CHAPTER 2

Represent the state of the game

2.1 Data representation

2.1.1 Initial design

The goal at this stage of the project is to find the best way to represent all the useful information for the game. Although it may seem easy, this step is essential and has consequences on the whole project, until the last stages. Therefore, it is important to devote considerable energy to it - while keeping in mind that it is never possible to plan everything.

Before starting computer concepts, it is strongly recommended to prepare this work on paper, by listing useful information for the game. It begins with the summary description of these, and then, while the list becomes large, it is relevant to start to make groupings.

This work can be done on a paper sheet if you are one or two, or on a white-board, if you are three to five. Beyond that, you will need more advanced project management techniques (not described in this book).

2.1.2 Types of information

The types of information required to represent the data are very numerous. Among these, it is often necessary to choose a way to represent the position of an element in the world. In the case of the Pacman game, it boils down to 2D coordinates (x, y). A concept of the relative position may also be needed, as in the Pacman game, where a moving character can be between two boxes. For games whose world is in 3D, a third coordinate is required, for example (x, y, z). To this can be added notions of places, such as level and/or floor in a building. It is also necessary to ask the question of the superposition of the elements. In some cases, elements can have similar coordinates, but in a certain order. For example, in strategy games, units can stack on the same cell, like in Civilization. In these cases, we need a position attribute in the stack.

There are then a large number of properties that can qualify an element or a group of elements. For example, in role-playing games, there are notions of life or mana. These attributes often have minimum and maximum values, which this stage of design must define. It is also necessary to specify if it is possible to have duplicates of properties and/or elements: this will have an important impact on the computer design that follows. There can also be notions of the lifetime of elements: these are most often represented by a time counter, like the “super” mode time for Pacman.

2.1.3 Example with the game Pacman

This section provides an example of a description of the elements of the Pacman game, in a simplified version.

A set of static elements (the world) and a set of moving elements (Pacman and ghosts) forms the game state. All elements have the following properties:

- Type of element
- Coordinates (x, y) in the grid

2.1.3.1 Static elements

A grid of elements called “boxes” or “cells” shapes the world. The size of this grid is set at the start of the level. The types of cells are:

Wall cells. They are impassable elements for moving elements. The possible textures are:

- Upper left corner, upper right corner, lower left corner, lower right corner
- Horizontal, vertical

The choice of the texture is purely aesthetic and does not influence the evolution of the game.

Space cells. The moving elements can cross space cells. The space types are:

- The empty spaces
- The gum spaces
- The super gum spaces
- The cemetery spaces, which serve to define the places where ghosts appear at the beginning of the game, but also places where those devoured by Pacman can resume a normal form.
- The start spaces, which define a possible initial position for Pacman.

2.1.3.2 Mobile elements

Moving elements have a direction (none, left, right, up or down), a speed, and a position. A position at zero means that the element is exactly on the cell. For the other values, it means that it is between two cells (the current one and the one defined by the direction of the element). When the position is equal to the speed, then the element moves to the next cell. Thus, the greater the numerical value of speed, the more the character will have a slow movement. Thanks to these mechanics, character movements are always synchronized with a global clock.

Pacman moving element. The player controls this element, thanks to the direction property. Pacman also has a “super counter”, which is used to determine the remaining time before returning to a normal state. Finally, we use a property that we will name “status”, which can take the following values:

- “Normal” status: the most common case where Pacman can move around the maze, and avoid ghosts
- “Super” status: Pacman can eat ghosts
- “Dead” status: A ghost caught Pacman

Ghost moving elements. The direction property also controls these elements, whether it comes from a human or an AI. These elements have two particular properties. The first one is “color”, which is purely aesthetic. The only rule regarding this color is that it is unique for each ghost. The second particular property is the “status”, which can take the following values:

- “Normal” status: the most common case where the ghost can kill Pacman.
- “Flee” status: where the ghost can be killed by “super” Pacman.
- “Eyes” status: where the ghost has been devoured by “super” Pacman.

⇒ Note: This example is intentionally incomplete: it is rare to imagine every possible case from conception. These definitions must be enhanced during the project.

2.1.4 Video Game Development: Specifications

It is now time to start the design of the game you have chosen:

→ Put on the paper the list of items you will need to represent a state of your game. Note that it is not necessary to define the rules of the game for the moment - even if you can start to think about it!

2.2 Basic information

Once the information is on the paper, one can begin to think about how to represent it from a computer point of view. At this stage, there is no programming yet, only design work on the structure of the representation.

