Lorann-Ex

Mercredi 06 Juin 2018

# Planning

At the beginning of the project we had to organize all the project to give differents tasks to the members of the team.

## Team

The team is composed of four persons each have a specific task in the project.

Rafik Belkadi

*Project leader*: In charge of the organisation of the project and the Controller and Model packages

Dounia Benmerrar

In charge of the view package of the project

Ghada Mensaour & Ikram Sereir

In charge of the design of the level, the documentation and the administration of the database.

Software’s system

1.1 MVC

The MVC (model-view-controller), is a data pattern used in JAVA to build efficiently softwares. It has the property ofseparate the program in three categories.

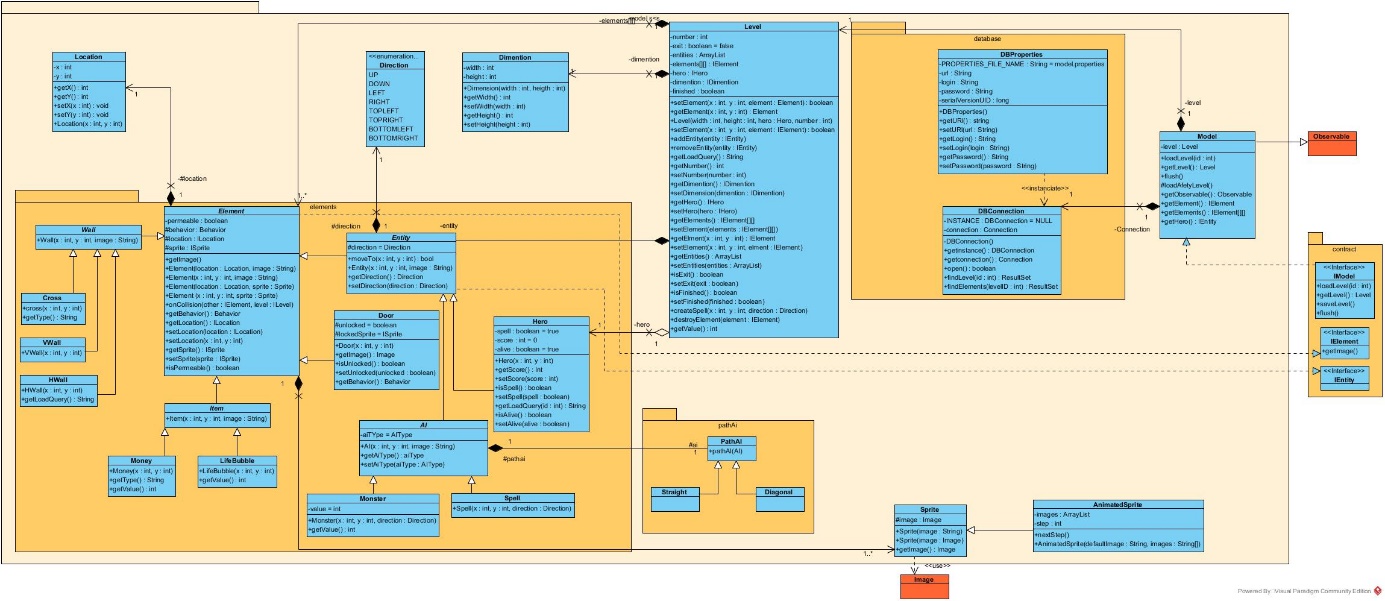
1.1.1 Model

The model part represents the software’s core, it shall treat the data and tell to the view what it should show to the user.

In our case the model has multiple functions:

* To get data from the database
  + DBProperties class
  + DBConnection class
  + Model class
* To build the level to play, set the elements to their location and define their properties
  + Element class and all its derivatives
  + Location class
  + Level class
  + Dimention class
  + Direction enumeration
* To select what sprite will appear on screen
  + Sprite
  + AnimatedSprite

To see a graphic view of the model unit please report to the next page.



1.1.2 View

The view is the visual interface trough which the user shall “see” the program.

