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. | //input: tab |
| e xit  ERROR: Invalid Token. | //input: white space |
| //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.

|  |  |
| --- | --- |
| 1st line//test input  2nd 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 | //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  //input:” ”+”look”+”\t”    //input: ”\t”+”LOOK”+” ”+ “SOUTHSOUTHWEST”  // input: “GET”+ a single white space |
| go  ERROR: Invalid Token.  go e  Command: GO, Optional: E  go west  Command: GO, Optional: W  go articfact  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],

"INVE(NTORY)", "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,WESTSOUTHWEST.

* 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:



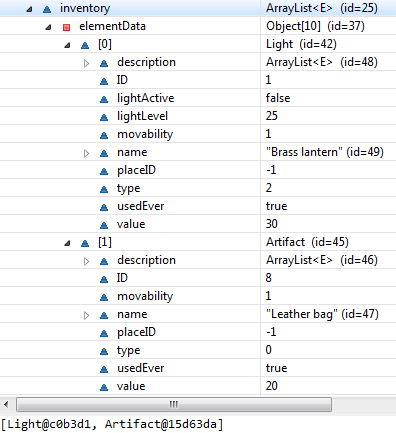
System output (the first line is input test command):



1. 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:



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



## Conclusion

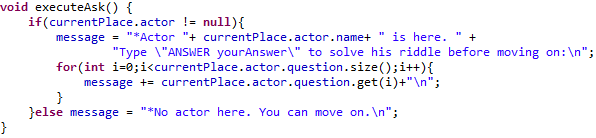
The executeInventory() function is functioning well, presenting the real-time content in the inventory.

# Test Command Ask and Answer

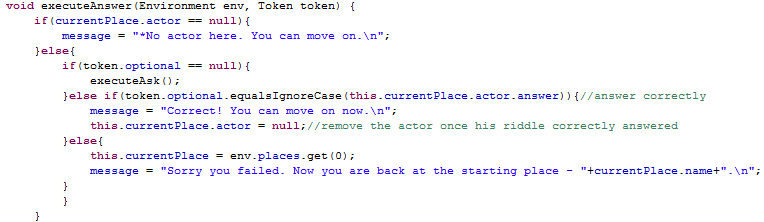
By code review, we ensure Ask and Answer commands are only allowed in game version 3.10, and this is realized in the executeToken() function in the game class.

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.

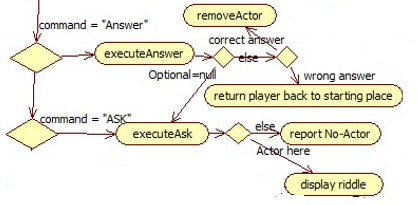
The ASK command comes with no parameter, and its function is to ask for description of the riddle in the current place, if there is an actor here holding a riddle.



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.



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



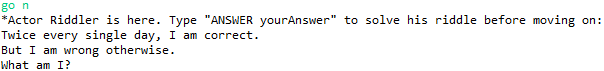
By code review, the functions look good to go. Then we do testing as shown next.

* Test environment: MystiCity20\_V3.10\_simplified.gdf

There are 2 actors[[1]](#footnote-1) in this map:

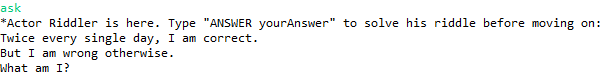
|  |  |
| --- | --- |
| 16 106 Ogre g  *\*What always ends everything?* | 5000 13 Riddler wrong clock  *\*Twice every single day, I am correct.*  *\*But I am wrong otherwise.*  *\*What am I?* |

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)



* Test cases and results:

1. ask



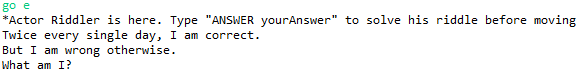
Test result: correct, as expected, displaying the riddle (actor’s name included).

1. Go to previous place without solving the riddle



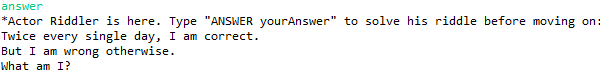
Test result: correct as expected. Successfully come back to previous place.

1. Go to a new place without solving the riddle



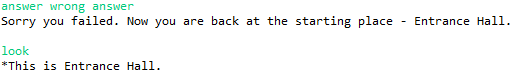
Test result: correct as expected. Going to another place than the previous one is not allowed without solving the riddle.

