INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

TPCPOO – Diaballik
Planning, Deliverables, and Tips

4INFO – Computer Science Department

Éric Anquetil
eric.anquetil@irisa.fr
Arnaud Blouin
arnaud.blouin@irisa.fr
Pascal Fresnay
pascal.fresnay@excense.fr
Romain.Lagneau
Romain.Lagneau@insa-rennes.fr
Gwendal Le Moulec
gwendal.le-moulec@irisa.fr
Grégoire Richard
gregoire.richard@excense.fr

Contents

1 Project Team

You have to follow these rules to create your project team:

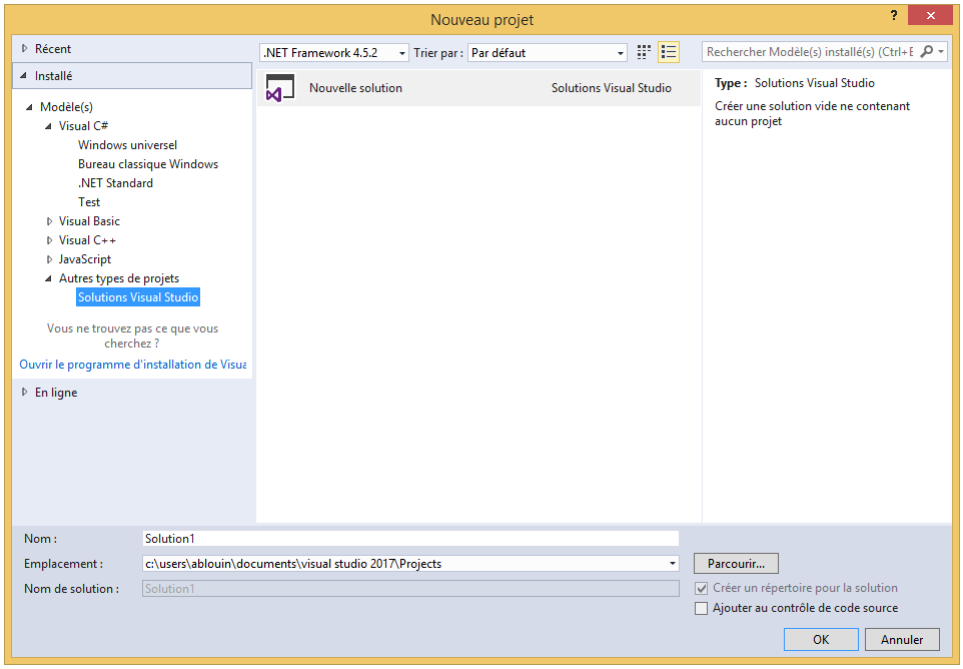
- **Two students per project team**. If your group has an odd number of students, one team can be composed of three students. For this group, the evaluation will be more strict.
- **A project team must have at max: one student that repeats the year ("*redoublant*") or a new comer ("*nouvel arrivant*")**. So, it is not possible to have a team composed of two "*redoublants*", or of one "*redoublant*" and one new-comer, or of two new comers.

2 Modelling (practical sessions #1 and #2)

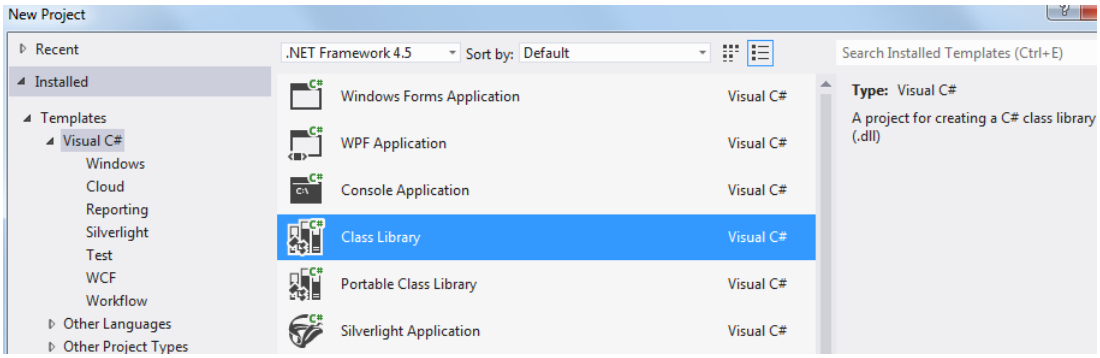
At the beginning of the project you have to **create a project on the Gitlab instance of the INSA de Rennes** (`https://gitlab.insa-rennes.fr/users/sign_in`). **Your project must be private.** You must **add all the supervisors as collaborators with 'MASTER' rights** of your private project (at least: *arnaud-blouin*).
Using Git, take care about conflicts during commits and polls: try to work on files that your colleague is not going to change; do atomic commits (small ones, one commit = one task); avoid the commit of configuration files (do that in concert with your colleague).
To merge upstream changes:

```
git fetch
git merge
```

