De l'analyse à la conception orientées objet

1- Rappels UML etc. basés sur le cours de Pascal Molli, Université de Nancy, Loria
2- Positionnement des cours suivants

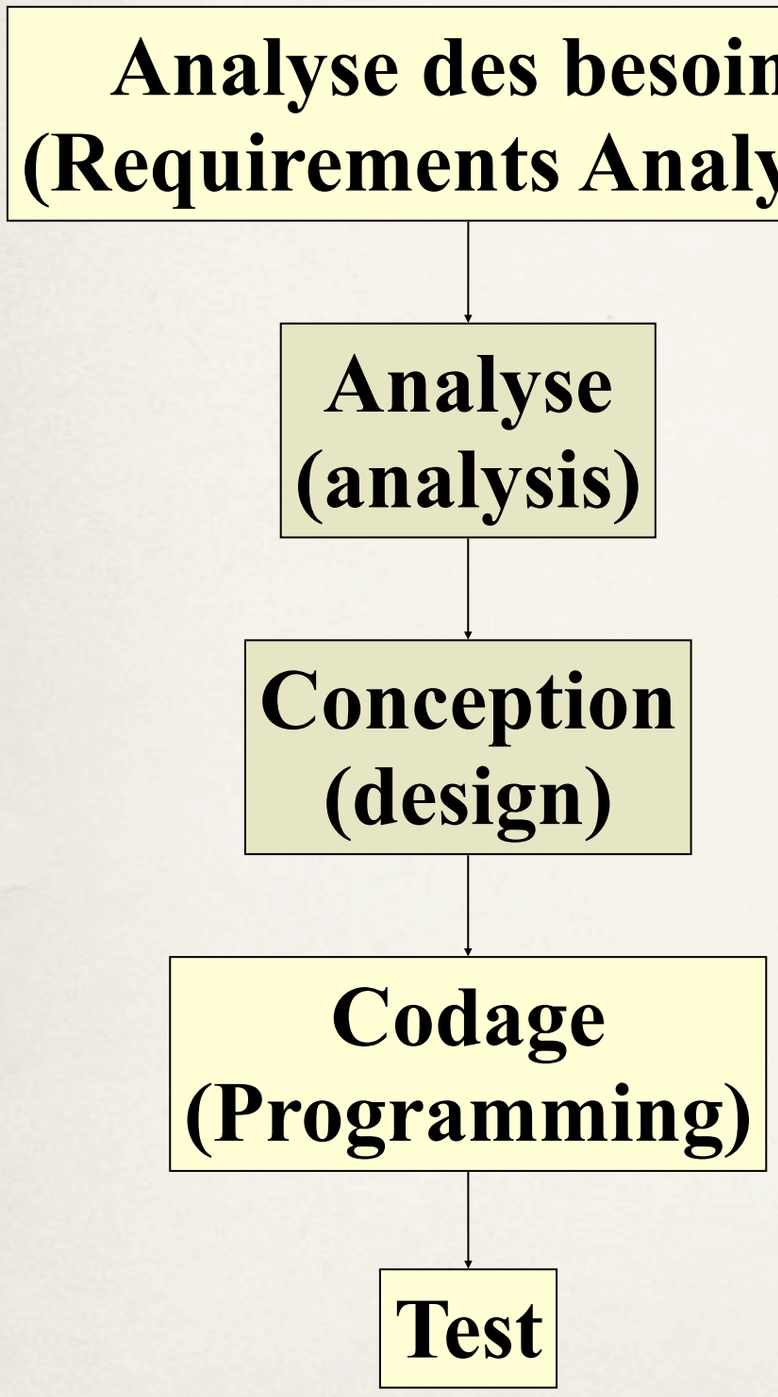
Mireille Blay-Fornarino, Université Nice Sophia Antipolis, Département Info IUT, Septembre 2014

Bibliographie

- Unified Modeling Language; Pascal Molli; Université Nancy1,Loria molli@loria.fr, www.loria.fr/~molli
- Craig Larman, UML2 et les Design Patterns

Introduction et vue d'ensemble

Introduction et vue d'ensemble



- UML : Produire des documents
  - Pertinents
  - Cohérents
  - Compréhensibles
  - Faciles à changer
  - Pour tout le cycle de vie

Pascal Molli, molli@loria.fr

Analyse

- ✓ Indépendante de l'implémentation !
- ✓ Vise à circonscrire l'application
  - ➡ Quelles sont ses *limites*? les *exigences* auxquelles elle doit répondre?
    - ➡ Identification des classes et des relations
    - ➡ Description des collaborations entre les objets des différentes classes

➡ Diagrammes de cas d'utilisation, de classes, de séquences, d'activités.

Conception (design)

✓Prise en compte de l'architecture informatique

✓Classes techniques pour gérer l 'interface graphique, la distribution, la persistance, la concurrence, ...

➡ Diagramme de classes, de séquences, *de composant*, *de déploiement*, *d 'états*

Importance des patterns de conception pour construire des codes «pragmatiques»

Programmation

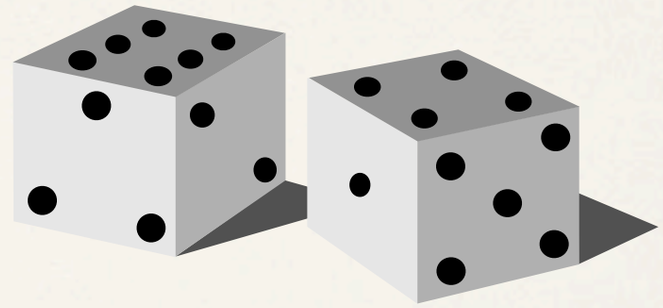
- ✓ Conversion des classes de conception vers les langages cibles: java, sql, c++, IDL
- ✓ Conversion des classes persistantes vers les modèles de persistance (SGBD, BDOO, Langages persistants)
- ✓ etc ...

Tests

- ✓ Tests unitaires: diagrammes de classes
- ✓ Tests d'intégration: diagrammes de séquences, d'activités
- ✓ Test du système: diagramme de cas d'utilisation, de séquences système.

Préparation et Evaluation des tests tout au long du cycle de vie du logiciel

d'après Pascal Molli, molli@loria.fr



Mise en oeuvre sur un exemple

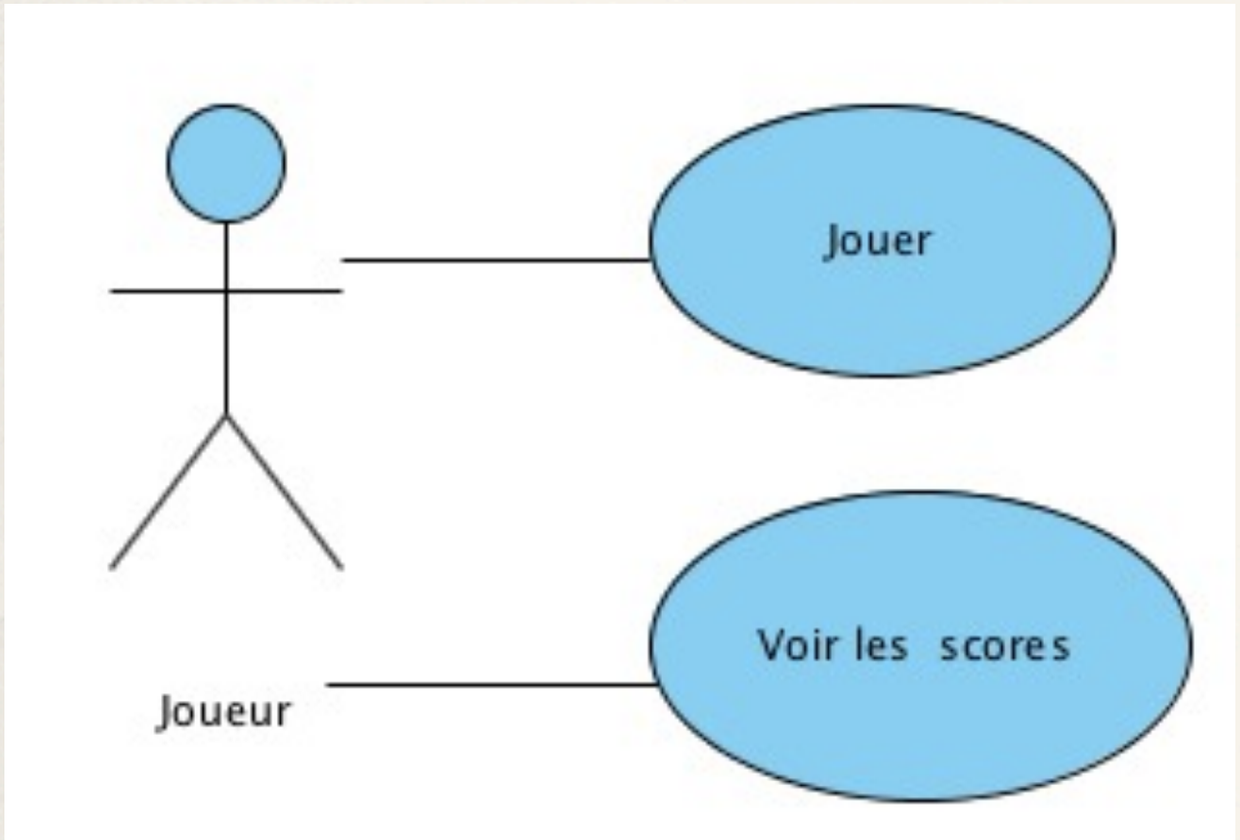
- Un jeu de dés
- Le joueur lance 10 x 2 dés
- Si le total fait 7, il marque 10 points à son score
- En fin de partie, son score est inscrit dans le tableau des scores.