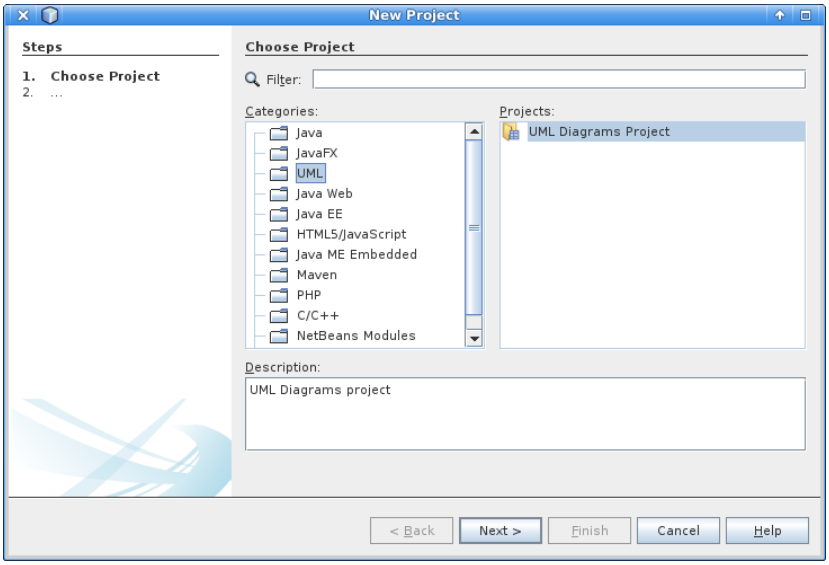
2.2.1 Classes

Each application has a specific data structure, and it is not possible to offer a single recipe for all cases. However, we can highlight several common tools. The first of these tools is the representation of basic information, such as the users of a system or the different types of characters in a game. In this scope, the naive approach proposes to make representations independent of each other. With this approach, classes can represent one or more types of data.

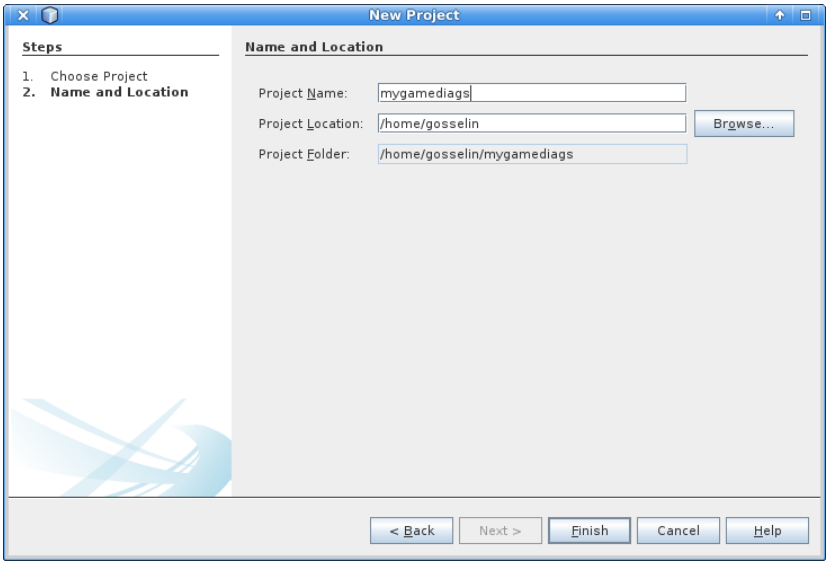
⇒ Note: There are several ways to design these classes. The first and most natural is to draw them on a paper sheet. This old support may seem outdated when discovering software design; it is not the case! It remains infinitely faster to scribble a few classes and their relationships on paper, rather than consider alternatives. Do not hesitate to use it, and even more when working in a group. Once your essay begins to converge, it becomes interesting to use UML software. These allow you to represent your classes in a very clean way, but also to generate some of the corresponding source code.