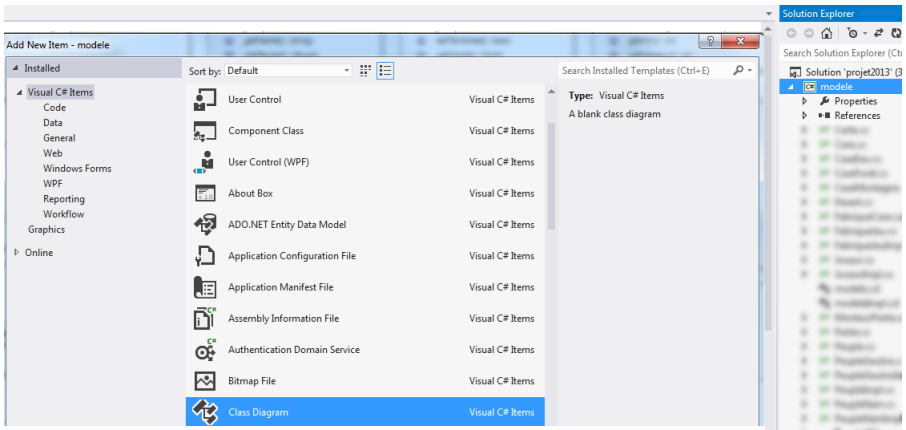
You will use Visual Studio 2017 Enterprise. The first thing to do is to create a "Solution" (i.e. a Visual Studio project):



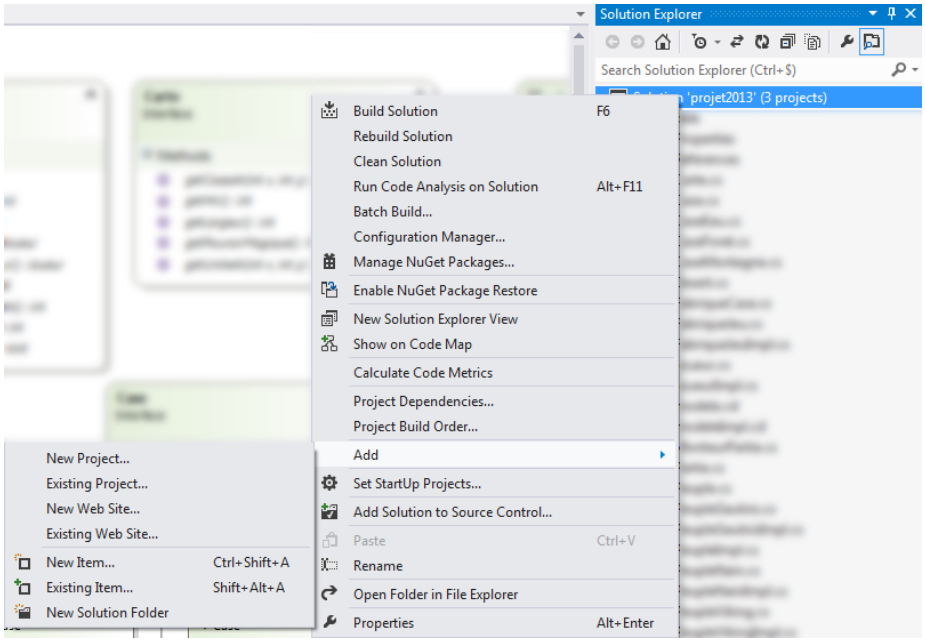
You will start the project by modelling the game. Note that the UML module is not more provided since Visual Studio 2017. You can still create a C# class diagram (that is not a UML class diagram), but for the other diagrams to design, use Papyrus (installed at the department on Linux): `https://eclipse.org/papyrus/`.
The easiest way to create class diagrams in the context of this project is to create a "Class Library" project as depicted by the following picture:



In this newly created project, you can add a class diagram:



The C# Class Diagram does not follow the UML syntax and supports some C# features (such as properties[1]). Thanks to the C# class diagram you will create, the corresponding code is automatically generated. Moreover, modifying the code modifies the class diagram. This class diagram does not permit the definition of packages (namespaces in C#).
A solution can contain several projects (class library, C++ library, WPF project, test project, etc.). To add a new project to a solution:



2.1 Description of the Modelling Phase

You have to design UML models that will ease the understanding of the structure of your game and of the development phase that will follow. The required UML models are:

- Class diagrams that will describe:
  - The structure and the organisation of the data model of the game (board, player, game, *etc.*);
  - The required design patterns:
    - *Builder*
    - *Strategy*
    - *Command*
    - *Memento*

    You have to find why these design patterns are useful. Each design pattern can be used several times.
- One UML statechart diagram to model the behaviour of different objects. For instance, you can model the behaviour of a game.
- Two UML interaction diagrams to help you in defining class diagrams (*e.g.* board initialisation, behavior of a turn).
- A mock-up of the user interfaces (UIs) you may develop. A mock-up is just an initial design of the GUIs of the application. Some tools are enumerated here: `http://graphicdesign.stackexchange.com/questions/37067/best-tool-to-create-a-mock-gui-quickly`.

[1] `https://programmers.stackexchange.com/questions/250914/why-does-c-allow-properties-in-interfaces`

Any additional and correct UML diagrams will be considered during the evaluation of the project.

2.2 Help

- Starting the modelling phase of a project is always a tedious task: "how I start?" "What should I model?" *etc.* We suggest you to start modelling with a pencil and paper to brainstorm more efficiently.
- Sequence diagrams help in identifying classes, attributes, and associations of class diagrams. Think about designing sequence and class diagrams together.
- You can use classes from .Net library as the type of attributes or operations. To do so, just write in the field *type* the full name of the class. For instance, to use the class *Color*, you need to write in the field *type System.Drawing.Color*. You can find the full name of .Net classes on the Web (in the C# documentation).
- Do not forget to model class constructors and their parameters.
- As introduced in the OOP class with WPF examples, the sychronisation between the model and the view is done in WPF using data binding. Some documentations about data binding in WPF:
  `https://msdn.microsoft.com/en-us/library/ms752039(v=vs.110).aspx`
  `https://msdn.microsoft.com/en-us/library/ms743643(v=vs.110).aspx`
- Getters and setters work differently than in Java. You should use *properties*. Please read:
  `http://msdn.microsoft.com/en-us/library/w86s7x04.aspx`
  `http://msdn.microsoft.com/en-us/library/64syzecx.aspx`
  for more information.

