ESIEEquest - Project report

IGI-1002 A3P

Pacien TRAN-GIRARD

Benoît LUBRANO DI SBARAGLIONE

ESIEE
PARIS

April 2014

# Contents

1 The project **5**

1.1 About . . . 5

Context . . . 5

Authors . . . 5

1.2 The game . . . 5

Theme . . . 5

Map . . . 5

Scenario . . . 7

2 Work progression **11**

2.1 Zuul better v1 . . . 11

printLocationInfo . . . 11

getExit . . . 11

getExitString . . . 11

HashMap, setExit . . . 11

Vertical direction . . . 11

keySet ? . . . 11

getExitString ? . . . 12

getLongDescription . . . 12

2.2 Zuul with features . . . 12

look . . . 12

eat . . . 12

2.3 Zuul better v2 . . . 12

showAll, showCommands . . . 12

Adding commands . . . 12

getCommandList . . . 12

Comparison with reference . . . 13

StringBuilder . . . 13

Room objects . . . 13

2.4 Zuul with images . . . 13

Game, GameEngine, UserInterface . . . 13

Parser/Scanner . . . 14
addActionListener() and actionPerformed() . . . 14
Add a button . . . 14
2.5 Zuul MVC . . . 14
Structure . . . 14
2.6 Zuul with items v1 . . . 15
Item . . . 15
item description . . . 15
items . . . 15
Collection choice . . . 15
2.7 Zuul with history . . . 15
back . . . 15
back test . . . 15
back back . . . 15
Stack . . . 16
2.8 Zuul with tests . . . 16
tests . . . 16
test command . . . 16
2.9 Zuul with items v2 . . . 17
Player . . . 17
take, drop . . . 17
Carry several items . . . 17
ItemList . . . 17
Maximum weight . . . 17
Inventory . . . 17
Magic cookie . . . 17
Tests . . . 17
2.10 Zuul with enums . . . 18
switch . . . 18
help with enum . . . 18
2.11 Zuul extended . . . 18
Time limit . . . 18
GUI . . . 18
Trap door . . . 18
Beamer . . . 18
Locked door . . . 19
Tests . . . 19
Transporter room . . . 19
Character . . . 19

2.12 Zuul even better . . . 19
Inheritance . . . 19
Abstract Command . . . 20
Packages . . . 20
2.13 Zuul without BlueJ . . . 20
main . . . 20
jar . . . 20
JApplet . . . 20
2.14 Zuul awesome . . . 20
Non-droppable items . . . 20
Hidden doors . . . 20
Room sides . . . 21
Quests . . . 21
Better GUI . . . 21
Music . . . 21
Scenes . . . 21
Save/load game . . . 21

3 External packages **23**
3.1 StretchIcon . . . 23
3.2 Guava . . . 23
3.3 Lombok . . . 23
3.4 Intrinsic Map . . . 23
3.5 Google Web Toolkit . . . 23
3.6 JSON Simple . . . 24
rejava.io . . . 24

4 Tools **25**
4.1 Version Control System: Git . . . 25
4.2 Integrated Development Environment: Eclipse . . . 25
4.3 Continuous Integration . . . 25

A User guide **27**
A.1 Running the game . . . 27
Web version . . . 27
Standalone app . . . 27
Console mode . . . 27
A.2 Commands . . . 28
A.3 Keyboard shortcuts . . . 29

1 The project

1.1 About

Context

The development of this game was conducted in part of the project-based learning unit IGI-1002, in which students are asked to write a simple adventure game in Java.

Authors

This game was fully developped by Pacien TRAN-GIRARD and Benoît LUBRANO DI SBARAGLIONE, two first year students ('18) at ESIEE Paris.

1.2 The game

Theme

A student attends a presentation at ESIEE Paris, when the Universe suddenly crashes. He must then find a way to reboot it.

Map

The map was divided into two parts: one playable, and one off-script.

Main map

The main map is the one in which the story unfolds, and where the player can move. The layout is based on the architecture of the building of the school, and only essential rooms are there.