1. answer



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

1. wrong answer 1



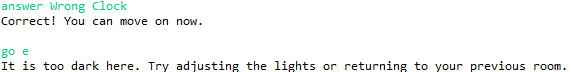
Test result: correct, as expected, transported to the starting place.

1. Wrong answer 2



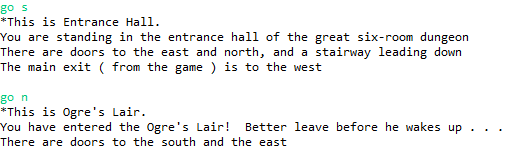
Test result: correct as expected, but showing the vulnerability of current design and implementation – only exact same string can be accepted (case insensitive but white spaces in the middle are not allowed).

1. Correct answer 1



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…



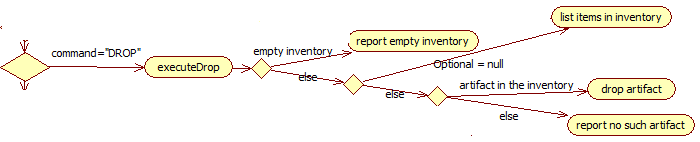
Test result: correct as expected. After solving the riddle, the actor disappears from the place.

## 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 he 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:

1. DROP (empty inventory)

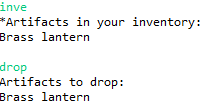
Right after the game starts, the inventory is empty. Type “drop”.



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

1. 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”.



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

1. DROP Artifact\_NOT\_in\_the\_inventory (non-empty inventory)

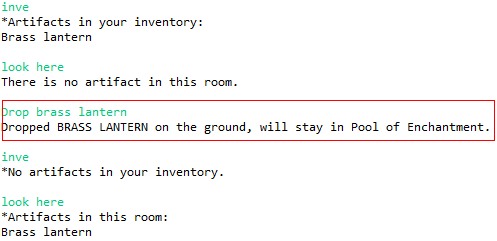
Following prior test 2), type “DROP something not existing”.



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

1. 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.

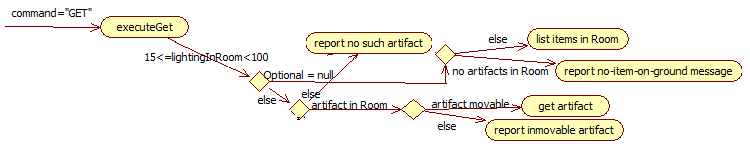


Test result: correct as expected, the specified artifact is removed from the inventory and left in the place.

## Conclusion

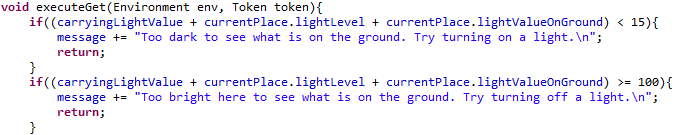
The DROP is designed and implemented properly.

# 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.



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”.



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.

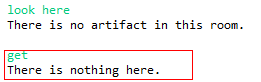
## Test GET in different situations

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)

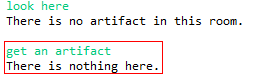
After the game starts, go E then go Down to the Potions Lab. Type “get”.



Test result: correct as expected, displaying corresponding message.

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

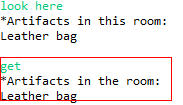
After the game starts, go E then go Down to the Potions Lab. Type “get an artifact”.



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

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

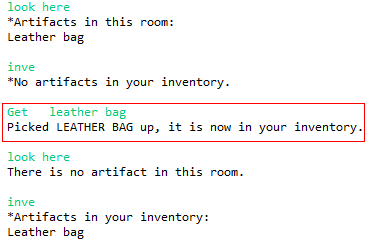
Right after the game starts, type “get” in the starting place (ID 12).



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

1. 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.



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

1. Test case: get Artifact that doesn’t exists in the place (there is artifact in the place)

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



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

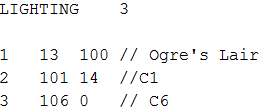
The following test use white box test. The test cases chosen are according to branch testing.

## 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:



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



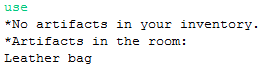
Test result: correct as expected, displaying corresponding message.

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



Test result: correct as expected, displaying corresponding message.

3) Test case: In entrance, type “use”



Test result: correct as expected, displaying corresponding message.