It has to show the results of the calculations from the model unit and to “listen” every action from the user that may cause a change in the program (mouse click, key pressed etc. . . )

In our game the view part:

\_ Open a frame to see the game

– GameFrame class

\_ Display graphic elements on the screen

– GamePanel class

\_ Update what the user see, based on the model’s information

– View class

\_ Keyes listener

– GameFrame class

About the model and the view In order to inform the view unit about the model changments, we have to set up a pattern observer. We create an observer interface that will notice the view every possible changes and will make it react,

so the information on screen update in real time.

To see a graphic view of the view unit please report to the next page.

1.1.3 Controller

This part of the program is in charge of the event management. It has to synchronize every action in the software, in

order update the view or the model correctly.

It receives events from the view unit and treat the information in order to trigger the appropriate reaction from the

model part.

In our program the controller has to:

\_ Start the game

– Controller class

\_ Synchronize the software’s actions

– Clock class

\_ Manage the behavior of every element that will appear on screen

– HeroManager class

– MoveManager class

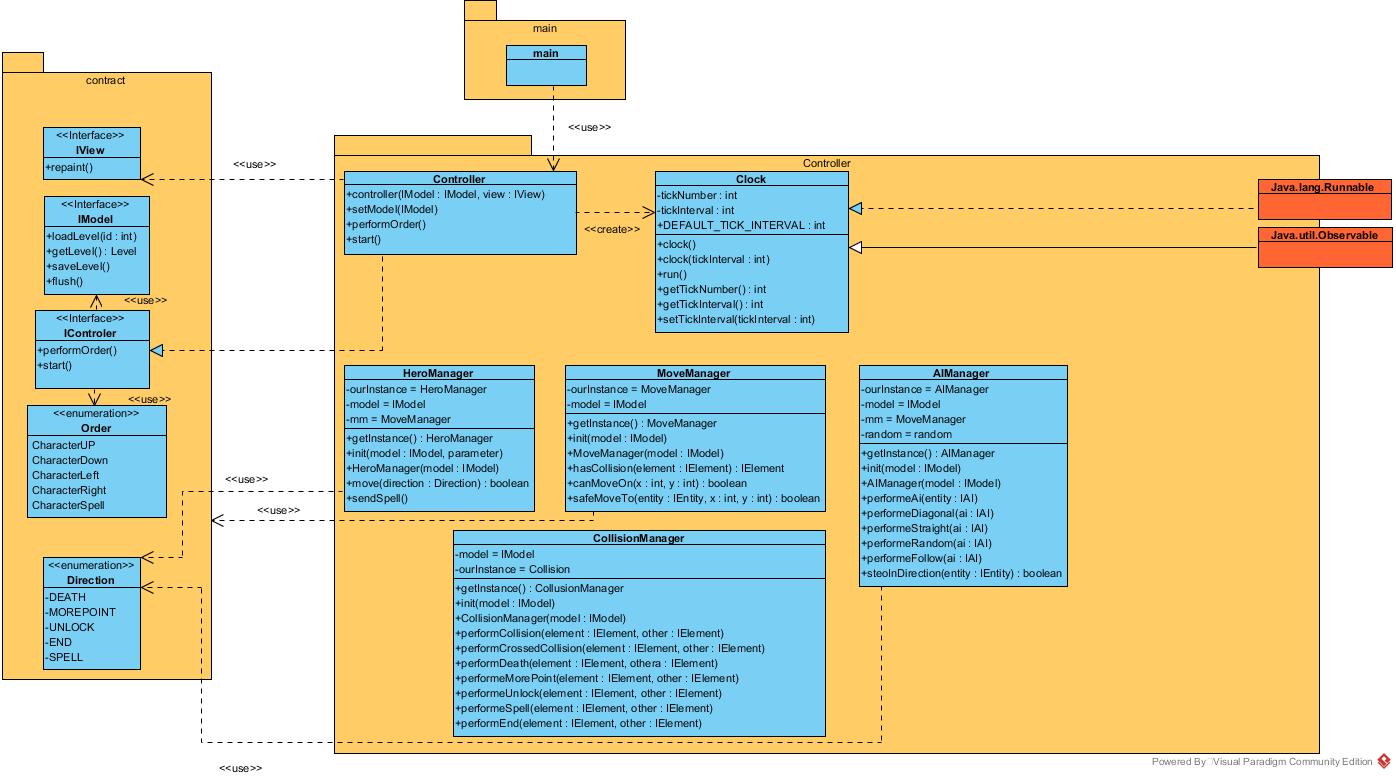
– CollisionManager class

– AIManager class

\_ Inform the model of every order to perform and changes to apply

– Controller class

To see a graphic view of the Controller unit please report to the next page.