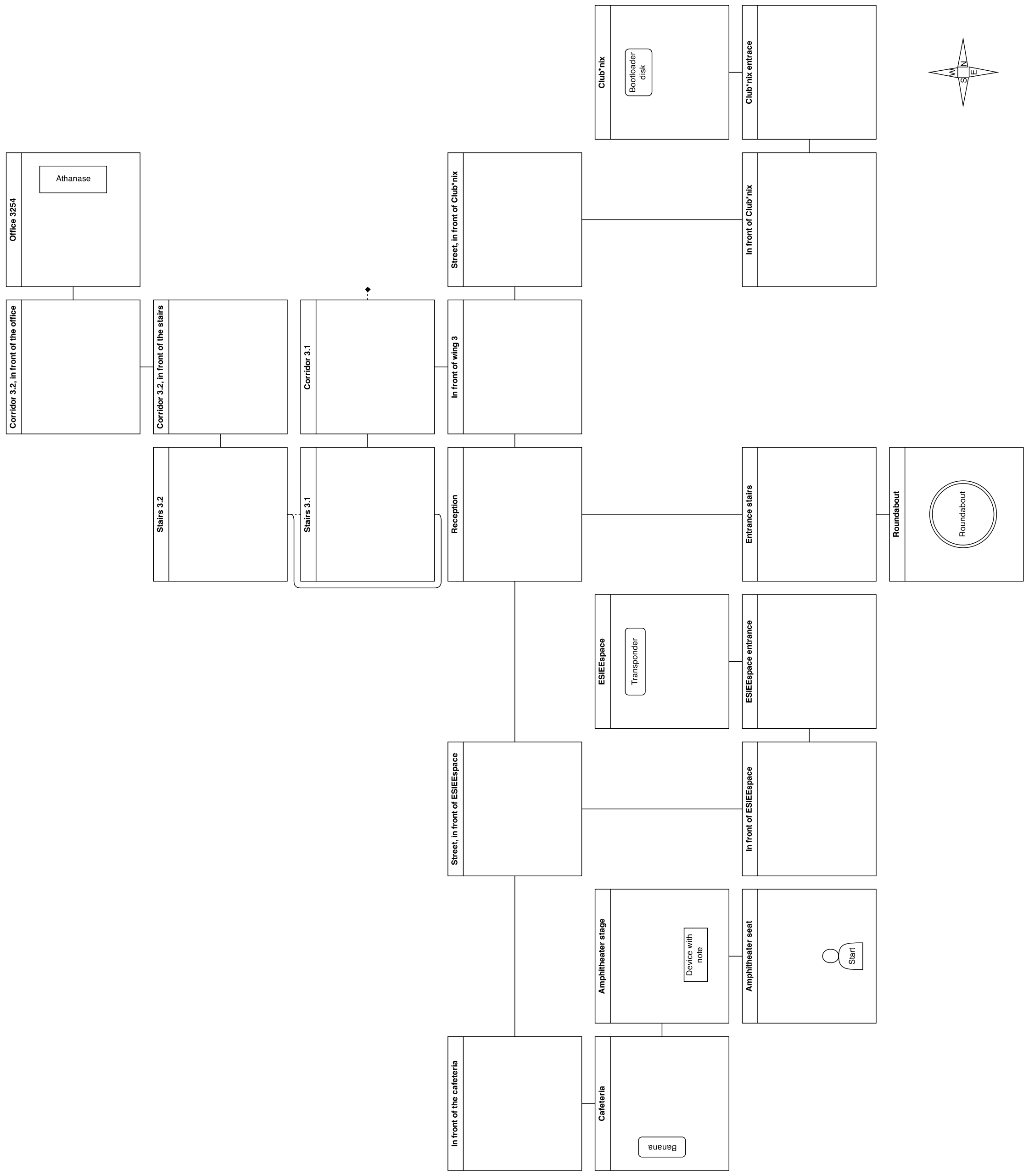

Figure 1.1: Main map

Off-script map

A second map was created to include the mandatory elements required in the exercises but being out of the scenario. Although independent, it can be accessed via the main map by entering the toilet (located on the ground floor of the third wing).

After entering this area, it is (deliberately) no longer possible to return to the main map and to continue the game.