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



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”

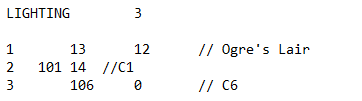
Using test map”MystiCity20\_V3.10\_s\_Lighting.gdf”



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”

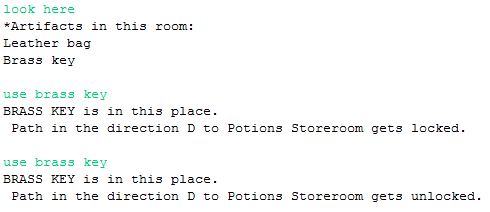
Using test map”MystiCity20\_V3.10\_s\_Lighting1.gdf” where LIGHTING section as follows:





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” .



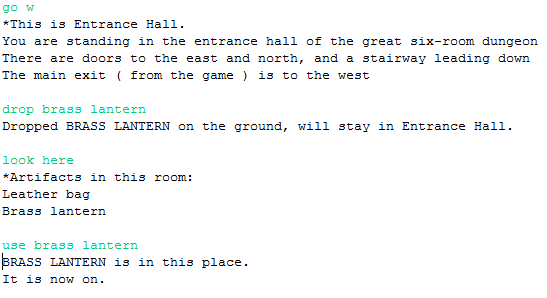
Test result: As expected, the user can use the key on the ground as long as he could see the key in this place.

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.



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

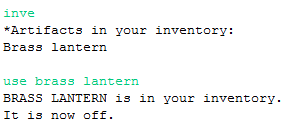
9) Test case: With brass lantern on the ground of Entrance Hall, type”use brass lantern”



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

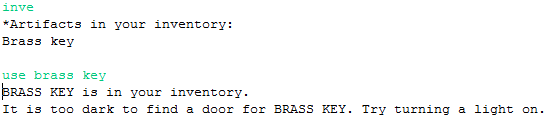
10) Test case: use light in the inventory.

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



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

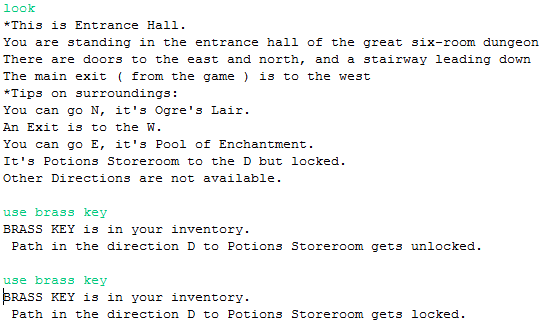
11) Test case: with a key in the inventory, go to dark room--- Room 106, and type “use [key]”



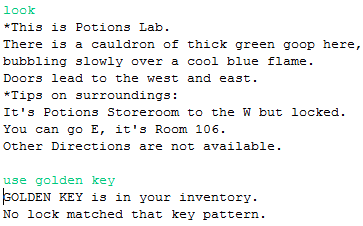
Test result: if the room light level is below 15, user cannot use the key as expected.

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.

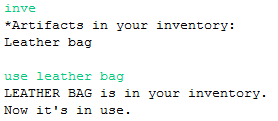


14) Test case: Golen key whose key pattern is 20, master code is 2, could toggle the doors whose lock pattern is with the range [2000,2099]. Expectedly, it could not toggle the door from Potions lab to Potions storeroom with lock pattern 1102.



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

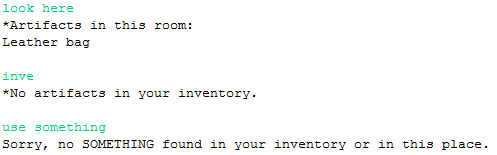
After game start, get “leather bag” in the first place, and use it.



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

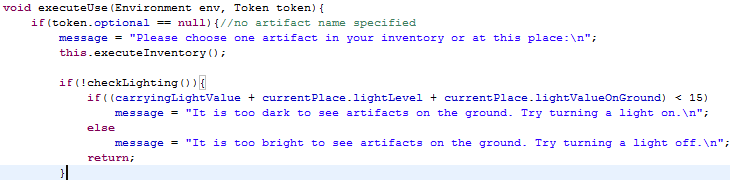
Right after the game starts, type “use something” in the starting place (ID 12).