Analyse des besoins

- Identifier les acteurs.
- Identifier les cas d'utilisations possibles du système
  - Ses fonctionnalités externes !

d'après Pascal Molli, molli@loria.fr

Premiers Cas d'utilisation



- **Jouer**:
  - Acteur: Joueur
  - Descr: Le joueur prend 10x les dés, à chaque fois que le total fait 7, +10pts
- **Voir les scores**
  - Acteur: Joueur
  - Descr: Le joueur consulte en read only les scores précédents obtenus par les joueurs

Use Case

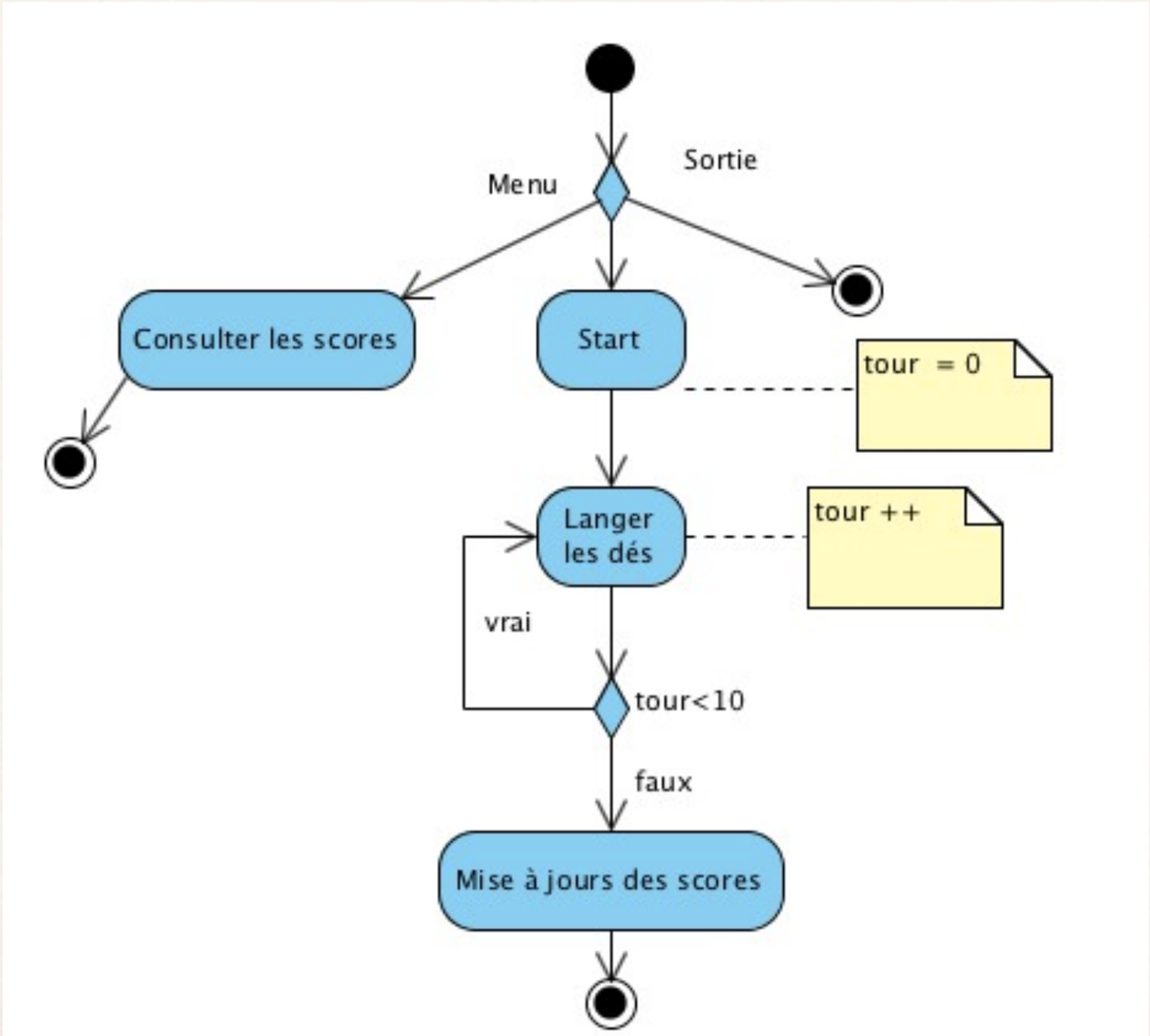
- Diagramme extrêmement important !
- IMHO, Il doit figurer dans un cahier des charges
- Il doit figurer dans toute présentation d'une application !
- IL DOIT ÊTRE COMMENTE de manière rigoureuse !
- Il sert de référence pour toute la suite des opérations.

Diagramme d'activité

- Identifier les activités (en s'appuyant sur les cas d'utilisation)
- Identifier les transitions entre activités et donc entre les cas d'utilisation

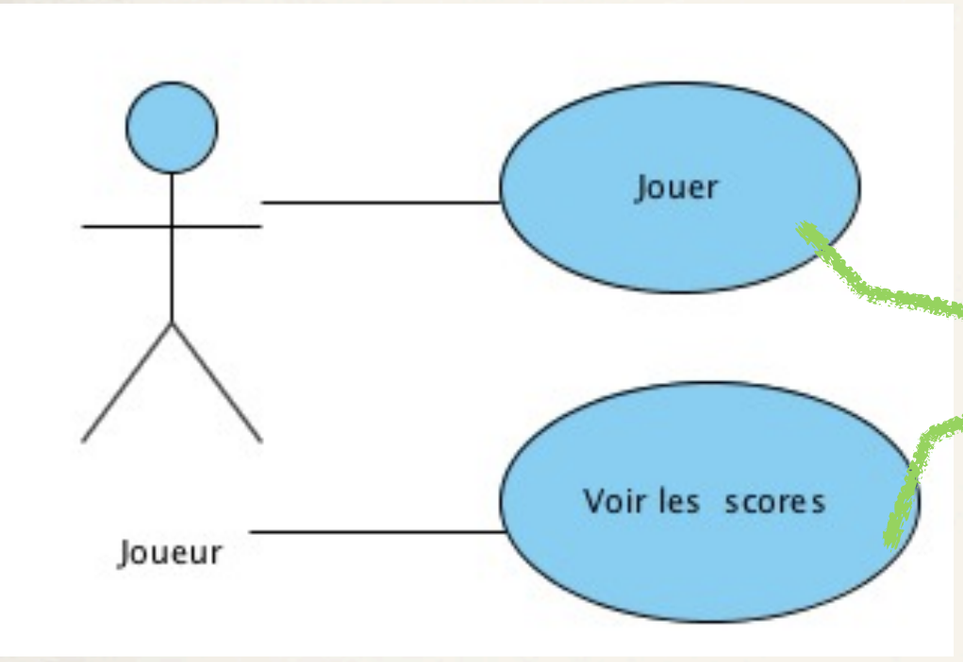
On les verra en détail dans un futur cours !