3 Implementation (from the practical session #3)

3.1 Description

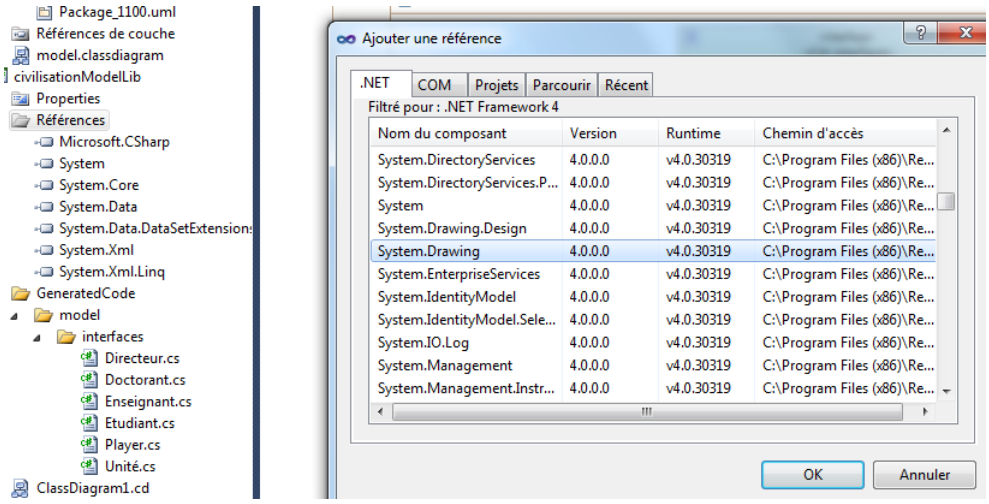
Your game will be developed in C# and C++: C++ for several algorithms; C# for the rest of the game. Keep in mind that a part of this code is generated from your class diagrams. You have to complete the generated code.

3.2 Tasks to do

During these practical sessions, you will complete the code generated from the class diagrams (*i.e.* implement the operations).

3.3 Help

- Do not forget to compile the code regularly to fix issues.
- You will probably find modelling issues. That's normal. In such cases, update the concerned UML diagrams.
- With Visual Studio, using a type that comes from a .net library requires to add this last as a reference to the project. To do so, go in the solution panel, right-click on the folder References of your project, select "add a reference", and go to the tab *.NET* is depicted by the following picture *System.Drawing* to use the class *Color*, *WindowsBase* to use *Point*, *etc.*):



- Executing a WPF application requires to tag the main project WPF as the project of the solution to execute: right-click on this project and on "Define as start-up project".

4 Software Testing (from practical session #3)

It is mandatory to test the different parts of a software system all along the coding phase. In our case, you have to test your C++ libraries and C# code. To simplify the testing process, you will test your C++ code through C# tests.

Coding C# tests. You have to add a test project to your solution ("Unit test project"). Then, go to the operation you want to test and select it. Right-click on it and click on *create unitary tests*. Following, select the test project. Here is a Microsoft tutorial on the definition of tests in Visual Studio:

`https://msdn.microsoft.com/en-us/library/ms182532(v=vs.140).aspx`

The goal of each test operation is to check a small amount of properties in your code using assertions:

`http://msdn.microsoft.com/fr-fr/library/ms182532(v=vs.80).aspx`

Each requirement of the requirements document must have at least one test (when possible). The name of these tests must explicitly use the requirement name.

You can also use the different UML diagrams your create to produce tests (*e.g.* sequence diagrams that describe algorithms). That will permit the checking of your specifications against your implementation. To execute tests, go to the file that contains the tests, right-click in the text editor, select "execute tests".

4.1 Help

- Do not forget that if sequence diagrams and friends are not very used to generate code, they are very useful to define tests.
- If you face compilation problems with your tests, maybe these last are located on the network. You must either move them on your computer or configure *Visual Studio* (cf. `http://coursonline.insa-rennes.fr/mod/forum/discuss.php?d=399`).

5 Development of the C++ library (practical session #5)

Algorithms must be developed in C++. The goal is to train in developing libraries (*so*, *lib*, or *dll* files) and to use them in applications. These algorithms will be used in your C# application

as a library. We provide you a C++ project, that will contain the algorithms, and a wrapper C# file you can find on Moodle. You have to import them in your Visual Studio solution, adapt/modify/implement them to work within your existing projects. An example of a C++ projet and its integration within other projects are available on Moodle in the resources of the project (file *CppLibProjetEtudiant.zip*).

5.1 Task to do

Develop a C++ library containing the following algorithms:

1. The starting IA algorithm. This algorithm will take as inputs all the required information (board, pieces, balls) and compute as output one to three moves.
2. Moves suggestion. All the possible moves that a human player can do when selecting a piece or a ball.