1.1.4 Contract

A classic MVC pattern has a default, it creates a large number of couple between the three parts, that may cause program

dysfunction and it decrease the code’s reusability.

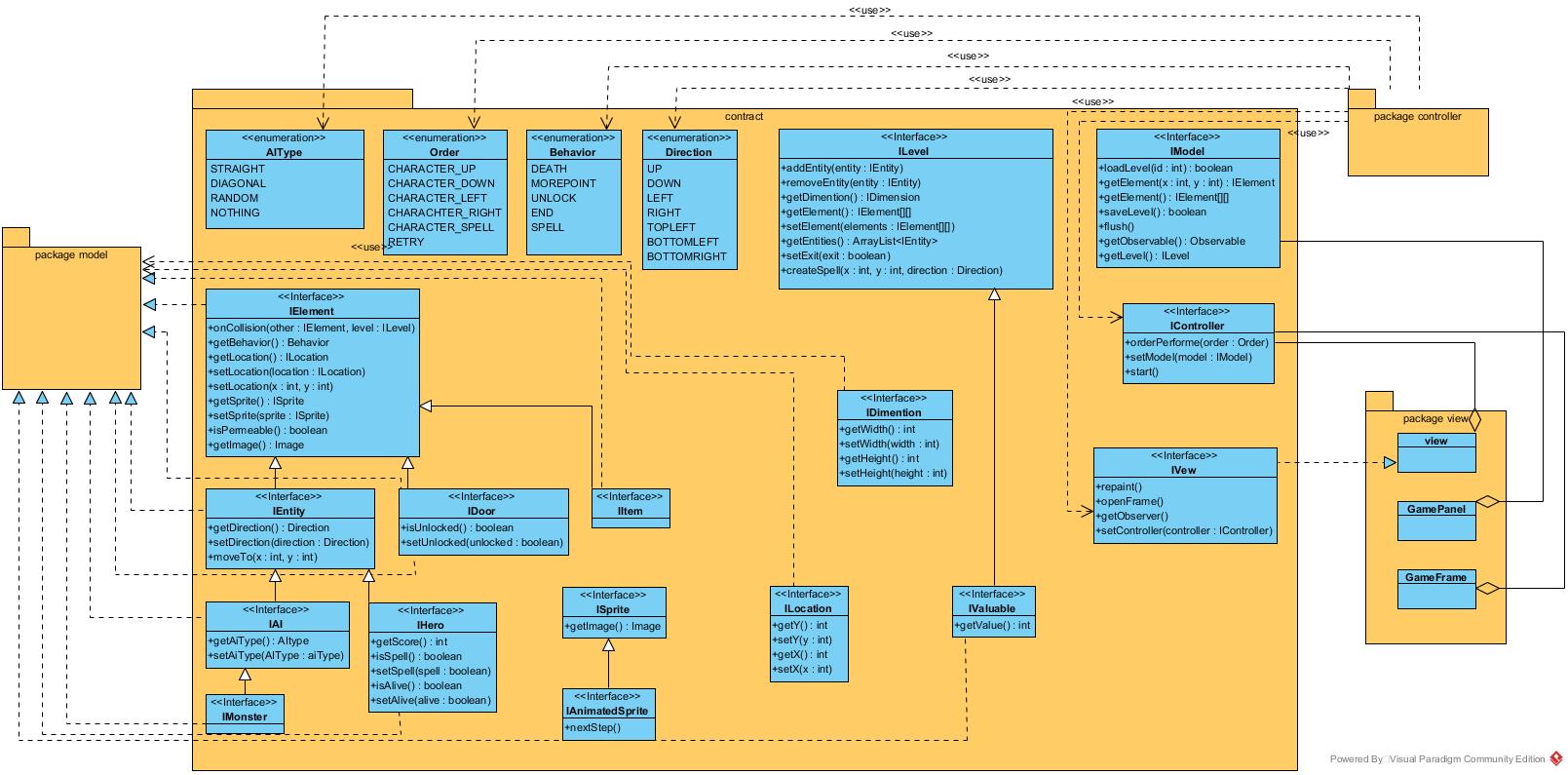
In order to face those problems, wet set a fourth unit in our program, the contract.