Diagramme d ’activité



d’après Pascal Molli, molli@loria.fr

Diagramme d ’activité



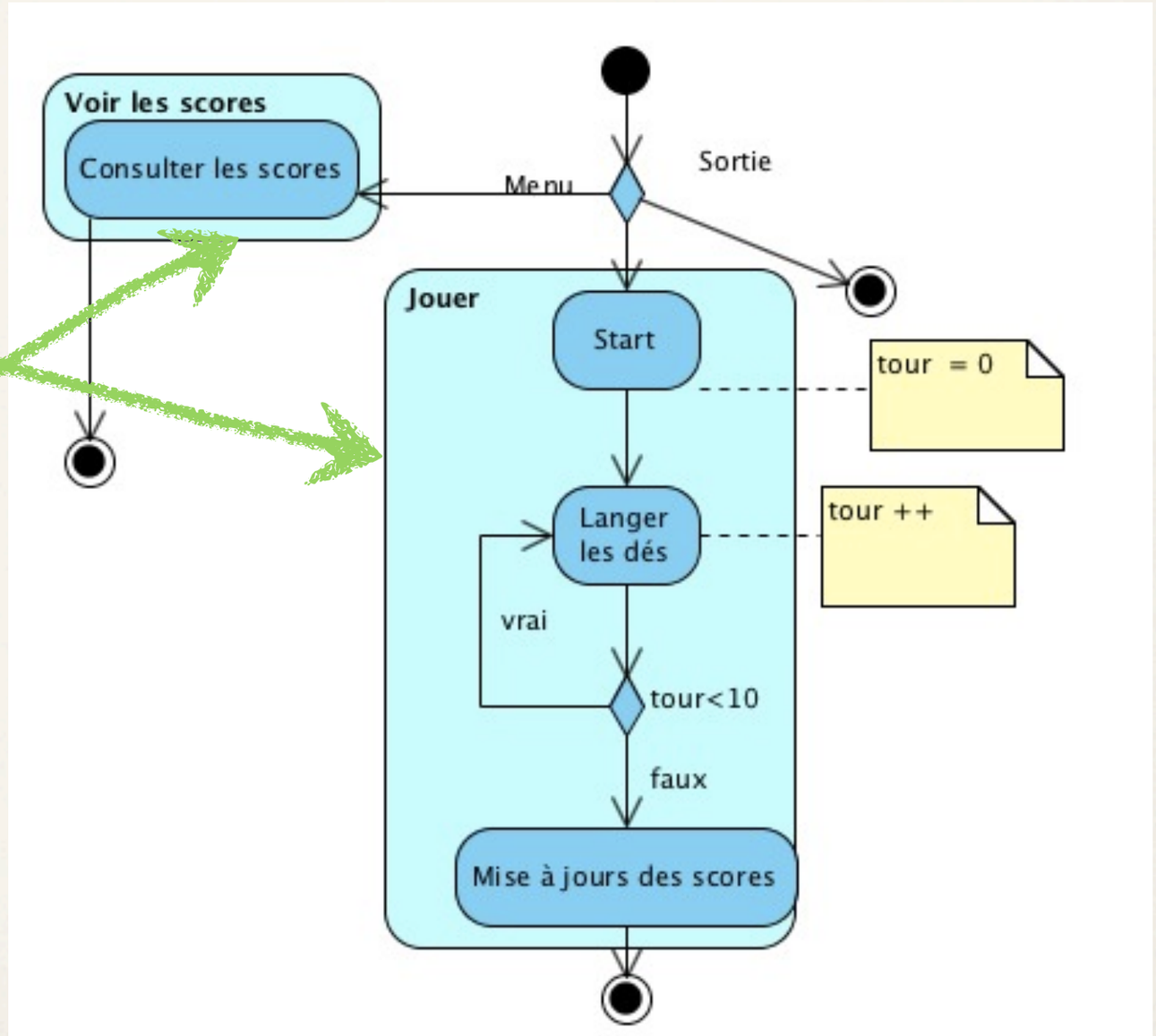
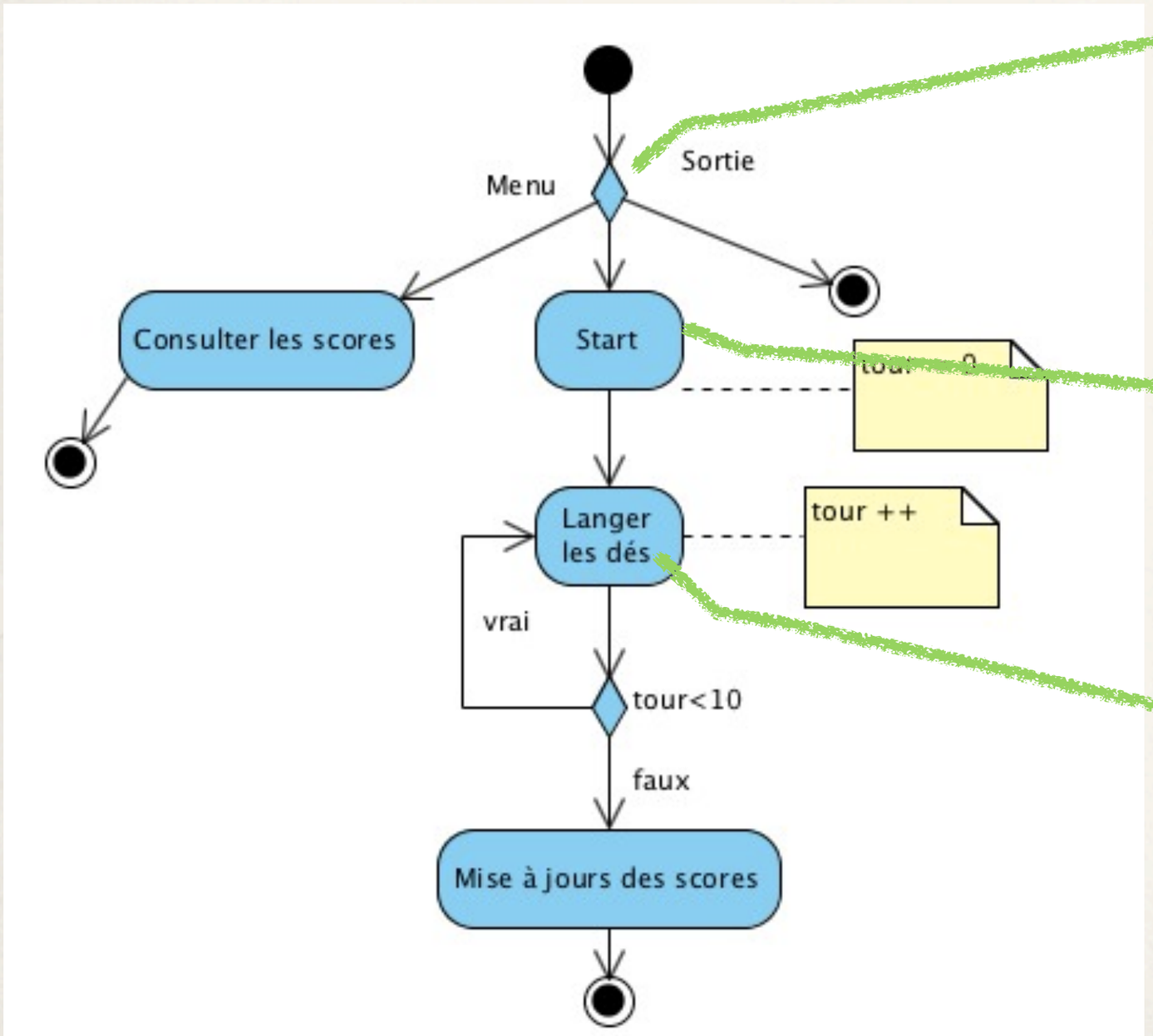
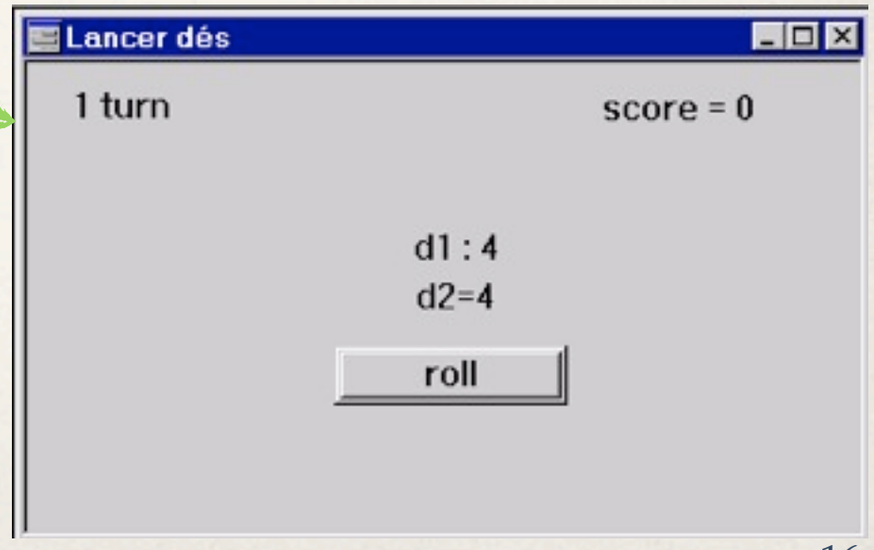


Diagramme d'activité





d'après Pascal Molli, molli@loria.fr

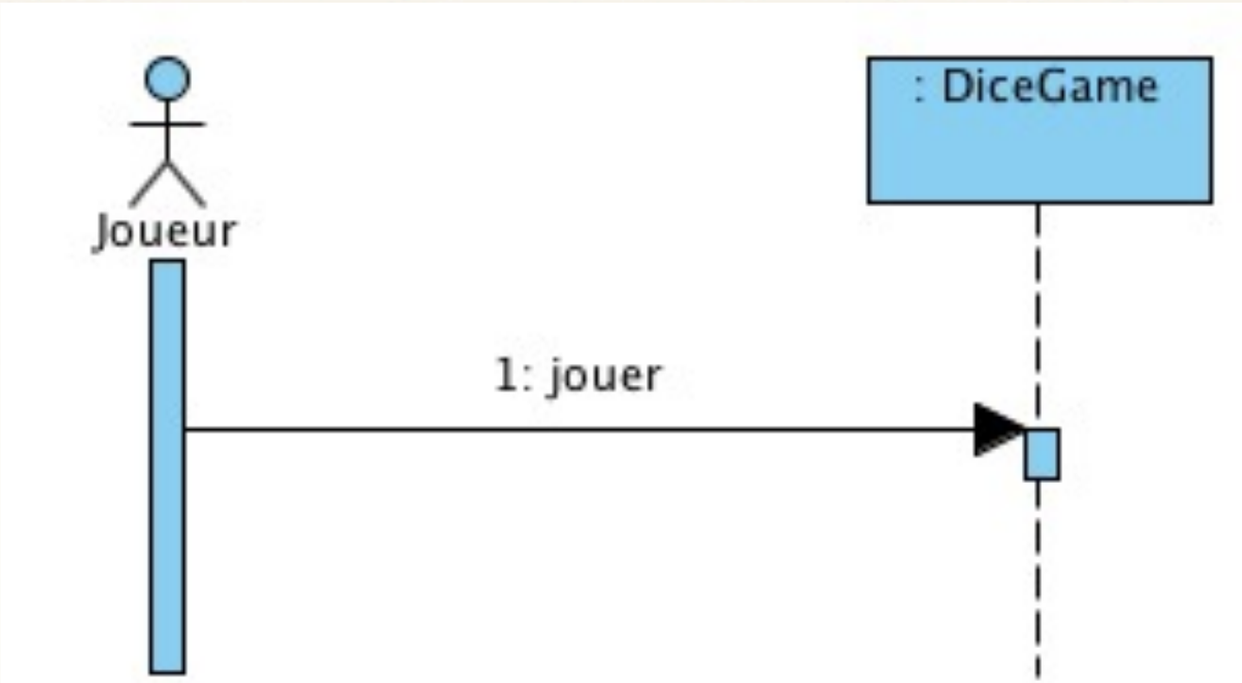
Analyse

- Indépendante de l'implantation
- Déterminer les classes d'objets du domaine étudié et concernées par l'application : premier diagramme de classe
- Modéliser la dynamique du système: diagramme de séquence.

