Third Iteration of the Project (I3): Prototype of a Conversational Game

Introduction

The third iteration (I3) continues with the development of the project and the skills and concepts necessary for it, as well as the introduction and use of appropriate tools for this activity. In this iteration, the Goose Game will be abandoned as a reference for development, since it is no longer an appropriate model to illustrate the additional functionalities needed to complete a system that supports Conversational Adventures. However, it will demonstrate the versatility of the developed system, creating a new edition of the Game of the Goose that uses the new versions of the modules of the project obtained as a result of this iteration.

Figure 1 illustrates the relationship between the different modules used by I3, on the basis of the materials of the previous iterations. On this occasion, the fundamental modules will be finished as explained in the introduction to the project.

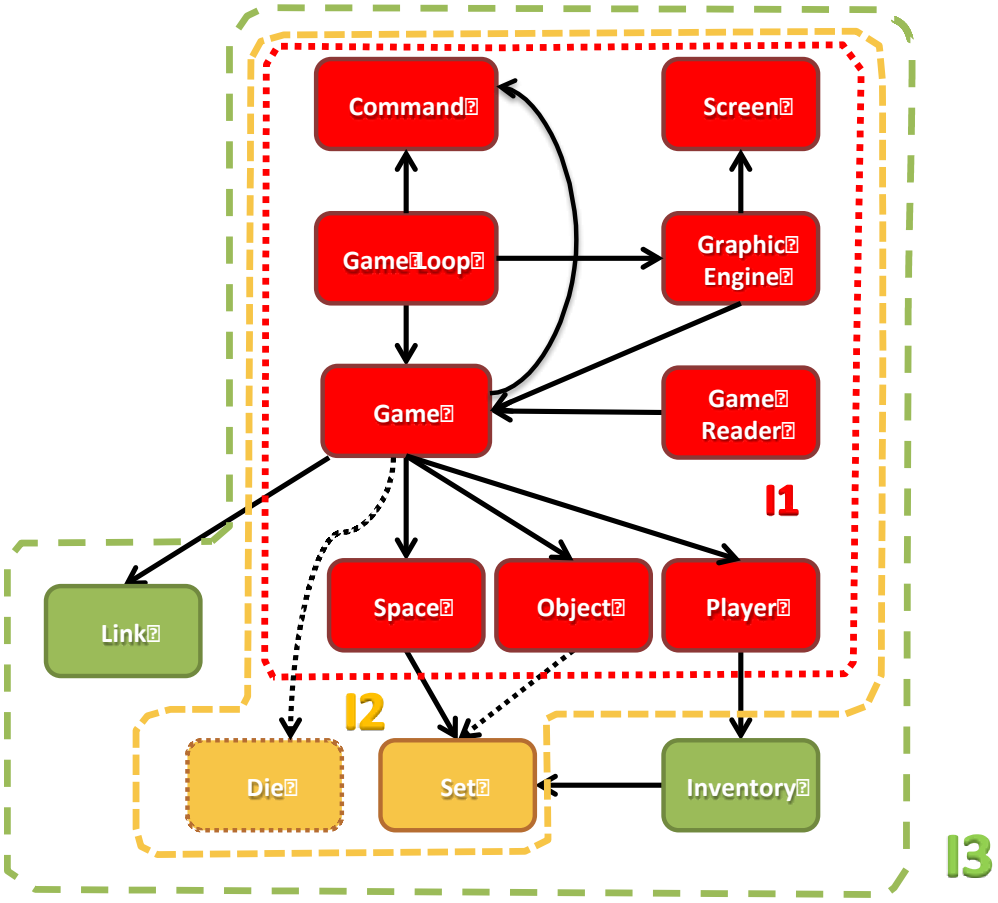

Figure 1. Modules involved in the third iteration (I3) of the project development.

Modules obtained as a result of I1 are shown in red, yellow is used for modules developed in I2. Finally, green modules need to be developed from scratch in I3. The two new modules are `Inventory` and `Link`. The `Inventory` module will be used to store the objects carried by each player (backpack). The `Link` module will introduce new options related with connections between spaces (for example, opened or closed doors, or locks that require a key).

In iterations I1 and I2 we generated an application that should be able to:

1. Load spaces and objects from a data file.
2. Manage everything needed to implement games with the characteristics of the Goose Game (spaces, objects, player capable of carrying an object, position on the board of an object and the player as well as movement between spaces linked together).
3. Support user interaction with the system, interpreting commands to move the player to manipulate objects (the player can carry a single object in I2, but there may be several in each space) and exit the program (in addition to use a die).
4. Display "graphically" (ASCII) the space in which the player is located and its adjacent spaces, plus the location of the objects as well as the value of the last die.
5. Release all resources used before the end of the program execution.

