Text Game V3.10 Release Plan

Group 10 – Yue, Dan, Nianzu Ma

For the final release of our Text Game V3.10, we’ll focus most on debugging and perfecting existing functions, only minor new functions introduced. One major innovation will be NPCs and riddles, as stated in section 1.2.

To improve the reliability of our program, we’ll take techniques to avoid, detect as well as tolerate certain faults. Certain faults are allowed in the map files and user inputs, stated in section 2; also we’ll categorize intolerable faults – treated as throwable errors – in the same section. To avoid certain faults we decide on some global map designing rules and coding rules in section 3, and we’ll inspect the original map file and source code according to these rules. Besides, testing and debugging techniques will also be used before the final release, to identify and remove faults if they exist.

# GDF v3.10 design and Implementation

In our GDF file v3.10, some modifications will be made to GDF file v2.0, following the rules below to ensure down-compatibility:

* For existing sections[[1]](#footnote-1), we will only add new parameters after all existing parameters, or change the meaning (thus usage) of existing parameters without changing its type and value domain.
* New sections can be added freely as necessary.

Note section orders are irrelevant in the GDF file. See details and explanation in section 2.1.

## Improvements

There are several improvement we want to make on our program version 2.0.

### Version control in map parsing

Our GDF file V3.10 will be down-compatible. Although only GDF file with version 3.10 GDF file is designed for our game edition, GDF file with version 2.0 or 1.0 will also be accepted, leaving missing parameters as initialized value while parsing the map or initializing the game.

### Unify the executions of different commands w.r.t. Lighting

Especially for the USE/GET Artifacts command, and differentiating Artifacts on the ground and in the player’s inventory.

## Innovations

### ~~Bi-directional path~~

~~Paths will be bi-directional. As long as there is a path in the map, its “twin” path will also be added. The twin paths Do Not share lock status, i.e. changing the lock status of one path will not change its twin path’s lock status.~~

~~For a place, there will only be~~ **~~one~~** ~~outgoing to each direction. Thus in the example above, if a path is specified in the map file, it will overwrite the generated twin path of ’s.~~

~~To implement this design, it does not really matter how the map file is written, but only the map parsing. It automatically adds twin paths after checking consistency. See section 2.1 for more details.~~

~~To help realize this design, we define~~ **~~direction pairs~~** ~~to cover the opposite direction pairs, such as N and S, SE and NW.~~

### USE Key to specific Direction

Add parameters in the USE command.

### NPC and Riddles

We introduce a simple NPC in our game v3.10, which is unmovable in a place. Once the player enters a new place and there is a NPC inside, the NPC will ask the player a riddle/question, and the player will have to answer it correctly before GOing to other places or EXITing the game – once the riddled is figured out, the NPC disappears.

The NPC section in the GDF file may be constructed in the following manner:

NPC nNPCs //NPC flag, followed by number of NPCs to parse

//one sample NPC presented below

NPCID name roomID riddleAnswer //NPCID is unique int, roomID must be valid

\*riddleQuestion //name and riddleAnswer must not be null, and are strings

\*riddleQuestion //a line containing an \* at the beginning is part of the question

\*riddleQuestion //riddleQuestion must be printable

The NPC section in the GDF file v3.10 shall be parsed in the following manner: Only one NPC is allowed in one place, thus a new NPC being parsed would overwrite the NPC in an existing room.

The user now has two commands to use. The user will enter a room, and the description (or using the LOOK command) should notify the user that an NPC is in the room. Command ASK will print the riddleQuestion to the screen. ANSWER by itself will print the question to the screen again. ANSWER userAnswer will be used for a string comparison to the NPC’s riddleAnswer. If they match, the riddle is solved; else the user is teleported to the entrance of the environment.

Some other commands are also limited if there is an NPC in the room and the riddle has not yet been solved. EXIT command will not work unless riddle solved; GO command will only work if their previous room is in that direction, meaning they can only return to their previous room. We allow the use of commands such as LOOK, USE, and GET, DROP, and INVE, keeping the possibilities in future game expansion we will add other ways to deal with the NPC.

# Fault Tolerance

Faults mainly exist in parsing the map file, and explaining and executing user input. In this section we’ll define tolerable faults, ignoring which will not affect the game playing, and intolerable faults, which will be thrown out as errors in a message to the player.