Diagramme de séquences

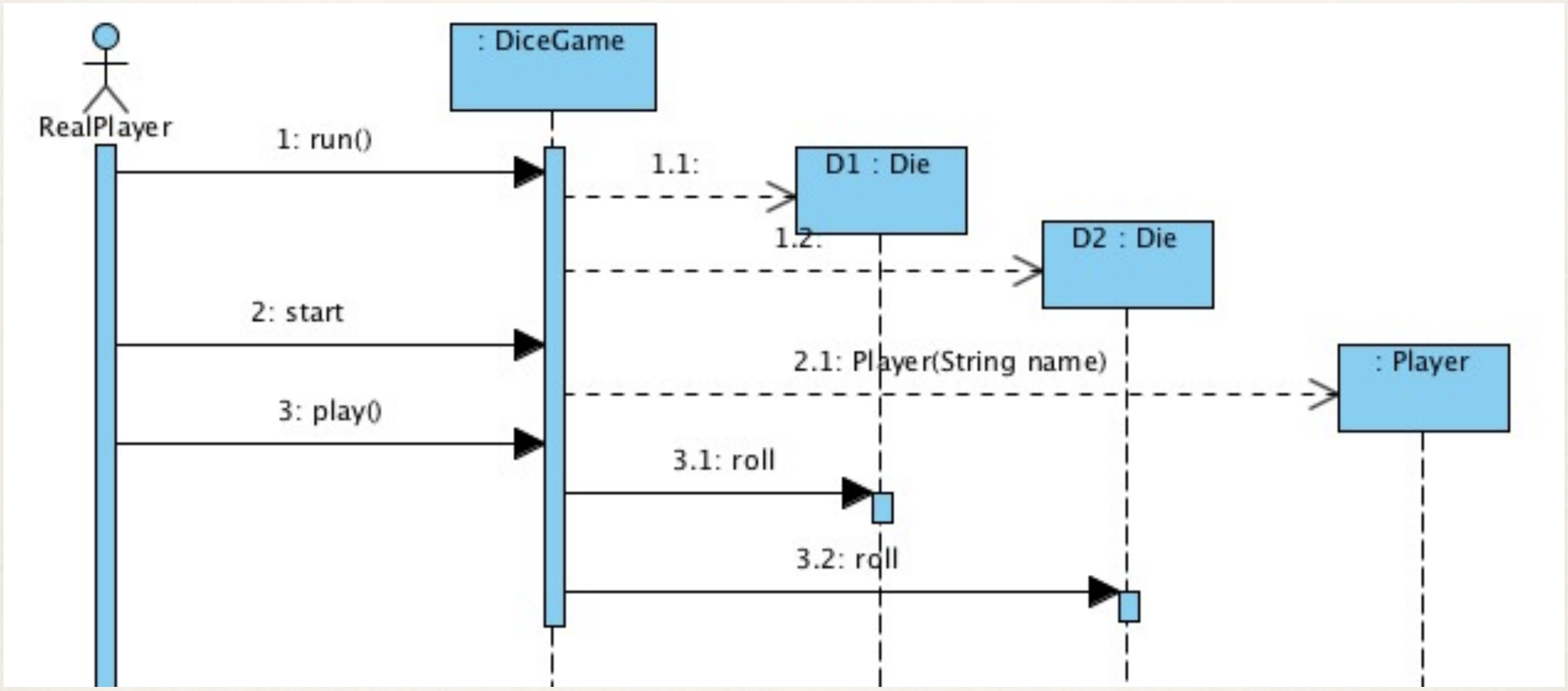
- Modélise la dynamique
- Focalise sur l'enchaînement des messages
- Permet d'identifier les objets, les messages et leur ordonnancement.

Diagramme de séquences



Pascal Molli, molli@loria.fr

Diagramme de séquences



Objets ?
Relations ?
Quand le «Player» est-il créé?

Diagramme de classes

- Identifier les classes
- Identifier les relations statiques et dynamiques entre les classes
- Déterminer les cardinalités des relations
- Déterminer les attributs des classes
- Déterminer les méthodes et leurs paramètres