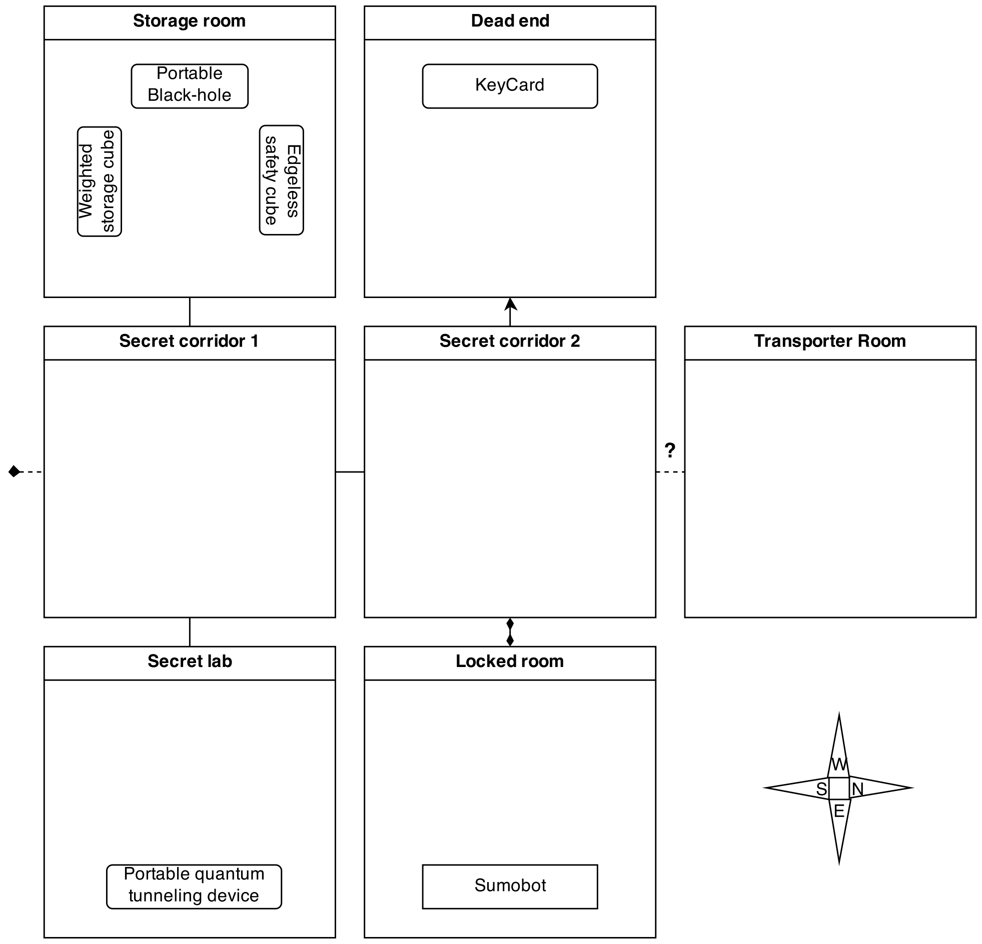

Figure 1.2: Off-script map

Scenario

The presentation of a student project goes wrong, and you are - almost - the only one who can save the world.

Preamble

The player attends the presentation of a mysterious project, led by Pacien and Benoît, both on stage.