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

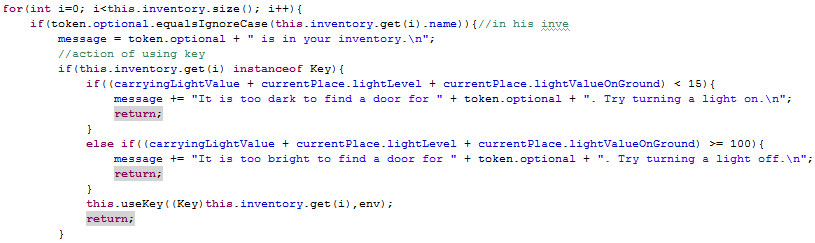
## Inspect Lighting restriction on “USE” command

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.



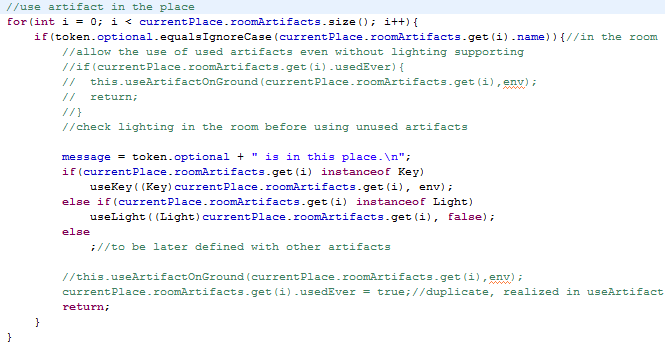
## Inspect “USE [key]” command in different situations

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.



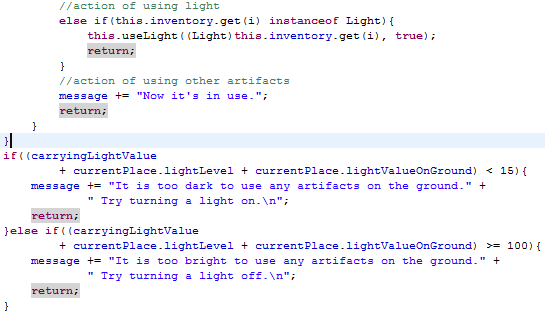
By statically reviewing the code, we ensure that the user can also use the key in current place, which is not in his inventory. The command of “use [key]” could toggle the doors if they match the key pattern. If the lock Patterns of some doors match the key pattern, the program will prompt user of the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

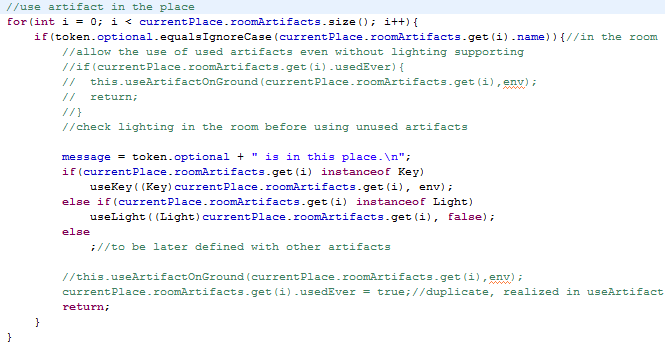
|  |
| --- |
| //execute USE Key at current Place  //unlock or lock all matching paths  **void** useKey(Key key,Environment env){  **boolean** foundLock = **false**;  **for**(**int** j=0; j<**this**.currentPlace.outGoing.size(); j++){  **if**(**this**.checkKeyAgainstLock(key.keyPattern,  key.masterCode,  **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 "+  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 "+  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";  } |

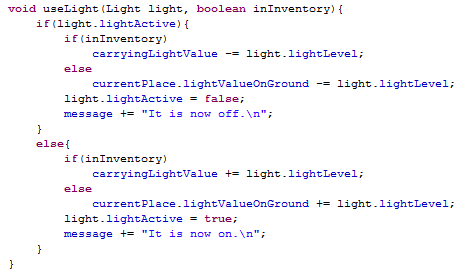


## Test “USE [light]” command in different situations

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.







## 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<=x<=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].

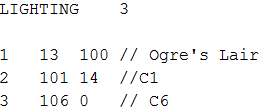
4.2 If the [object] is in user’s inventory, the program will give the description of the [object].