5.2 Help

1. Visual Studio does not really appreciate when you use files (*e.g.* dll files) stored on the network. Check that when you observe strange compilation behaviour.
2. If you have linkage problem with your C++ library during the compilation in the mode "release" of your project, check the following solution: go to the properties of the wrapper project (mode *release*) -> Linkage editing -> enter -> Additional dependencies -> add the path to the *.lib* file (in quotes).
3. You cannot pass C# objects to the DLL. You must transform them as primitive types (or arrays of primitive types).

6 Graphical and Interactive Features (practical sessions #6, #7, and #8)

The graphical interface is composed of a view that displays the game board, a panel containing information on the current game or the select piece/ball, and a button to end the current turn.

6.1 Tasks to do

Try to progress step by step:

- Create a project WPF application (if not already done) that will use the C++ and C# libraries.
- Create a configuration page to initialise the game.
- When a game is create, display the board. The board is too large to be entirely visible. Scrollers should be used in this case.
- Display pieces and balls.
- Be able to select a piece or a ball by clicking on it.
- Create a panel to see the characteristics of the selected piece/ball and information about the current game.
- Be able to move a piece/ball using the keyboard or the mouse. You should use the numeric pad to move units (*Key.NumPadX*).

- The end of a turn can be done by clicking on the button "end of turn" or by pressing the key "enter".
- Save and load a game. This feature will be very useful during your final demonstration.
- Replay.

6.2 Help

- You can use the conceot of *page* to define different views and navigation through them:
  `https://msdn.microsoft.com/en-us/library/ms750478(v=vs.110).aspx`
- You can use Snoop to analyse, debug, modify at run time WPF GUIs:
  `https://snoopwpf.codeplex.com/`
- You may have to use the concept of *event*:
  `https://msdn.microsoft.com/en-us/library/ms753115(v=vs.110).aspx`
  `https://msdn.microsoft.com/en-us/library/ms742806(v=vs.110).aspx`
- To let data notify their observers about a change (observer pattern), you should use the INotifyPropertyChanged interface.
  `https://msdn.microsoft.com/en-us/library/system.componentmodel.inotifypropertychanged(v=vs.110).aspx`
  `http://grenangen.se/node/75`
- You should use the concept of *UniformGrid* to layout the map:
  `https://msdn.microsoft.com/en-us/library/system.windows.controls.primitives.uniformgrid(v=vs.110).aspx`
- You may use the concept of *command* to associate to widgets:
  `https://msdn.microsoft.com/en-us/library/ms752308(v=vs.110).aspx`

7 Deliverables

7.1 D#1: Modelling report

You have to return a modelling report **for November the 10th before 11.59999PM on the Moodle platform**. Since some diagrams may be very large, please attach with the report the pictures of these diagrams in a vectorial format (PDF, SVG).

7.2 D#2: Model/lib/wrapper implementation and tests

You have to return an archive containing your current implementation of the model of the application (the C# code generated from the class diagrams you implemented), of the wrapper and the C++ library **for December the 7th before 11.59999PM on the Moodle platform**. A serious test suite is also required for this deliverable.

7.3 D#3: Final application

You must return the project on Moodle **for January the 10th 11.599PM** with the following elements:

1. The user documentation of the game (how to use it);

2. The source code. The features expected at the end of the project are identified in the sections "tasks to do".
3. An executable file that works!

7.4 D#4: Project presentation

You must also prepare a defence of 8 minutes composed of a short presentation (to highlight the features you implemented or not) and a demonstration. The defense is more a client talk than a technical talk. Do not re-explain the rules, the goal of the project, *etc*. Defences will occur January the 11th 2017.

8 Marking

The following approximative marks will be used:

1. the modelling report $\approx$ 7 pts ;
2. the user documentation (a user manual of the game that can include a tutorial, rules, etc.), the code (commented, cleaned, tested, *etc*.), and executable files $\approx$ 7 pts ;
3. Defence (presentation + demonstration) $\approx$ 6 pts.