As a result of I3, it is expected:

1. A program that allows testing maps formed by spaces that contain objects. The spaces may be connected by links that are either open or close. They have a textual description (not only the ASCII picture) that may be inspected by the user. The objects have names and also a textual description. A player may take more than one object. The purpose of this program is to test the new functionalities implemented, which are more than necessary for the Goose Game.

2. A new version of the Goose Game that uses the new functionalities.

    a. The spaces will be connected by links that are always open.

    b. The player should be able to move towards north, south, east or west as an alternative to the already implemented movements (forward, back, east, and west).

    c. The player should be able to carry multiple objects in her inventory even if only one is needed for the Goose Game (by limiting the inventory size to one), the player should also be able to select a particular object from the inventory and leave/drop it

3. At the same time, the application should continue covering the Game Goose functionalities, but using connected spaces by links (always open), commands to move towards north, south, east or west as an alternative to the already implemented movements (forward, back, east, and west), and a player able to carry multiple objects, even though the backpack will only contain one object (the limit of objects carried will be 1), and a command for dropping a particular object from the player inventory. The objective of this program is to test the versatility of the developed system.

Due to all of that, the modules should incorporate the following new functionalities:

1. Loading the links between the spaces from a data file, as it is done for spaces and objects.
2. Managing the necessary data (spaces, links, objects, player, and the die if needed) for implementing the applications with the characteristics already mentioned.

3. Supporting the user interaction for carrying objects, examining descriptions of spaces and objects, and moving in between connected spaces using the four directions (north, south, east and west).
4. Showing the existing descriptions for spaces and objects.

Objectives

The objectives of this third iteration of the project (I3) are twofold: on the one hand, you should improve your: (a) management skills, (b) test automation (c) documentation (Doxygen) and (d) proficiency using tools for debugging programs (`gdb` and `valgrind`). On the other hand, you should put into practice everything learned and implement the requested features on the basis of I1 and I2.

The required improvements and modifications (requirements R1, R2, etc.), and the activities and tasks that should be done are:

1. [R1] Modify the `Game` module and make its data structure opaque, as it was done in the other modules of the project.
2. [R2] Create a new module `Inventory` so that each player has a container (backpack), which carries the identifier of various objects. This functionality is similar to the one implemented for the `Space` module so it can be related with several objects. Inventories should be implemented as a data structure with two fields, one for storing the set of identifiers (using the `Set` module developed in I2), and another to set the maximum number of objects that the user can carry. This maximum number is independent of the maximum size defined in the implementation of `Set` as it represents the size of the backpack. Do not forget to implement all necessary functions to `create` and `destroy` the inventory, change the carried objects (`set` and `get`) and `print` the inventory content. Additionally, you should create a program (`inventory_test`) to test this module.
3. [R3] Modify the `Player` module by incorporating an `Inventory`. This modification should allow players to carry several objects. In order to achieve this goal, replace in the `Player` data structure the pointer to an object by a pointer to an `Inventory`. You will need to: (a) modify accordingly all the functions that used the old player data structure and (b) add, if necessary, new methods able to access the inventory content as well as modify it. For example, functions for adding an element to the backpack, for knowing if an object is inside the backpack or for obtaining the object identifiers. Additionally, you should create a program (`player_test`) to test this module.
4. [R4] Create a new function in `GameReader` for loading the player information in the game, similar to the one for loading spaces and objects. The format of the line in the data file will be "`#p:1|ply1|0|1|`", where they are specified the player identifier, the name, position in the game and the maximum number of objects that can contain (backpack size).
5. [R5] Create a `Link` module which provides the necessary functionality for managing links between spaces. In particular, links should be implemented as a data structure with multiple fields: a unique identifier for each link, a name, two fields (containing identifiers) that indicate the spaces linked together, and a field to determine the status of the link (open or close). As always, the module should provide the necessary functions to `create` and `destroy` links, change the values of its fields (`set` and `get`) and `print`

the contents thereof for debugging. Additionally, you should create a program (`link_test`) to test this module.