Pascal Molli, molli@loria.fr

Diagramme de classes

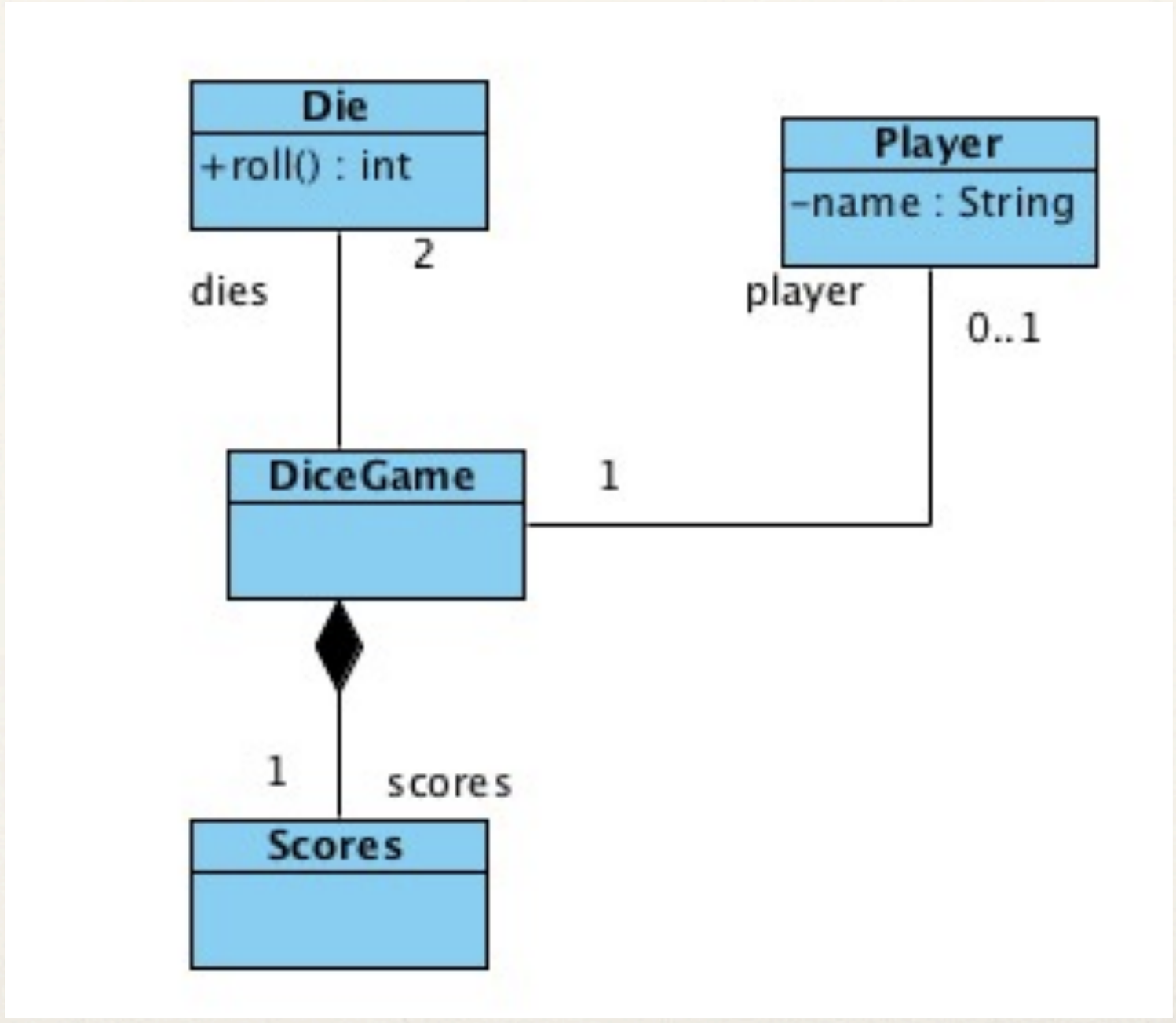


Diagramme de classes contre le diagramme de séquence

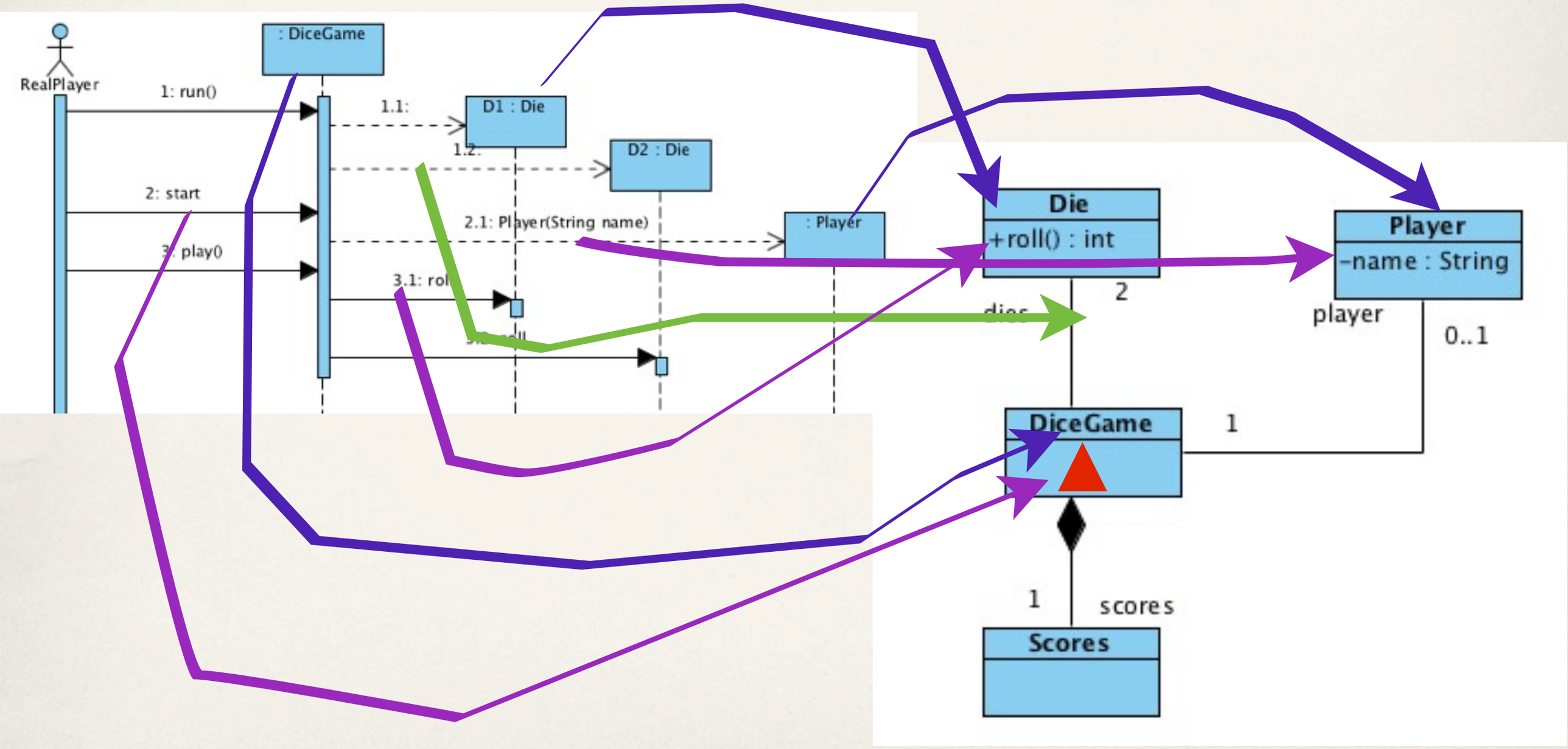


Diagramme d'états

- Identifier les états d 'un **objet**
- Identifier les transitions entre les états

Pascal Molli, molli@loria.fr

Diagramme d'état **d'un objet «partie»**

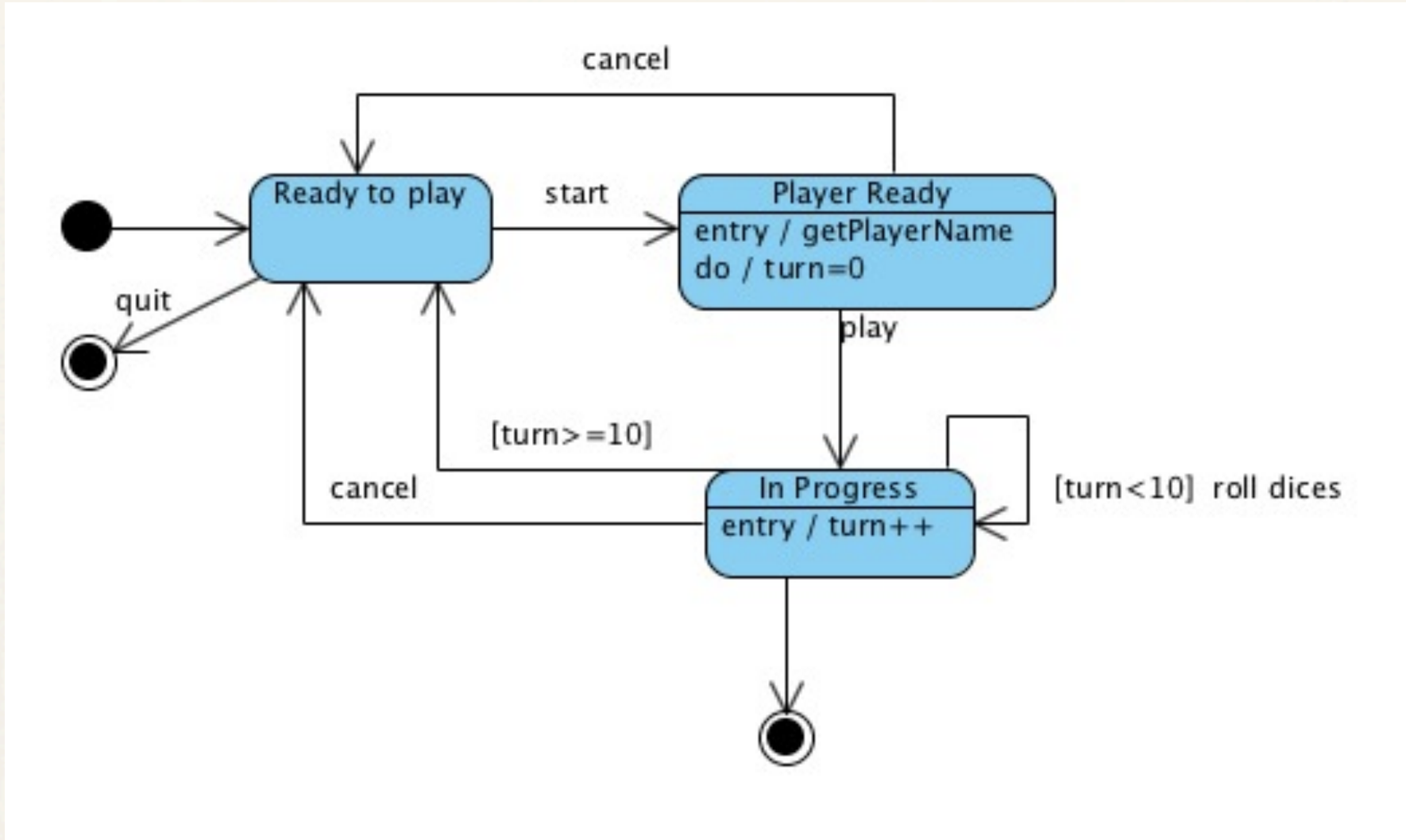
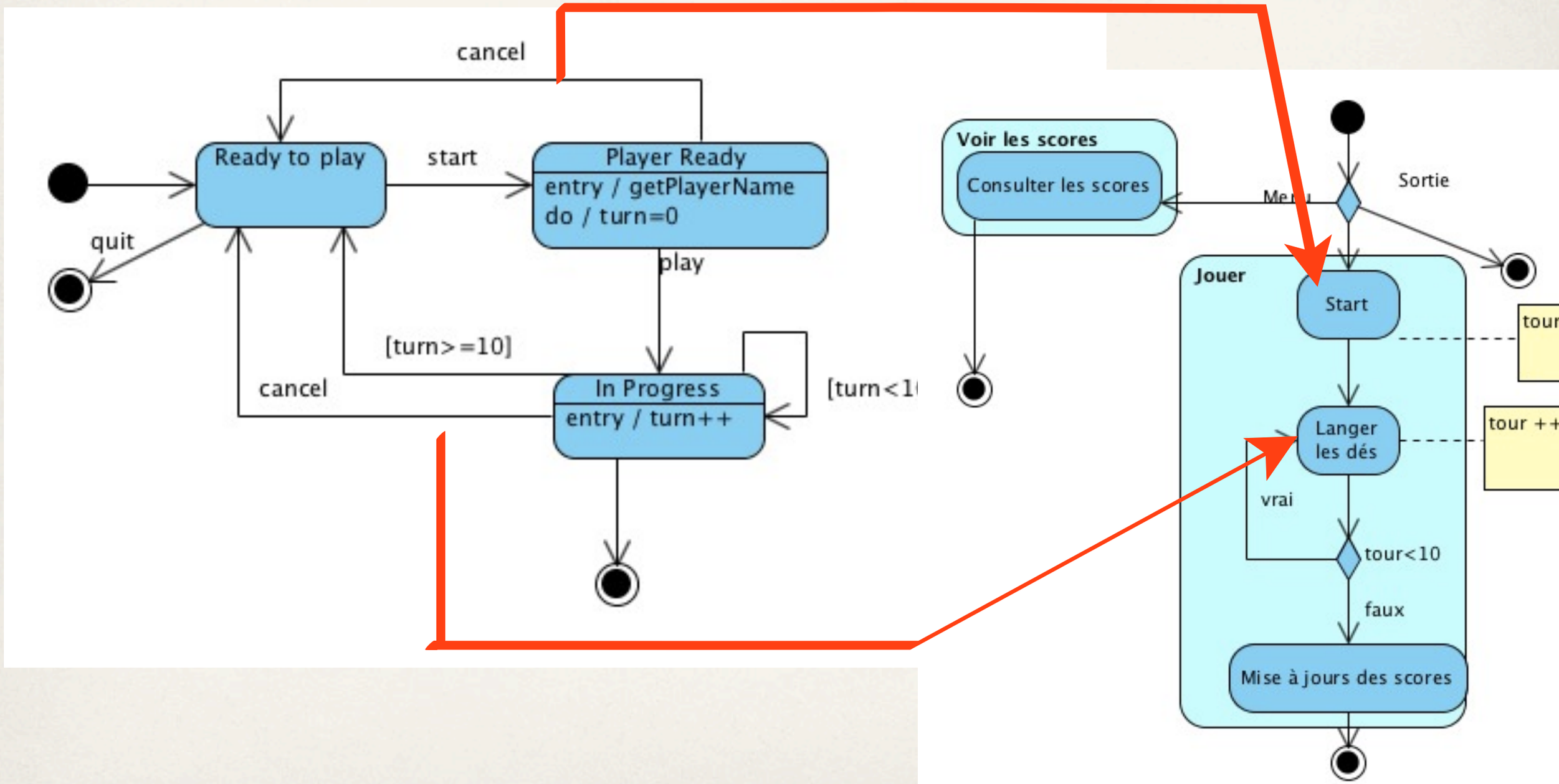
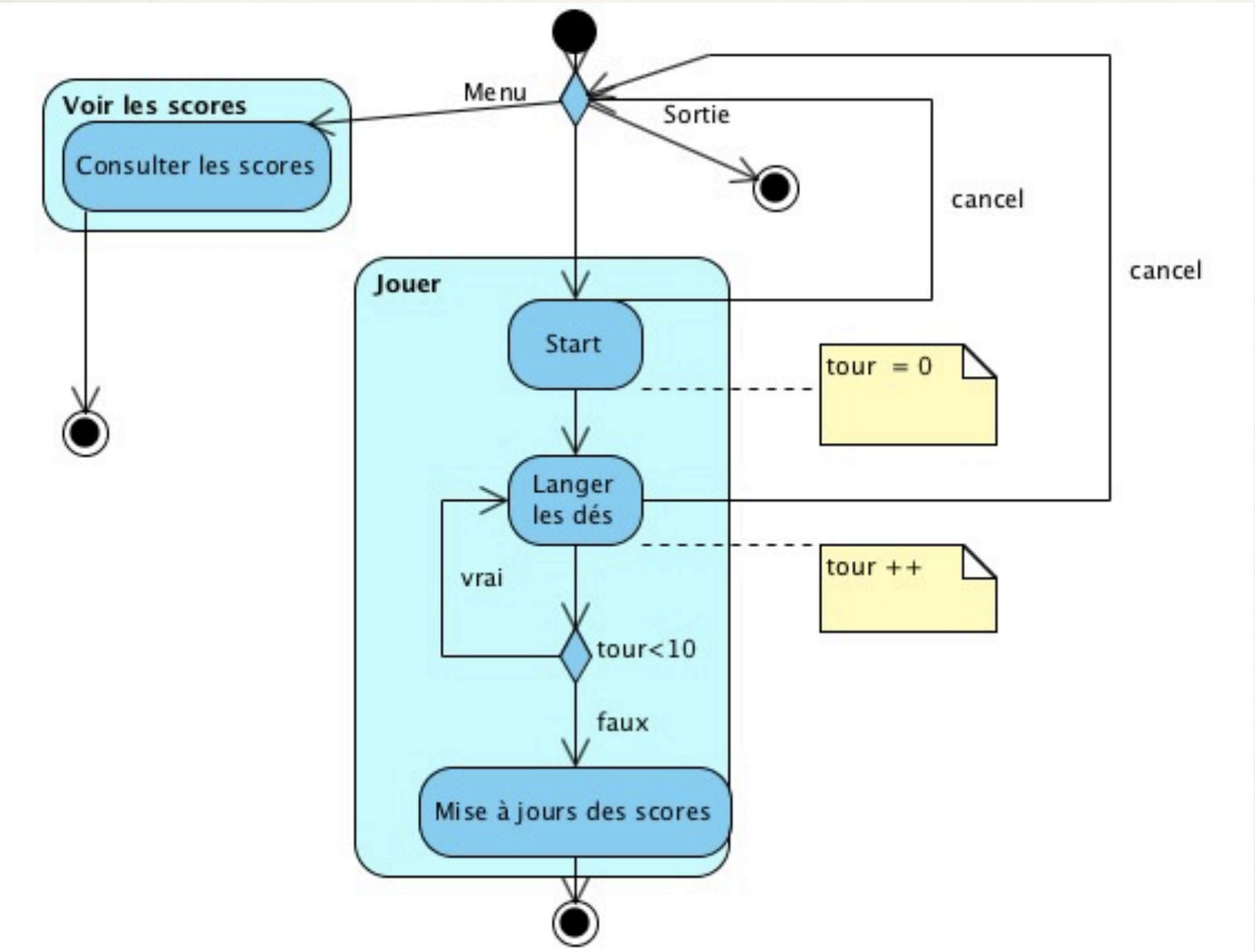