## Fault Tolerance

Some basic fault tolerance mechanisms (applied anywhere in the program):

* White space is always tolerated inside map file or within user input. Note “\t” is replaced with space while extracting Tokens from user input.
* Only headline and section PLACES are required in the map file, other sections are optional. Which is to say, as long as there are places in the map, the game is allowed to begin.
* Order of sections are irrelevant in the map file. But of course, all sections depend on the PLACES section, so PLACES better comes first when designing a map file.

We try to tolerate as many faults as possible, as long as the game is playable.

* While checking duplicate ID when parsing the map, Keys and Lights are only checked within their own types. Keys and Lights are all stored in the Artifacts list in the game class in the runtime, with type 1 and 2 respectively, and normal Artifacts are of type 0.

|  |  |
| --- | --- |
| **Tolerable Faults in map parsing** | **Solution** |
| The number of staff stated in each section is inconsistent with real number.  e.g. nPlaces != places.size(), nArtifacts != artifacts.size(), … | Ignored. |
| Incomplete map with OPTIONAL sections missing |
| Invalid lightLevel value in LIGHTING, i.e. lightLevel beyond 0~100 | That line ignored. |
| Invalid Place ID in LIGHTING or NPC. |
| Another NPC in a place where there’s already a NPC. |

## Intolerant Fault

There are certain faults tolerating which will very probably lead to errors and failures in the game in the runtime, so we avoid them, and throw out ERROR message to the player when coming across.

|  |  |
| --- | --- |
| **Intolerant Faults** | **Solution** |
| Invalid map file with no proper title line | Thrown out as errors, report it to the player, and ask for a correct map file. |
| Incomplete map without PLACES |
| Incomplete map without the “GDF…” line. i.e. no magic word, version number and environment name |
| Place or Artifact (including Key and Light) has no name |
| Duplicate ID within any section (excludes PATHS and LIGHTING where not tested) |
| Invalid source Place ID, destination Place ID, or direction in Path |
| Invalid location Place ID in Artifact |
|  |  |

# Coding Rules and Inspection

Coding Rule 1: every variable has to be initialized in the source code.

Coding Rule 2: precondition shall be checked within each function.

# Testing

Test cases will be used to test all functional requirements as well as the fault-tolerant designs stated above. Here in this report we briefly introduce the main functions for each part of the game to test. All designs in Section 2 and 3 will be tested or reviewed.

## Test Map parsing

All possible faults regarding file operations and contents in Section 2 will be tested.

Parsing the numbers in the map file: Negative integer numbers are allowed to be used as staff ID, as long as it’s unique.

## Test Game playing

1. **Test Lighting**

By default, the room light level is 50. //If it changes with time, it might be unplayable at night.

Items are visible only when the light level in the room (including the usage of Lights) is between 15 and 100. // for executing LOOK, GET, GO (going to previous place is allowed), USE (using artifacts in his inventory is allowed)

1. **Test GO**

Possible results: Go to some place; cannot go somewhere because it’s locked or nothing there; exit the game if the destination is an EXIT.

1. **Test LOOK**

The LOOK command can come with or without Direction or Object, meaning:

* LOOK: look around the surroundings of the room, describe current place and all outgoing paths.
* LOOK Direction: describe the path in the specific direction.
* LOOK HERE: look inside the room, list all items here without detailed description.
* LOOK Object: describe the specific item in the room or in the inventory.

1. **Test USE Artifacts**

Artifacts are treated as non-consuming, thus remaining in the INVE after executing the USE command (unless DROPped by the player).

After USE-ing a key to lock/unlock a path, even if the character leaves that place or later DROPs the key, those paths remain locked/unlocked.

After USE-ing a light to turn it on, the player carries its lighting with him. So if the character goes to another room with light active, the light will contribute to the lighting in this new room.

If a light is USEd (on) then DROPped, it will continue lighting.

# Timetable

|  |  |
| --- | --- |
| ***Time*** | ***Milestone*** |
| Due Apr. 21 (end of 13th week) | Implementation and debugging completed. |
| Due Apr. 28 (end of 14th week) | Code reviews and testing completed.  Testing report finished. |

1. Note there are 6 component sections totally in GDF v2.0, which include PLACES, PATHS, LIGHTING, ARTIFACTS, KEYS, and LIGHTS. The title line, containing the environment name, magic word and version number and so on, is not counted as a section. [↑](#footnote-ref-1)