2.2.1.1 Create a new class diagram with Netbeans / EasyUML

→ Click on the menu **File - New project . . .**, then choose the category **UML** and the type of project **UML Diagrams Project**:



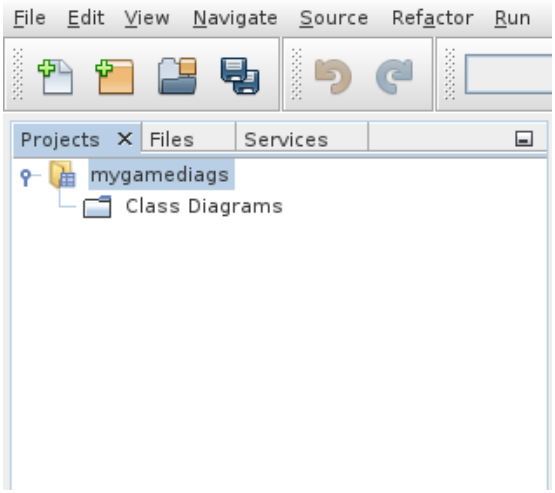
→ Click the **Next** button, and then enter the project name in the **Project Name** box and its folder in the **Project Location** box:



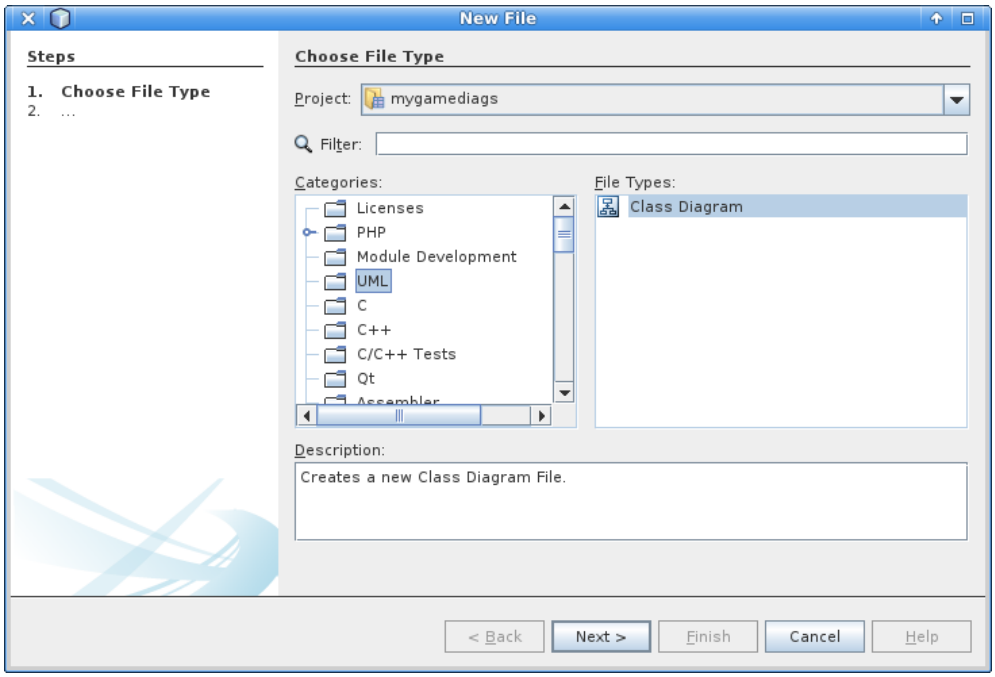
→ Click the **Finish** button.

After the UML project creation, it is possible to add class diagrams.

→ Right-click the **Class Diagrams** folder in the UML project. → This brings up the context menu: choose **New - Other . . .** :