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:



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:



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:



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

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

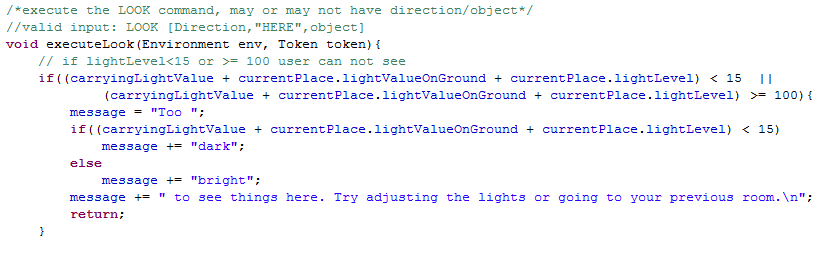
|  |  |
| --- | --- |
|  |  |
|  | 4) Test case: In Entrance Hall, type “look here”  Test result: as expected, the program returned message as bellow:    5) Test case: In entrance Hall, “get leather bag”, type “look here”  Test result: as expected, the program returned message as bellow:    6) Test case: get “Leather bag”, go to Pool of Enchantment, type “look Leather bag”  Test result: as expected, the program returned message as bellow:    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:    8) Test case: In Entrance Hall, type “look e”  Test result: as expected, the program returned message as bellow:    9) Test case: In Entrance Hall, type “look s”  Test result: as expected, the program returned message as bellow:    10) Test case: In Entrance Hall, type “look d”  Test result: as expected, the program returned message as bellow:    11) Test case: In Entrance Hall, type “look w”  Test result: as expected, the program prompt information as below:    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    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:    13) Test case: In Entrance Hall, type “look d”  Test result: as expected, the program prompt information as below: |



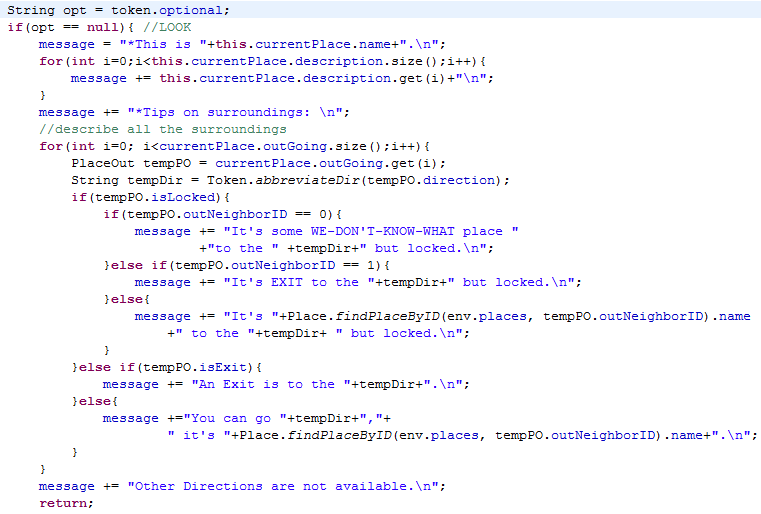
Conclusion: “LOOK”, “LOOK HERE”, “LOOK [direction]”, “LOOK [object]”command is executed correctly.

## Reviewing command LOOK

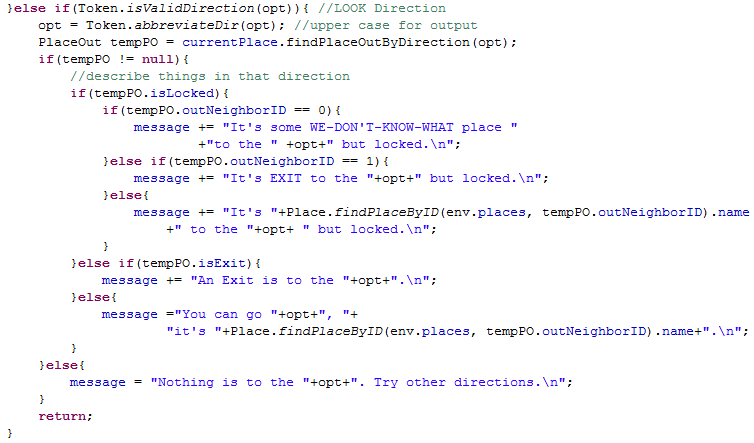
(1) In the first part in the executeLook method, by inspection and debugging, it’s shown that when invoke this method, it first check the light level of current place. By inspection and debugging, when light level is not between 15 and 99 (include 15 and 99), the program will prompt that it is too dark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



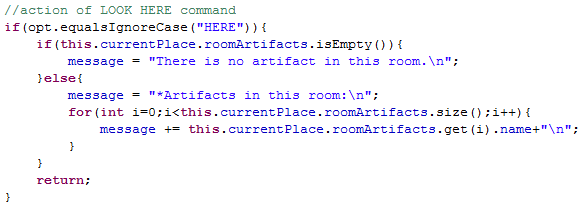
(2) By inspection and debugging, when light level is between 15 and 99, it’s shown that the execution of command “look” outputs the information according to different situation as we expected.