6. [R6] Create a function in `GameReader` that can load links as we have done for spaces and objects, following the specified format in Table 1. These links allow to connect spaces in a more powerful way. With this purpose, it should be replaced all the spaces identifiers (used to connect a particular space with its north, east, south and west neighbours) in the `Space` structure, by the identifiers of four links that establish the corresponding connections with spaces in the four cardinal points. In addition to modifying the data structures you should modify all the `Space` primitives that use these fields, as well as adding new functions if necessary, for managing properly the links. For example, for determining if a space connects with another one via a link or if a link is passable. Additionally, you should update the program `space_test` to test the modifications included in this module.
7. [R7] Update the game visualization, for instance, by printing the id of each link in the visible spaces and the destination space ids, as shown in Figure 2.

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~         |               15|                              ~  Objects location:                    ~
~         |                 |                              ~  red:16, yellow:16, green:18          ~
~         |                 |                              ~                                       ~
~         |                 |                              ~  Player object: blue                  ~
~         |                 |                              ~                                       ~
~         +-----------------+                              ~  Last die value: 2                    ~
~                  ^ 15                                    ~                                       ~
~      30 +-----------------+                              ~  Descriptions:                        ~
~   8 <-- | >8D           16|                              ~  red - Red token                      ~
~         |      ___        |                              ~                                       ~
~         |     / _ \       |                              ~                                       ~
~         |    / / \ \      |                              ~                                       ~
~         |red,yellow       |                              ~                                       ~
~         +-----------------+                              ~                                       ~
~                  v 16                                    ~                                       ~
~         +-----------------+ 28                           ~                                       ~
~         |               17| --> 21                       ~                                       ~
~         |                 |                              ~                                       ~
~         |    <O)__/       |                              ~                                       ~
~         |     \_-/        |                              ~                                       ~
~         |                 |                              ~                                       ~
~                                                          ~                                       ~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ The game of the Goose ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~ The commands you can use are:                                                                    ~
~     next or n, back or b, take or t, drop or d, roll or rl, left or l, right or r, move or m, ins~
~pect or i, exit or e                                                                              ~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~ Take: OK                                                                                         ~
~ Inspect: OK                                                                                      ~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
prompt:>
```

Figure 2. Visualization of the state of the game: top left, square 15, empty; bottom, bridge square 16 with objects `red` y `yellow`, and the player `>8D`; bottom, goose square 17; right side, objects location, objects carried by the player, last die roll and the last description looked up.

8. [R8] Modify the data file needed for the new version of the Game of the Goose. Specifically, include information about links in the data file and modify the information about spaces accordingly. To facilitate this task you may use the Game of the Goose board map in Figure 3. In this figure, the geese squares (5, 9, 13, 17 and 21) are represented in grey, the bridges (8 and 16) in blue and the death (22) in black. The north/south links (sequential in the Goose Game) are continuous line arrows, while the east/west (special links for some squares) are discontinuous arrows. Doble arrows indicate that links can be used for communicating in both directions, for example, square 2 connects in the south with square 3 and square 3 connects in the north with square 2, both of them using link