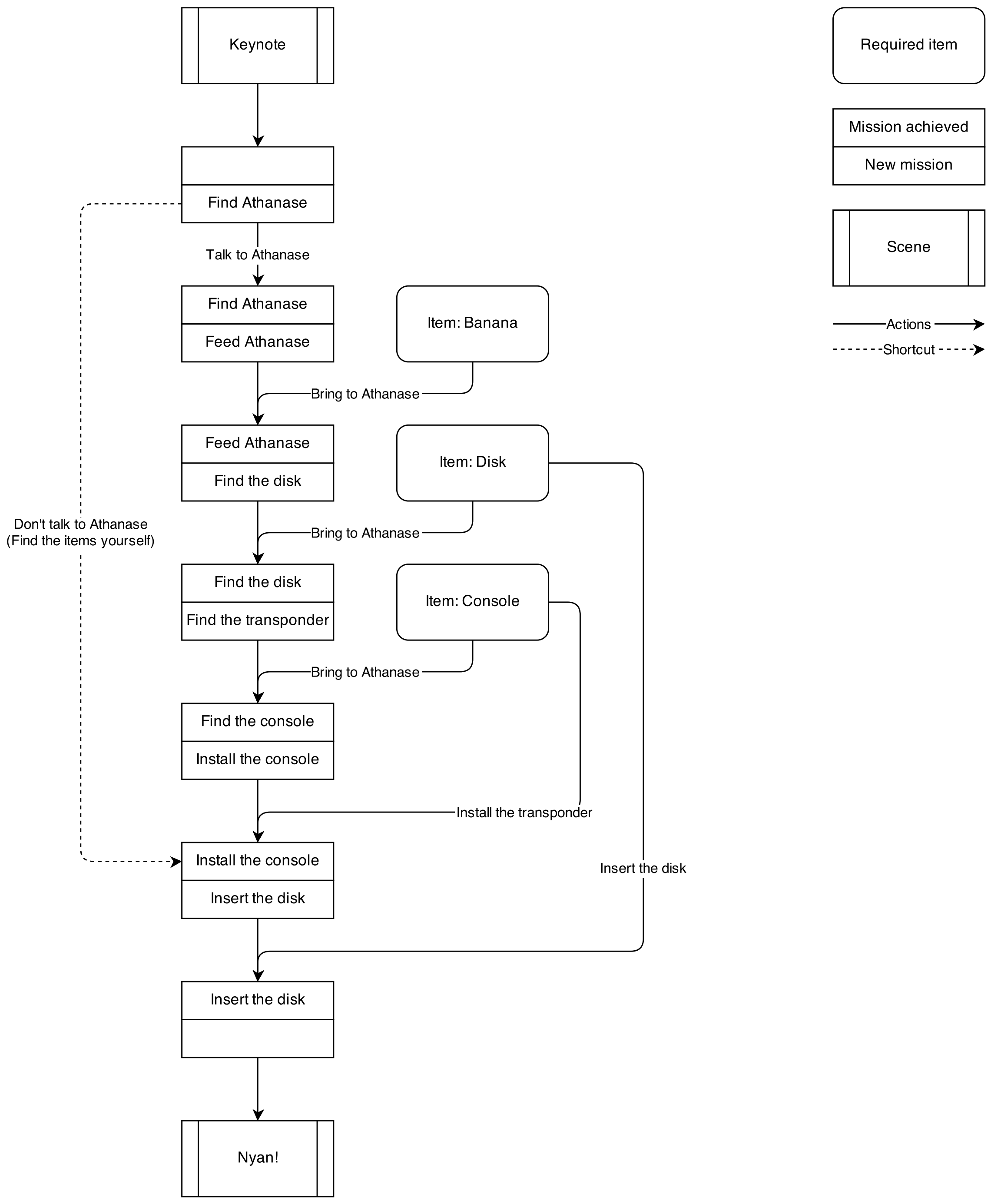

Figure 1.3: Scenario diagram

As it begins, a corrupt image appears instead of the slideshow. Both students therefore decides to demonstrate their invention, hidden from view of the player by the public in front. Barely begun, the light suddenly turns off, and then returns.

The player discovers that everyone has disappeared.

He goes on the stage and finds a note instead of the machine: "In case of emergency, find Athanase, Office 3254."

First quest: the friend

Athanase is very, very hungry, and makes it clear to the player.

→ The player must bring a banana from the cafeteria to Athanase.

Second quest: the program

Athanase tells the player that the Universe has crashed, and that there is a disk containing an emergency program created by someone called Jesus to overcome this incongruous situation.

→ The player must bring this disk, hidden at the Club*nix, to Athanase.

Third quest: the device

Athanase asks the player to connect to the Universe in order to run this program. To achieve this, he must use a portable transponder developed at ESIEEspace.

→ The player must pick up the transponder at ESIEEspace and bring it to Athanase.

Fourth quest: the new start

Athanase asks the player to run the program on the transponder. However, in order that it can connect to the Universe , he must find a way to amplify the signal, with an antenna.

→ The player must go to the roundabout in front of the ESIEE main entrance and use the giant peak as an antenna.

Ending

The transponder connects to the Universe and the initialisation script starts.