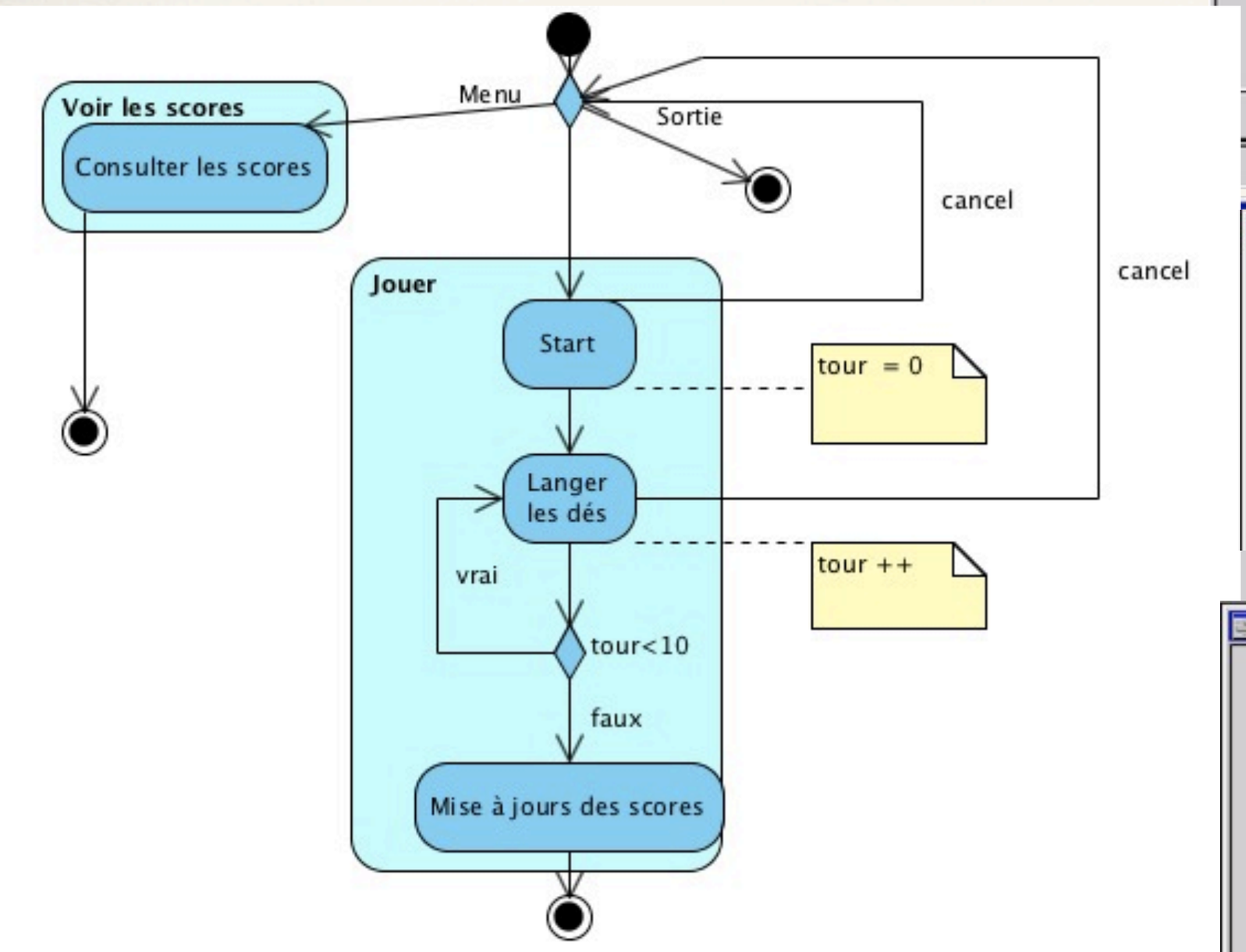
Diagramme d'état **d'un objet «partie»** & diagramme d'activité



Mettre à jour les schémas



Vérifier la cohérence !