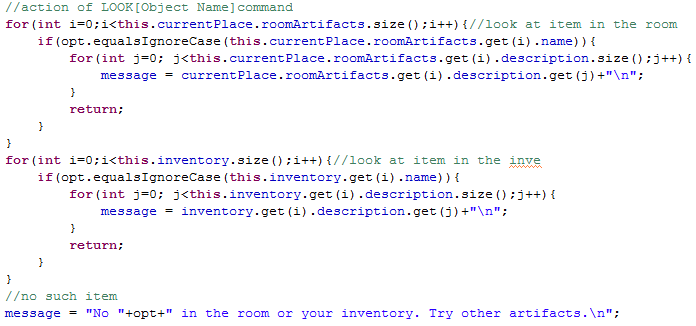
(3) By inspection and debugging, when light level is between 15 and 99, it’s shown that the execution of command “look [direction]” outputs the information according to different situation as we expected.



(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.



(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.



# 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

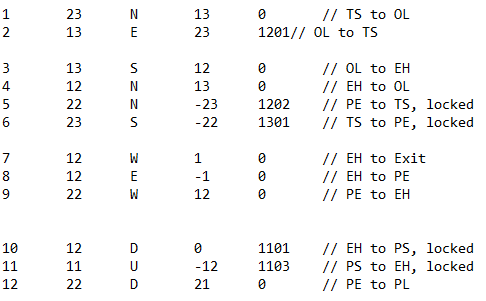
Using map “MystiCity20\_V3.10\_simplified.gdf”

|  |
| --- |
| 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



The test case below use the map “MystiCity20\_V3.10\_s\_Go.gdf”

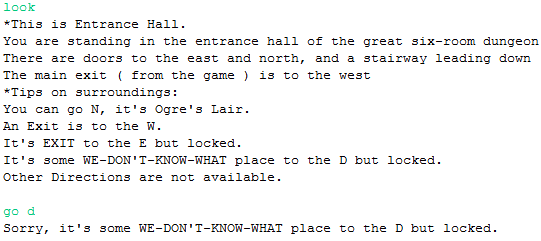
2) Test case: type “go n” to the Ogre’s Lair. As is shown in the modified map, the pathway from Orge’s Lair to Treasure Room is unlocked.

|  |
| --- |
| 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.

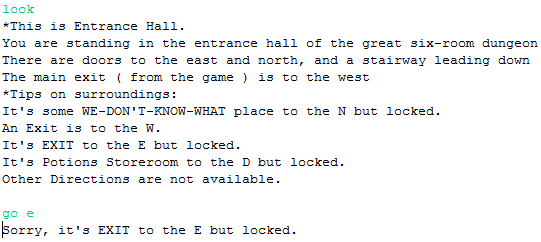
3) Similarly, when we change the light level below 15, the player can only go back, cannot go other place even if the pathway is unlocked.

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



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



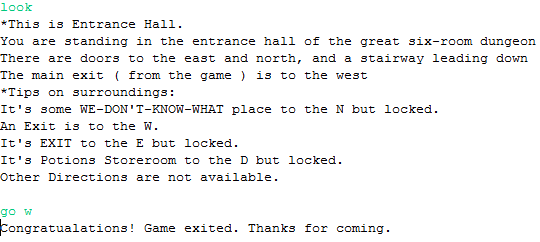
Test result: according to the modified map, the pathway from Entrance Hall to Pool of Enchantment is EXIT but locked. The test result shows the exact result as we expected.

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



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. |

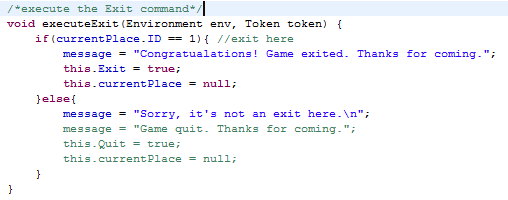
Test result: as expected, when types “go [direction]” that the direction is suggested by the system which means the player can go through, then the player will be in that place with the description of that place by the system.

## 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.”



Test case: type “exit” in the Entrance Hall.



Test result: as expected, the program prompts “sorry, it’s not an exit here.”

## Conclusion on command EXIT

It’s implemented correctly as designed.

1. Actor format:

   ID placeID name long\_answer\_with\_spaces

   \*Riddle description [↑](#footnote-ref-1)