2 Work progression

2.1 Zuul better v1

printLocationInfo

e52a34789 - Pacien - This code duplication has been previously avoided with the printRoomInfo() method. It has been renamed to printLocationInfo() to match the exercise.

getExit

53c427ff3 - Pacien - The exit attributes have been made private and a getter has been added. A switch statement has been used instead of multiple if statements.

getExitString

ca65af2e2 - Pacien - This method, which returns a String containing the informations about the Room's exits, has been added.

HashMap, setExit

c9d890b9b - Pacien - Room exits are now stored in an HashMap. The setExits() method has been replaced by setExit() which takes advantages of the HashMap.

Vertical direction

4145a5e8c - Pacien - The getExitString() method has been modified to be able to print the availability of the new exits, used in the stairwell at wing 3. Due to the architecture of these rooms, the side exits that were previously settled to link them have been kept.

keySet ?

Benoît - The keySet() method of the class HashMap returns a Set of the keys associated to values stored in the Map.

getExitString ?

Benoît - The getExitString() method returns a String listing the Room's exits. To achieve that, it iterates through the exits Map's keys, which is a Set of String-s, appending each one to the String that it returns.

getLongDescription

e510b08d0 - Pacien - The Room class now uses the previously explained getExitString() method and includes the getLongDescription() method that returns the full description of the room.

2.2 Zuul with features

look

698e3cd25 - Benoît - The look command, which prints informations about the current Room, has been added.

eat

40b9b4816 - Benoît - The eat command, that just prints a special message, has been added. This command has then been deleted later in the development since it was useless in the scenario.

2.3 Zuul better v2

showAll, showCommands

79d33230b - Pacien - Theses methods have been implemented.

Adding commands

Benoît - Adding new commands would not require modifying the printHelp() method anymore. However, adding new commands and related functions would still require editing the Game class and its processCommand() method.

getCommandList

5f1d0ada2 - Benoît - The command list is not printed in the CommandWords class anymore. Instead, this class returns a String, forwarded by the Parser class, that is then printed in the Game class.

Comparison with reference

590a932e5 - Pacien - The printLocationInfo() method, used only twice, has been trimmed and has been replaced by a call to the getLongDescription() method of the Room class.

f84606424 - Benoît - A missing getter for the Room description has been added.

0c5793abf - Benoît - The loop building the command list String has been modified to use an Iterator.

StringBuilder

ee5ec33aa - Benoît - The command list and the exit list are now created using a StringBuilder. This avoids the creation of a new String object at each concatenation, and thus allows better performances.

Room objects

f64f1ffb0 - Pacien - Rooms are now stored in an HashMap, so they can be passed to any method in any class.

2.4 Zuul with images

Game, GameEngine, UserInterface

54e102463 - Pacien - The methods have been implemented.

Game

Pacien - The constructor of this class instantiates the GameEngine and the UserInterface, and sets the output of the first to the second.

GameEngine

Pacien - This class contains the attributes and methods previously contained in the Game class.

Instead of printing to the standard console output, it prints to the Swing GUI.

UserInterface

Pacien - This class implements the graphical user interface of the game. It basically creates the different window components and provides methods to interact with them.

Parser/Scanner

Pacien - Since commands are entered through a text field instead of the console, the use of Scanner that read from the standard system input is not required for the Swing GUI.

addActionListener() and actionPerformed()

Pacien - The addActionListener() of an object x takes as parameter an object y that have a actionPerformed() method. When an action event occurs on x, this method on y is called and an ActionEvent is passed as parameter.

The actionPerformed() method then does the appropriate action according to the event that happened.

Add a button

7f153a4c1 - Benoît - A help button has been added. The processCommand() method has been modified to take the command String as a parameter instead of reading it only from the text field, and the actionPerformed() method has been modified accordingly to forward the command.

2.5 Zuul MVC

Structure

Pacien - Three packages were made to help matching this architecture: model, view and controller.

The model package contains the objects storing the game state and various elements representing a data, that is to say the Game, Room and Command classes.

The view package contains the user interfaces, rich and console, that displays the game state to the user and enable him to interact with the game by firing events to the controller.