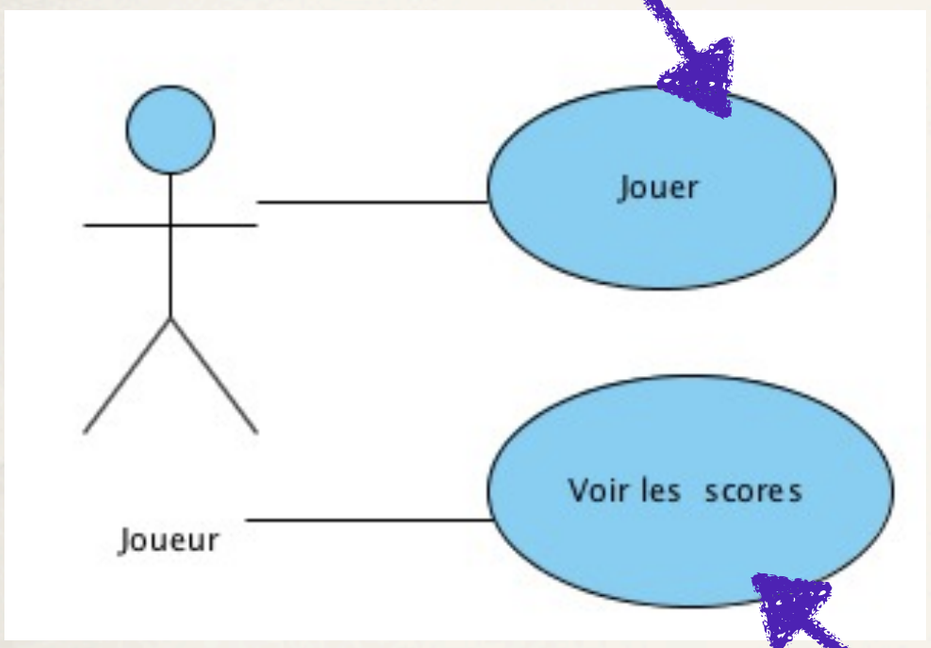


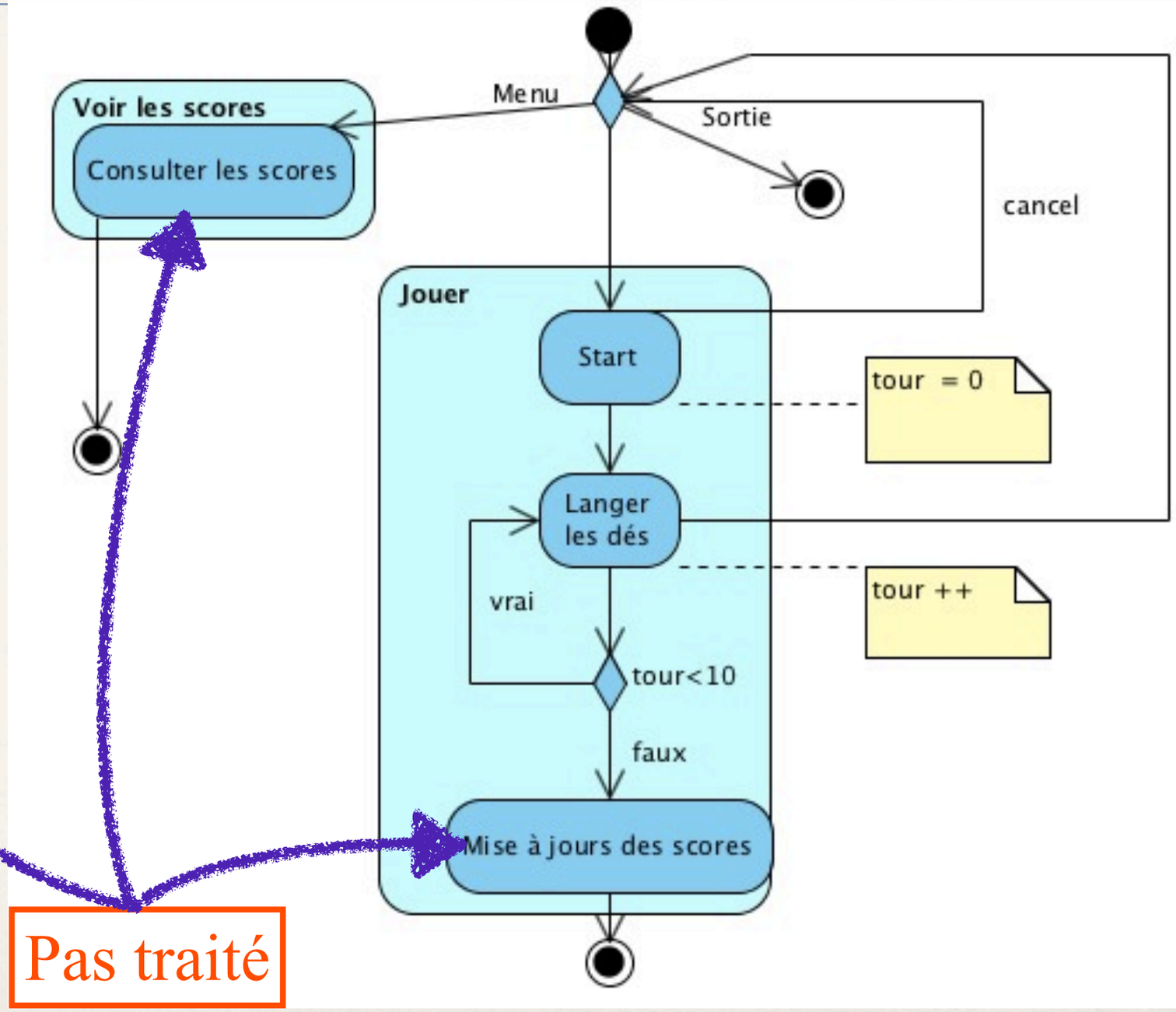
Analyse terminée ?

- Vérifier la couverture des diagrammes «use-case» et d ’activités…
- Use case « Voir les scores » ?
- Use case « Jouer » partiellement traité.

