case: type “go n” to the Ogre’s Lair. As is shown in the modified map, the pathway from Orge’s Lair to Treasure Room is unlocked.

```
go n
*Actor Ogre is here. Type "ANSWER yourAnswer" to solve his riddle before moving on:
What always ends everything?

answer g
Correct! You can move on now.

look
Too bright to see things here. Try adjusting the lights or going to your previous room.

go e
```


It is too bright here. Try adjusting the lights or returning to your previous room.

go s

*This is Entrance Hall.

You are standing in the entrance hall of the great six-room dungeon

There are doors to the east and north, and a stairway leading down

The main exit (from the game) is to the west

Test result: as expected, when the light level of current place is bigger than range [15, 99], the player can only go back, cannot go other place even if the pathway is unlocked.

3) Similarly, when we change the light level below 15, the player can only go back, cannot go other place even if the pathway is unlocked.

4) Test case: type “go d” in the Entrance Hall

look

*This is Entrance Hall.

You are standing in the entrance hall of the great six-room dungeon

There are doors to the east and north, and a stairway leading down

The main exit (from the game) is to the west

*Tips on surroundings:

You can go N, it's Ogre's Lair.

An Exit is to the W.

It's EXIT to the E but locked.

It's some WE-DON'T-KNOW-WHAT place to the D but locked.

Other Directions are not available.

go d

Sorry, it's some WE-DON'T-KNOW-WHAT place to the D but locked.

Test Result: according to the modified map, the pathway from Entrance Hall to down leads to the place we do not know. The test result shows the exact result as we expected.

5) Test case: type “go e” in the Entrance Hall

look

*This is Entrance Hall.

You are standing in the entrance hall of the great six-room dungeon

There are doors to the east and north, and a stairway leading down

The main exit (from the game) is to the west

*Tips on surroundings:

It's some WE-DON'T-KNOW-WHAT place to the N but locked.

An Exit is to the W.

It's EXIT to the E but locked.

It's Potions Storeroom to the D but locked.

Other Directions are not available.

go e

Sorry, it's EXIT to the E but locked.

Test result: according to the modified map, the pathway from Entrance Hall to Pool of Enchantment is EXIT but locked. The test result shows the exact result as we expected.

6) Test case: In the Entrance Hall, type “go d”

look

*This is Entrance Hall.

```
You are standing in the entrance hall of the great six-room dungeon
There are doors to the east and north, and a stairway leading down
The main exit ( from the game ) is to the west
*Tips on surroundings:
You can go N, it's Ogre's Lair.
An Exit is to the W.
You can go E, it's Pool of Enchantment.
It's Potions Storeroom to the D but locked.
Other Directions are not available.
```

```
go d
```

```
Sorry, it's Potions Storeroom to the D but locked.
```

Test result: as expected, if the pathway is locked, the program will prompt the player so.

7) Test case: type “go w” in the Entrance Hall

```
look
*This is Entrance Hall.
You are standing in the entrance hall of the great six-room dungeon
There are doors to the east and north, and a stairway leading down
The main exit ( from the game ) is to the west
*Tips on surroundings:
It's some WE-DON'T-KNOW-WHAT place to the N but locked.
An Exit is to the W.
It's EXIT to the E but locked.
It's Potions Storeroom to the D but locked.
Other Directions are not available.

go w
Congratualations! Game exited. Thanks for coming.
```

Test result: according to the modified map, the west direction of Entrance Hall is the exit pathway. The test result as is shown below gives the exact result as we expected.

8) Test case: In the Entrance Hall, type “go e”, as suggested in the system

Using map “MystiCity20_V3.10_simplified.gdf”

```
look
*This is Entrance Hall.
You are standing in the entrance hall of the great six-room dungeon
There are doors to the east and north, and a stairway leading down
The main exit ( from the game ) is to the west
*Tips on surroundings:
You can go N, it's Ogre's Lair.
An Exit is to the W.
You can go E, it's Pool of Enchantment.
It's Potions Storeroom to the D but locked.
Other Directions are not available.

go e
*This is Pool of Enchantment.
You are in a round room with a clear enchanting pool of water.
There are doors to the north and west.
There is a slide leading downwards to the floor below.
You can go down safely, but you might not be able to get back up.
```

Test result: as expected, when types “go [direction]” that the direction is suggested by the system which means the player can go through, then the player will be in that place with the description of that place by the system.

Conclusion on command GO

It's implemented correctly as designed.

Test command exit

Statically review the code, the first "if condition" is reserved for future use because PlaceID is not allow to be 1 when we check the validation of the map. So if the user types "exit" at any place, the program will prompt the message that "sorry, it's not an exit here."

```
/*execute the Exit command*/  
void executeExit(Environment env, Token token) {  
    if(currentPlace.ID == 1){ //exit here  
        message = "Congratualations! Game exited. Thanks for coming.";  
        this.Exit = true;  
        this.currentPlace = null;  
    }else{  
        message = "Sorry, it's not an exit here.\n";  
        message = "Game quit. Thanks for coming.";  
        this.Quit = true;  
        this.currentPlace = null;  
    }  
}
```

Test case: type "exit" in the Entrance Hall.

```
exit  
Sorry, it's not an exit here.
```

Test result: as expected, the program prompts "sorry, it's not an exit here."

Conclusion on command EXIT

It's implemented correctly as designed.