The controller consists of several classes and represents the logic part of the program, which are the GameEngine, the Interpret, the Parser and the Performer. They handle the events created by the user's actions and modify the Game model accordingly, refreshing the view to display the new game state.

bd639caa0 - Pacien - Several internal modifications in the existing classes were made in order to separate the logic from the data storage.

Later in the development, it has been found that this pattern is not easily usable in a language like Java, and has been replaced by a model-coordinator-view pattern in this project. Then, this pattern faded as some classes had to implement logical interfaces themselves. However, the view/ui package has been kept appart.

2.6 Zuul with items v1

Item

3fc06f393 - Pacien - Item support has been added.

item description

Pacien - In order to respect the MVC pattern, the Performer controller should get the description from the Item model and ask the View to display it.

items

04f03d128 - Pacien - A Room can now contain multiple items, which are stored in an HashMap.

Collection choice

Pacien - The collection chosen to implement the multiple items support was the HashMap in order to associate an item name (String) to an Item, so that the user is able to perform actions on a particular one (take, drop, use, etc. . .).

2.7 Zuul with history

back

f218bf5fa - Pacien - The back command has been implemented.

back test

Benoît - The back command works even with any parameter. We will check this later with “zuul-with-enums“.

back back

Benoît - When the user enters back twice, he goes two rooms backward. This is working correctly.

Stack

Pacien - The Stack class represents a last-in-first-out (LIFO) stack of objects. The push() method allows to add and element at the top of the stack and the pop() method allows to retrieve the top item and removes it.

A condition checks whether the Stack is empty before calling the pop() method, so that the program does not throw an Exception. This solves the case when the player is at the starting point.

2.8 Zuul with tests

tests

Benoît - In the current version of the game, the outputs of commands such as help or quit might be easily tested. Furthermore, movement from Room-s to Room-s, and arising modifications on the model, can also be candidates for testing.

Automatic tests

Pacien - Functionalities of the program can be tested automatically using unit testing for classes and methods, and interactive testing to simulate the user's interactions.

Unit tests can later be implemented using the JUnit testing framework.

User interactions like command input can be simulated by reading and interpreting commands from a text file.

test command

967f40d71 - Pacien - The test command has been implemented by modifying the Console view. Instead of reading commands from the user's input from the console, the Scanner reads the commands from a text file.

This functionnality was not implemented as a command inside the game, but as a command line flag that can be used by executing the program from a terminal with "–file <file path>".

test files

Benoît - Three test files have been created. The first one tests simple commands such as help and quit, the second one explores all rooms, and the last one executes all actions to win the game.

2.9 Zuul with items v2

Player

92f671b84 - Benoît - The current Room and the previous Room-s were moved from the Game class to the Player class.

take, drop

5ddb6df63 - Pacien - The take and drop commands have been implemented. An Inventory interface, used by the Room and Player classes, was added. Items are stored using HashMaps, and a moveItem procedure was added to the Performer.

Carry several items

5ddb6df63 - Pacien - The possibility to carry multiple Item-s was already implemented.

ItemList

db2c22c9b - Pacien - The ItemList was implemented as the Inventory class. The Inventory is no longer an interface.

Maximum weight

b8771ccb2 - Benoît - The maximum carryable weight has been implement. In order to achieve this, a getTotalWeight() function was added to the Inventory class.

Inventory

a3f6ce16e - Pacien - The Player's Inventory Item-s listing has been implemented, using the recently added listing utility class.

Magic cookie

35ee3b1ee - Pacien - Since there are already too many commands, the carryable weight expansion was implemented in a smarter way using a negative weight item. We could have instead increased the maximum carryable weight with a setter on the Player class, called by the Performer with the “eat” command.

Tests

7b610fc05 - Benoît - The commands test file was updated.

2.10 Zuul with enums

switch

d24dd6cc9 - Pacien - The String switch statement was replaced by an enum switch. Instead of using an HashMap to match entered Strings to the corresponding enum value, the valueOf() method was used.

help with enum

159b168b2 - Pacien - The help command now generates the list of the available commands from the enum.

2.11 Zuul extended

Time limit

2fee72805 - Benoît - The limit was implemented as a steps limit for the player. When this limit is reached, the game quits with a message. This limit can be activated by initiating a new game using the “new challenge“ command.