In this last part we create many interfaces within which contain operation from every class that has to be use in

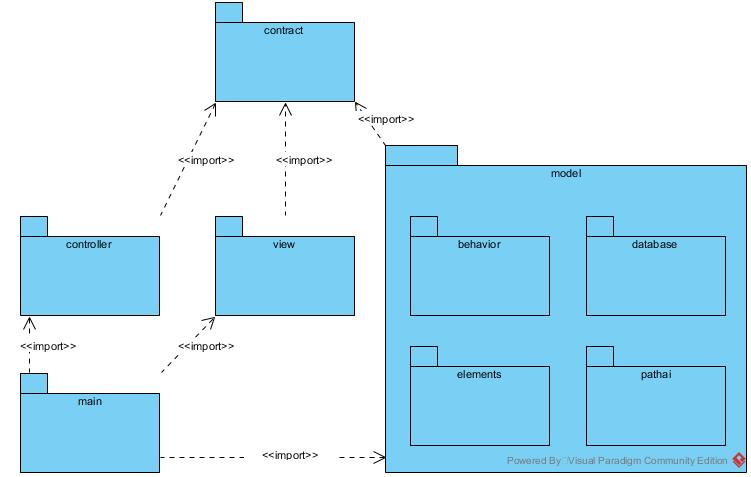
another part of the program than its native unit.

For example, the view needs information from the model, so it will use its interface, IModel.

to see a graphic view of the Contract unit please report to the next page.