2. Simple arrows indicate connections in just one direction, for example, square 13 communicates in the east with square 17 using link 27, but square 17 do not connect with 13 in the west. Square 22 communicates in the east and west with square 1 using link 31.

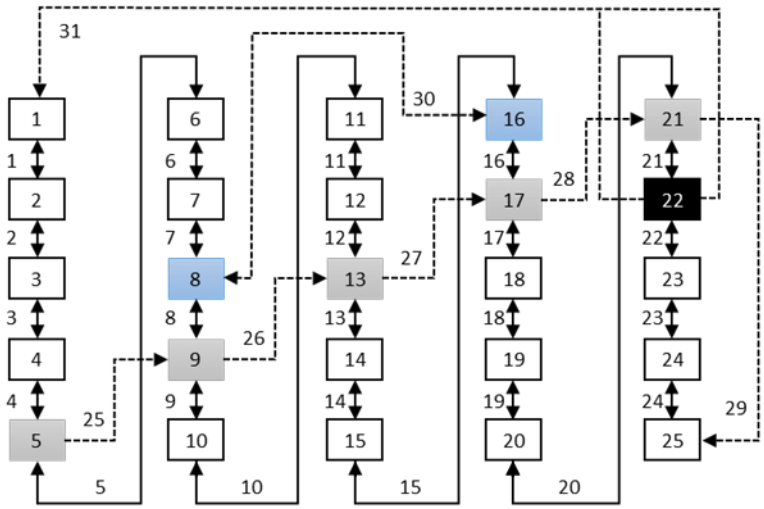

Figure 3. Example of the Game of the Goose board map, in which the spaces (squares) are represented using boxes, and the links between them are represented with arrows, both labelled with their corresponding numeric identifiers (assigned in the data file).

Table 1 shows a fragment of the data file used to store space information as used in I2 (top) and the equivalent implementation of the fragment for I3 (row I3), according to the map in Figure 3.

I2: Data file fragment of the spaces in the **I2** style, where the connections between spaces where directly stablished using the identifiers.	… #s:3|Spc 3|2|-1|4|-1| #s:4|Spc 4|3|-1|5|-1| #s:5|Spc 5 (Oca)|4|-1|6|9| |<o)__/ | _=/ | #s:6|Spc 6|5|-1|7|-1| #s:7|Spc 7|6|-1|8|-1| …
I3: Data file fragments of the spaces (up) and links (bottom) in the **I3** style, where the connections between spaces are stablished using the links defined in the data file. The links can have some properties (open/close) that in iteration **I2** where impossible. In particular, column 1 stablish the link id, column 3 the id of one of the connected spaces, column 4 the id of the other connected space, and column 5 indicates if it is open/close (0/1 respectively).	… #s:3|Spc 3|2|-1|3|-1| #s:4|Spc 4|3|-1|4|-1| #s:5|Spc 5 (Oca)|4|-1|5|25| |<o)__/ | _=/ | #s:6|Spc 6|5|-1|6|-1| #s:7|Spc 7|6|-1|7|-1| … … #l:2|Lnk 2|2|3|0| #l:3|Lnk 3|3|4|0| #l:4|Lnk 4|4|5|0| #l:5|Lnk 5|5|6|0| #l:6|Lnk 6|6|7|0| #l:7|Lnk 7|7|8|0| … #l:25|Lnk 25|5|9|0| …

Table 1. Data file fragment related to spaces according with styles in iterations **I2** and **I3**.

9. [R9] Add a new command to move through the spaces following the links in a generic way. Use the word "`move`" followed by the direction (either north, south, east or west). For example, "`move north`" or "`move west`" (or compact form "`m n`" o "`m w`").
10. [R10] Add a command to examine objects. This command should return the object description using a new `description` field in the `Object` structure. This command should include the object to be examined. For example, "`inspect book` " (or compact form "`i book` "). This command allows to examine objects that are either in the space where the player is located or the ones in the player's backpack (inventory). Additionally, you should create a program (`object_test`) to test this module.
11. [R11] Modify the `inspect` command to make possible to inspect the space description where the player is located. For this purpose, add a new `description` field in the `Space` structure, and use a special term for this task ("`space`"). For example "`inspect space`" (or its compact form "`i s`") will show the field `description` of the space where the player is. The reserved keywords "`space`" and "`s`" cannot be used for objects.
12. [R12] Modify, if necessary, other existing modules to use the new implemented and modified modules, and for maintaining the previous functionality and adding the new one. For example, in the `Game` module: (a) include a vector (array) in the corresponding data structure to keep all links in place, as it was done for spaces and objects; (b) modify the initialization routines to handle properly the new field; and (c) update all existing functions so that they use this new vector field. Do not forget to modify the test programs of each module so that they incorporate the changes.
13. [R13] Modify the commands `take` and `drop` to use the name of the object (instead of the identifier). In this way, instead of writing "`take o12`" we will write "`take book`". Also modify the game visualization, so the object names are shown instead of their identifiers (see Figure 2).
14. [R14] Provide at least two files with different commands in order to test the new functionality added: the first one must use the command's format used in I2 (`next`, `back`, etc.), and the second one should implement their equivalence in I3 (`move south`, `move north`, etc.).

    To write the command files, you can use the log file that is generated with the "`-l`" option implemented in iteration 2, and then replace the old commands with the new ones to generate the second file.

    Both files should have one command per line following the format "`VERB GOBJECT`", where `VERB` should be the action name, in particular `move`, `take`, `drop`, `inspect` or `exit`, and where `GOBJECT` should be the grammatical object on which the action is performed, like the name of some cardinal point (`north`, `east`, `south` and `west`), the name of a game object (e.g. `lantern`, `notebook`, or `gun`), some reserved keyword (like `space` in the previous requirement), or nothing, for the actions that do not need a grammatical object (e.g. `exit`). It is compulsory to follow this nomenclature. An example of such a file would be:

```
move north

move north

move west
```

```
take lantern

move west

drop lantern

inspect lantern

…

exit
```

To receive this instructions file in the program, you should not read this file from the main program. Instead of that, you should use the redirection capabilities provided by the command shell. In this way the file will be piped to the standard input and it would behave as a command typed by a user. For example, if the command line executed to play the Goose Game in I3 is:

```
./goosegame data.dat
```

The command for reading the file, if an instructions file called `round1.goose` is prepared for the game, will be:

```
./goosegame data.dat < round1.goose
```

15. Manage the project during iteration I3 by holding meetings documented with minutes that include commitments for each team member, assignments, and terms and conditions of deliveries. Also by planning the project (record tasks, resources and time) with Gantt charts. Monitor your plan, adjust it if needed and reflect the new schedule in the Gantt diagram.
16. Document the project using Doxygen. First, modify all source files including the appropriate labels (as indicated in class). Remember to update (a) the header comments in ALL ".h" and ".c" files; (b) the comments in the function headers of all `.h` files; and (c) in the comments of enumerated types and data structures of the `.h` files. Once the files have been updated, generate the documentation using Doxygen in HTML format.
17. Modify the `Makefile` to automate the compilation and linking of the whole project, so the files are organised in subdirectories in the following way:
    a. Source code files (`.c`) in subdirectory `./src`.
    b. Header files (`.h`) in subdirectory `./include`.
    c. Object files (`.o`) in subdirectory `./obj`.
    d. Documentation (generated with Doxygen) `./doc`.
18. Make unit tests for all the modules in the project, as well as integration tests of all the modules in the project (the files created in R14 can be useful for these tests). For those functions that are difficult to test automatically, the manual tests done must be specified in a test document.
19. Debug the code until it works properly.

Assessment Criteria

The final score for this assignment is part of the final grade according to the percentage set for I3 at the beginning of the course. In particular, the assessment of this deliverable is calculated according to the following criteria:

- **C**: If C is obtained in every row of the rubric table.
- **B**: If at least four Bs are obtained, and the remaining rows are Cs. Exceptionally with only three B.
- **A**: If the project gets at least four As and the remaining rows are Bs. Exceptionally with only three A.

Every submission that does not obtain the requirements of column C will obtain a score lower than 5.

Rubric table:

	C **(5 - 6,9)**	**B** **(7 - 8,9)**	**A** **(9 - 10)**
Delivery and compilation	(a) All the required files have been delivered on time. AND (b) It is possible to compile and link all the sources to get an executable file using `Makefile` (including the test programs).	In addition to the previous column: (a) Compilation and linking do not provide error messages neither warnings when compiling with `-Wall`. AND (b) The files follow the directory structure proposed in point R17.	In addition to the previous column: (a) Compilation and linking do not provide error messages neither warnings when compiling with `-Wall -pedantic`. AND (b) The delivered `Makefile` produces the technical documentation using Doxygen under a default task.
Functionality	Requirements from R1 to R6 are satisfied.	In addition to the previous column: Requirements from R7 to R12 are satisfied.	In addition to the previous column: Requirements from R13 and R14 are satisfied.
Test	At least two unit tests have been implemented for each function of the modules related with requirements from R1 to R6.	In addition to the previous column: At least two relevant unit tests have been performed for each function of the modules involved in the requirements from R7 to R12.	In addition to the previous column: Integration tests are defined.

Coding style and documentation	(a) Variables and functions have names that help to understand their purpose. AND (b) All constants, global variables, public enumerations, and public structs are documented. AND (c) The code is properly indented[1]. AND (d) Source files and functions contain a header, are correctly commented, and include all the required fields, including data about unique author.	In addition to the previous column: (a) Module interfaces are NOT violated. AND (b) Doxygen comments are included in: • EVERY file header. • Function prototypes. • Enumerated types and data structures. AND (c) Check for errors in arguments and returns of functions used for resource allocation or update.	In addition to the previous column: (a) Coding style is homogeneous[2]. AND (b) There is an explanatory comment for each local variable when it is needed. AND (c) Technical documentation is correctly generated in HTML format using Doxygen.
Project Management	At least one meeting report is submitted describing the work organization of the team for I3. The report should include a Gantt diagram. The first version of the chronogram should be submitted during the first week of this iteration.	In addition to the previous column: At least two meeting reports with Gantt diagrams are submitted describing the work organization at the beginning and at the end of I3.	In addition to the previous column: One meeting report with Gantt diagram is submitted for each week of the duration of I3. The reports should reflect and justify the changes in organization as the work progresses.

[1] Indentation should be homogeneous. Every code block at the same level must have the same indentation. Besides, either tabulation characters or spaces (always the same number of spaces for each level) must be used, and combinations are not allowed.

[2] At least the function names should start with the name of the module; the variables, functions, etc. should follow either a *camel case* or *snake case* notation, but they should never be mixed; the coding style should always be the same (e.g. *K&R*, *Linux coding conventions*, etc.) and never mix different coding styles.