GUI

The current GUI will be improved at the end of the project, once all new functionalities will be implemented.

Trap door

6e388ec78 - Pacien - A TrapDoor has been added. When crossing a TrapDoor, the Performer clears the Player’s previous Rooms Stack.

It has been used to link the secret corridor of the off-script map to a trap Room, which can possibly be escaped using the Beamer.

In later versions, the TrapDoor extends the Door class and implements its own cross() function, which performs the action the clear the Player’s Stack.

Beamer

6a154d2fe - Pacien * A Beamer has been added. It can be used once picked up via the “use” command. It alternatively memorises the Player’s current Room and teleports him back.

Locked door

e9c61548b - Pacien - A locked door has been added in the off-script area. To pass through, the player needs to have a corresponding Key in his Inventory.

Tests

Benoît - The test files were updated to fit the last modifications.

Transporter room

c5a766345 - Pacien - The transporter has been implemented as a sub-type of Door instead of Room, since the Player has to be transported when he crosses the Door.

A transporter Room, which is a simple Room that has only TransporterDoor-s, has then been added in the off-script area.

alea

c99590be11 - Pacien - An "alea" overriding command for the random TransporterDoor was implemented. It is now possible to force the destination Room with this command.

Character

3e7f68425 - Pacien - Characters that can talk have been added.

Moving character

Pacien - Moving characters that can move randomly (WanderingCharacter) or by following the player's moves (FollowingCharacter) once permitted have been added.

The MovingCharacter class contains a method moveAll() which is called each time the player enters a command.

2.12 Zuul even better

86d6f56dd - Pacien - A code refactoring was done.

Inheritance

Pacien - Inheritance (and interfaces) was already used for Character-s, Door-s, Item-s, and View-s since the beginning of the project.

Abstract Command

Pacien - The Performer class was replaced by multiple Command sub-types. Instead of using an AbstractCommand class, an interface (Executable) was created, and the command arguments passed by parameter instead of being stored in a field. This makes more sense than instantiating a Command each time it is executed.

Packages

Pacien - The code was already organised in packages.

The “pkg_” prefix was NOT used in this project, since it led to inconsistences with other packages.

2.13 Zuul without BlueJ

main

Pacien - The main() static method has been already implement in the Main class.

jar

Pacien - An executable .jar archive containing the Game, the manifest file, and some resources (pictures and musics) is automatically generated at each modification.

JApplet

Benoît - An init() method has been added to the startup Main class and an HTML file embedding the applet has been created at /applet.html.

Since applets are deprecated, an alternative has been implemented (see 2.14).

2.14 Zuul awesome

Non-droppable items

ef51d5137 - Benoît - Non-droppable items were added.

Hidden doors

20c9b8b28 - Benoît - Hidden Doors that are not shown as exits were added and used to hide the off-script part of the map.

Room sides

Pacien - Each Room has been divided into four Side-s that have each an corresponding image (or a dynamically generated placeholder image if not found) and inventory. This makes the Player look at the elements as the main character, rather than looking by the top, and makes easier for him to navigate in the map.

The “Turn” has been also added to enable the Player to turn on himself.

Quests

Pacien - Quests were added to the Game to guide the Player to the predefined goals and to defines losing and winning situations.

Better GUI

Pacien - As applets were designated as deprecated a long time ago, and because newer versions of the Oracle Java Runtime Environment are requiring a mandatory code signing certificate that we can not affors, a new graphical interface using the Google Web Toolkit was implemented. This set of tools enabled translating the game code from Java to JavaScript, allowing its execution directly in the user’s browser, without the need of having the Java Runtime Environment installed.

12a0686cb - This addition was facilitated by the use of the MVC model and required a few changes in the core classes.

Music

Pacien - Background musics corresponding to the current Quest were added, using the new HTML5 <audio> tag for the web interface, and the EasyOgg library for the Swing GUI.

Scenes

Pacien * An introduction and a conclusion scene were added.

Save/load game

Pacien - The save and load features were implemented using serialisation via the JSON simple library.

save

Pacien - A recursive call through all the model’s Serialisable objects enables the JSON serialisation of the current state of the Game.