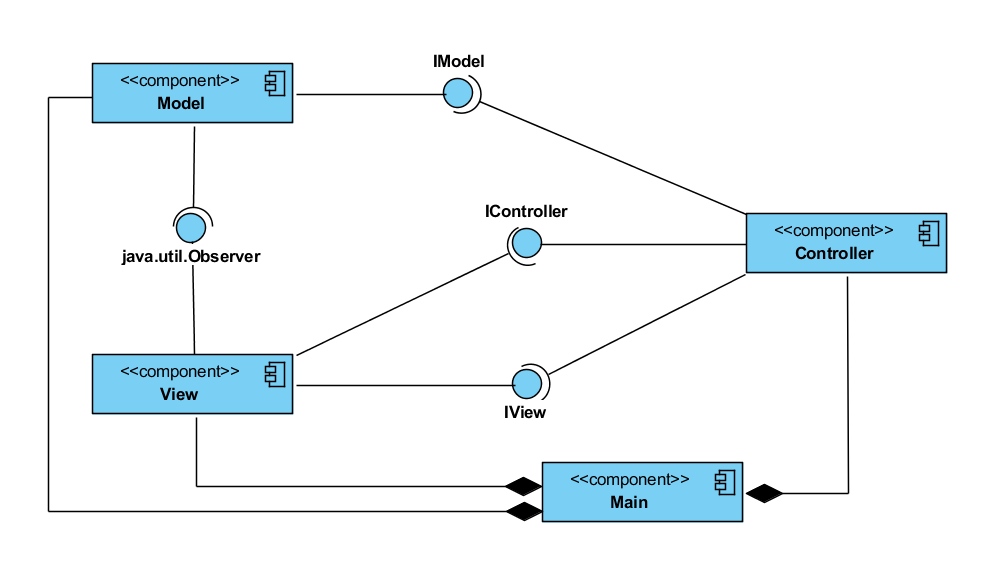


1.1.4 Package



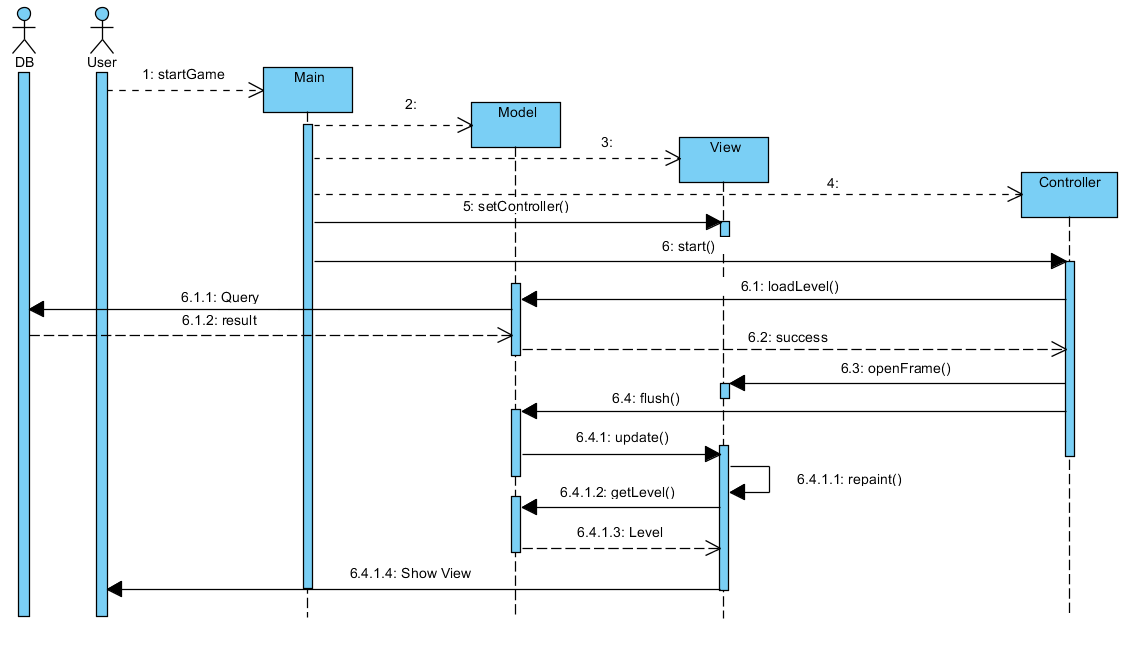
1.2 Components

Finally the program’s components are connecting this way



1.3 Sequence in game

The program is obeing to the following diagram.



First the user starts the game, the main will activate the model, the view then the controller. The controller will ask the model to load the first level, the level is charged from the data base and return to the model. Next the controller will make the view open a frame that will the interface with the user. At every player’s action the view will update by

charging the model’s data.

1.4 Database

1.4.1 Map

To create the maps first we made a text file were each letter was corresponding to an element of the game. Then with a nfilling script we fill the database with all the elements and their location (x,y) on the different levels of the game.

1.5 Stored procedures

The database is composed of 3 stored procedure to create de different levels.

get\_elements\_by\_level return all the elements from a specific level

1 DELIMITER |

2 CREATE PROCEDURE get\_elements\_by\_level (IN lvlID int )

3 BEGIN

4 SELECT element .id , element .type , element .x, element .y, element .

direction , element . lvl\_number FROM element WHERE lvl\_number =

lvlID

5 END |

6 DELIMITER ;

get\_levels return all the levels

1 DELIMITER |

2 CREATE PROCEDURE get\_levels

3 BEGIN

4 SELECT \*

5 FROM lvl

6 ORDER BY lvl. number

7 END |

8 DELIMITER ;

get\_level\_by\_id return a specific level

1 DELIMITER |

2 CREATE PROCEDURE get\_level\_by\_id (IN id int)

3 BEGIN

4 SELECT \* FROM lvl WHERE lvl.id = id LIMIT 1

5 END |

6 DELIMITER ;