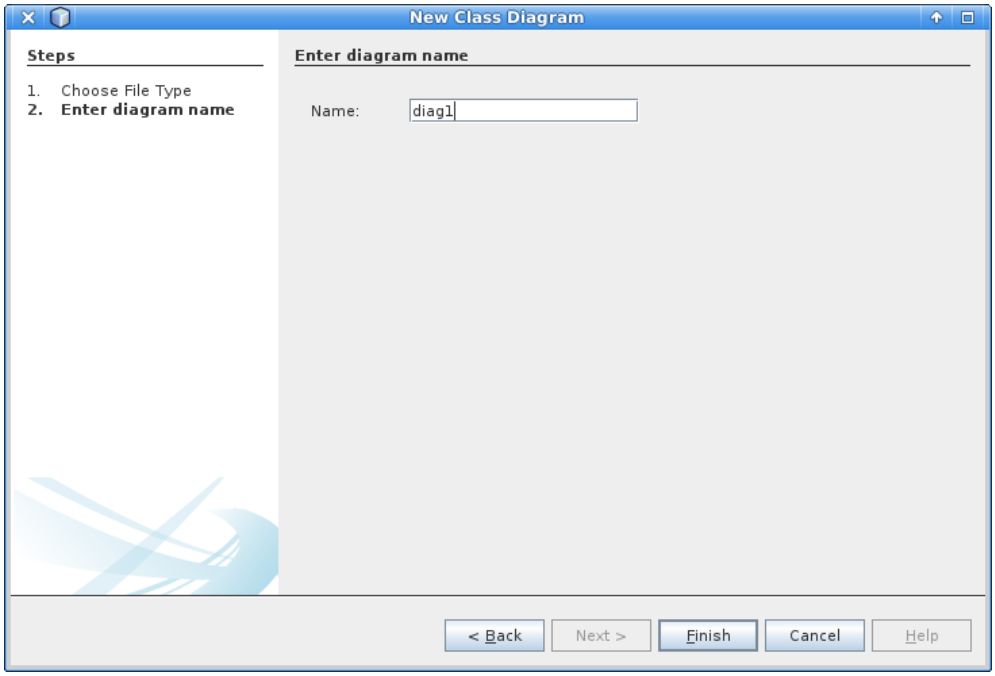


→ Choose the **UML** category and the **Class Diagram** file type:



→ Click the **Next** button

→ Define the diagram name in the **Name** box:

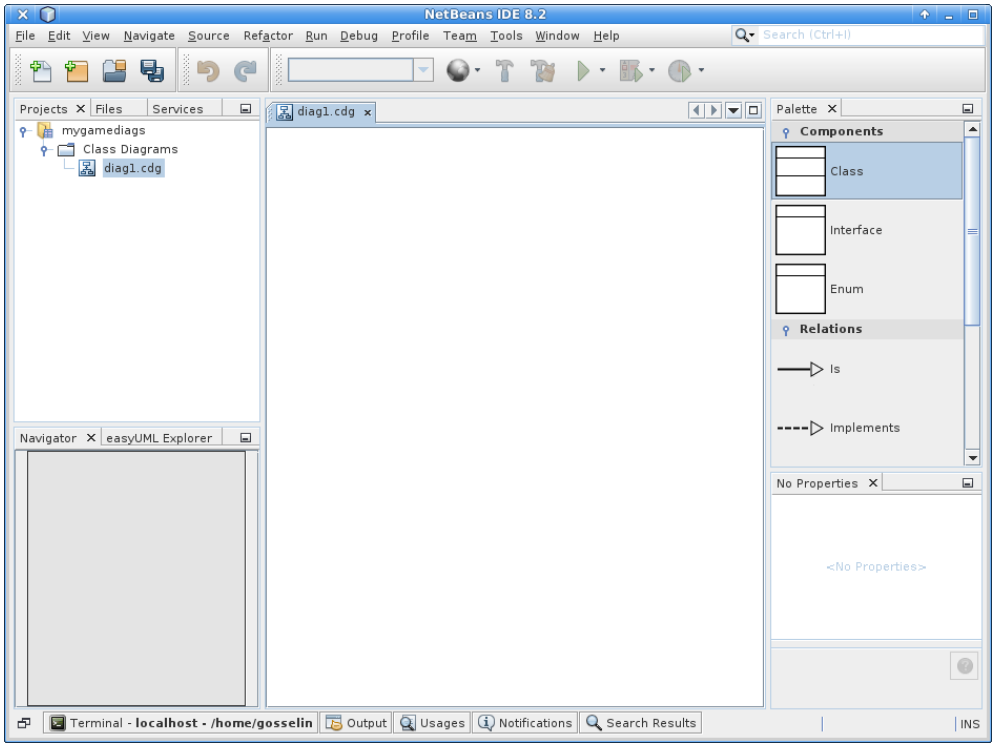


→ Click **Finish**

→ Open the class diagram by double-clicking on its file in the **Project** window.

It brings up several windows:

- The central window is the diagram window (empty in this example).
- The window at the top right is the tools palette to form the diagram.
- The window at the bottom right contains the properties of the currently selected element (if any, none in this example).
- The window at the bottom left provides an overview of the diagram.



2.2.1.2 Adding a class

Creating classes with EasyUML is very simple, for example, to create a `Pacman` class with two integer attributes x and y:

→ Click the **Class** tool from the palette to the diagram;

→ Double-click on the title of the new class, then enter “Pacman”;

→ Double-click in the box under the class title (you should see **double-click to add field**), then enter “int x”;