The resulting JSON Object String is written and stored in a text file for the Swing Rich GUI and the console mode. In the web interface, it is stored in the browser’s content storage.

This feature introduces the “save <file name>” command.

load

Pacien - The Game can restore its state from a previously saved JSON String by deserialisation via the “load <file name>” command.

3 External packages

Some external packages that are not part of the Java Standard Development Kit were used in this project.

3.1 StretchIcon

StretchIcon enabled the resize of the Room illustration according to the window's size in the rich Swing GUI, keeping a correct ratio.

3.2 Guava

Guava, from Google, contains several collections, caching, primitives support, concurrency, annotations and string processing utilities.

3.3 Lombok

Lombok helped avoiding writing dozens of getters and setters by replacing them with shorter annotations.

3.4 Intrinsic Map

As Items were already containing there names in a field, HashMap<String, Item> were redundant. IntrinsicMap<Item> and the Mappable interface were created to avoid this redundance.

3.5 Google Web Toolkit

The Google Web Toolkit was used to create the web version of the game. It permitted the compilation of Java sources to Javascript code executable directly inside a web browser, without requiring the Java Virtual Machine.

3.6 JSON Simple

JSON Simple was used to implement the save and load functionalities by enabling the serialisation of the game datas into JSON Objects.

rejava.io

As some packages required by JSON Simple were not compatible with GWT, the standard Java StringReader and StringWriter classes were partially rewritten.

4 Tools

In order to simplify and organise the work around this project, several tools were used.

4.1 Version Control System: Git

In order to keep tracks of the changes in the code and to simplify the sharing of the sources by the team, the Git version control system was used.

A Gitlab server has been used as a central hub.

4.2 Integrated Development Environment: Eclipse

As BlueJ quickly shown its limits for other tasks than simple exercises programs, the team needed a more complete development environment.

Eclipse was chosen due to its better autocompletion and debugging features, and the availibility of numerous useful plugins such as WindowBuilder, Git and Google Plugin.

4.3 Continuous Integration

The periodic update of the website, compilation, regeneration of the Javadocs and of the .jar file were automated using ant and Gitlab CI.

These tasks were executed automagically on a server at each commit.

A User guide

A.1 Running the game

Feature	Web	App	Console
Animations	YES	NO	NO
Sound	YES	YES	NO
Illustrations	YES	YES	NO
Visual inventory	YES	YES	NO
Keyboard shortcuts	YES	YES	NO

Web version

To start the web version of the game, simply go to the project's website using an up to date web browser.

Standalone app

An offline version of the game referred as "standalone app" can be executed by opening the .jar. The Java Runtime Environment is required.

Because of the size limit imposed by JNews, the version sent for evaluation contains low resolution images. To obtain the full version of the game, please go to the project webpage.

Applet

This interface can be executed as an applet on the project's website /applet.html.

Console mode

To start the game in the console mode, open the .jar via the command line with the "–console" argument.

Test mode

To execute the commands contained in a test file, execute the game from the command line with the “–file” argument, followed by the path to the test file.

A.2 Commands

Command	Argument	Description
alea	<room name>	Force the destination of the transporter doors.
back	none	Go back to the previous room.
do	none	Automatically take an item or talk to a character, according to the context.
drop	<item>	Drop an item in the current side of the current room.
go	<cardinal direction|none>	Go in the given direction or forward (in the direction of the current side)
help	none	Display the available commands.
inventory	none	Display the inventory.
load	<file path>	Load a saved game from a file.
look	none	Display informations about the current room.
new	<challenge|none>	Create a new game, optionally in challenge mode.
quit	none	Quit the game.
save	<file path>	Save the game to a file.
sound	none	Toggle the sound (GUI and web only).
take	<item|none>	Take an item from the room. Take one of the item of the current side if no name specified.
talk	<character|none>	Talk to a character in the room. Talk to the character directly facing the player if no name specified.
turn	<left|right>	Turn in the given direction.
use	<item>	Use an item.

A.3 Keyboard shortcuts

Key	Action
Left arrow	Turn left
Right arrow	Turn right
Up arrow	Go forward
Down arrow	Go back
PgUp	Take/Talk
Home	Open/Close the inventory