Couverture des diagrammes

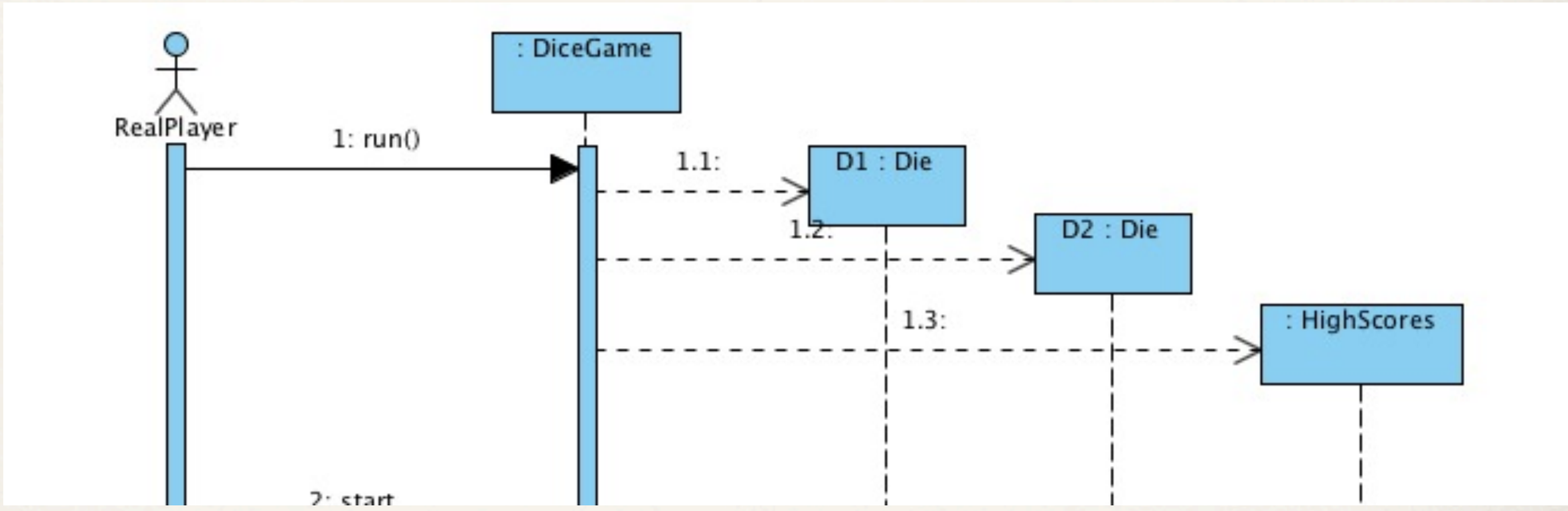
Partiellement traité





Pas traité

Diagramme de séquences complété



d'après Pascal Molli, molli@loria.fr

Diagramme de séquence modifié

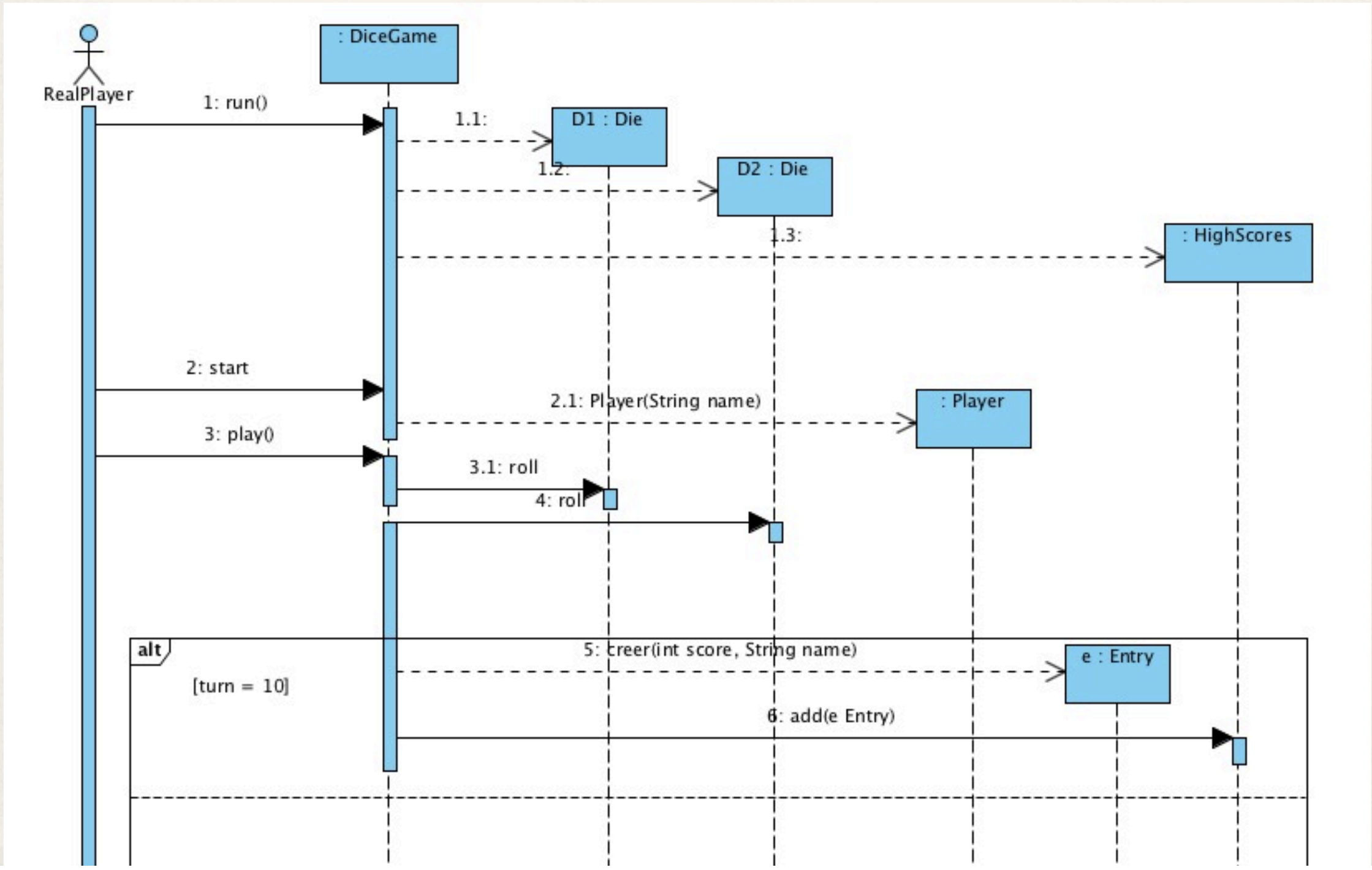
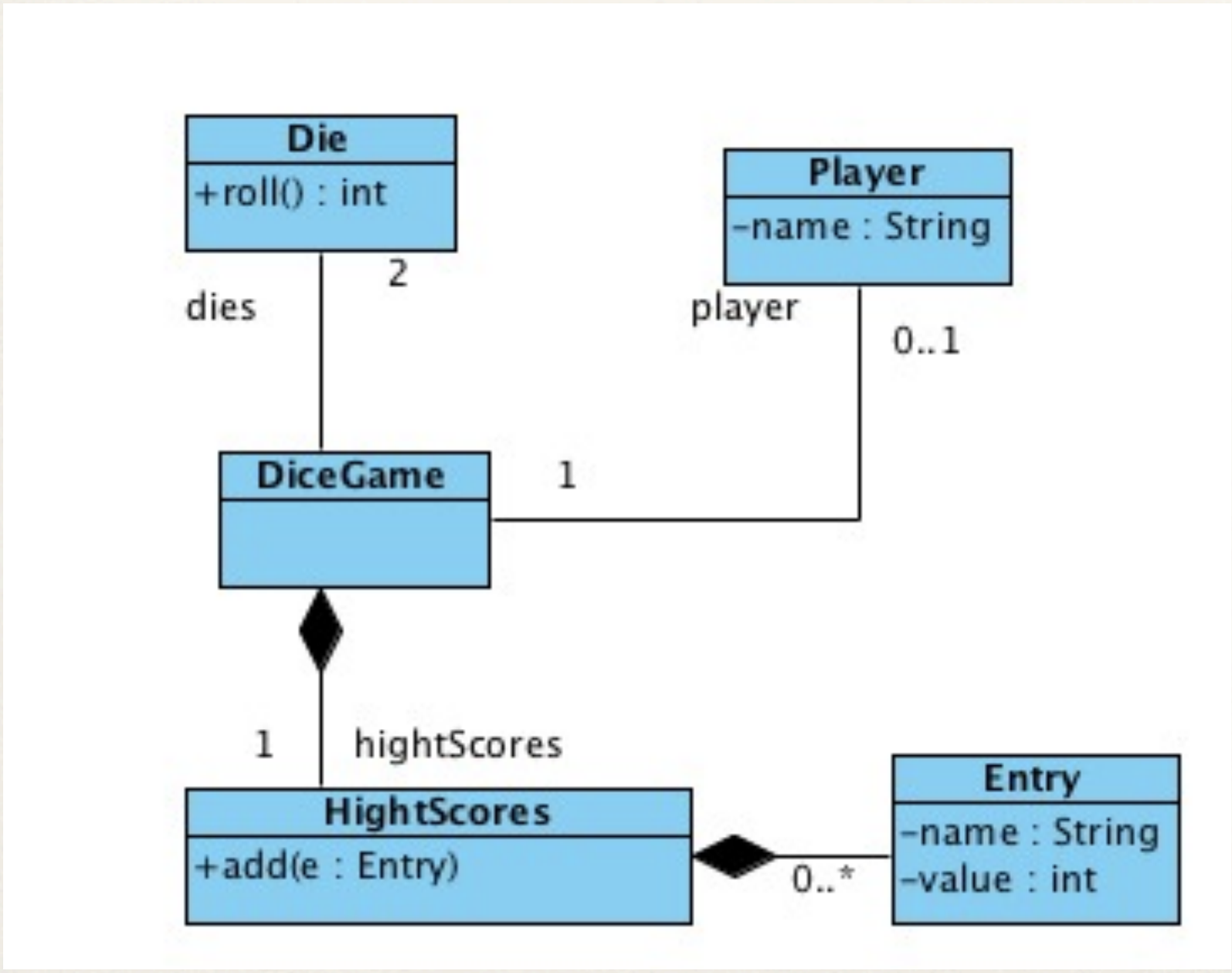


Diagramme de classes



Fin de l'analyse ?

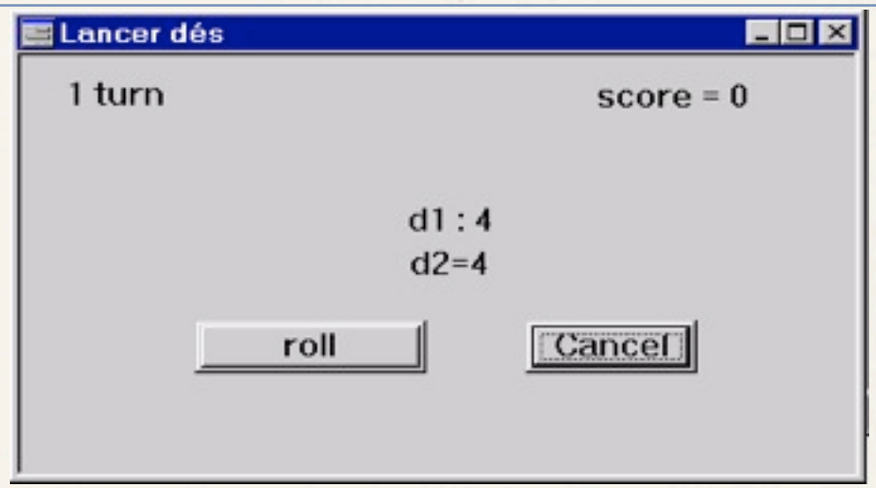
- Couverture « à peu près » bonne
- Cohérence entre les schémas correctes
  - La dynamique manque de détail (dynamique du cancel ?)
  - Les schémas sont trop peu expliqués…
  - Les diagrammes de séquence du jeu ne sont pas assez détaillés : manque quelques méthodes...

Conception

- Définir l'architecture
- Rajouter les classes techniques permettant d ’implanter cette architecture !
- Prendre en compte l’implémentation
  - Gérer la partie interface graphique
  - Gérer la persistance

Conception de l'architecture

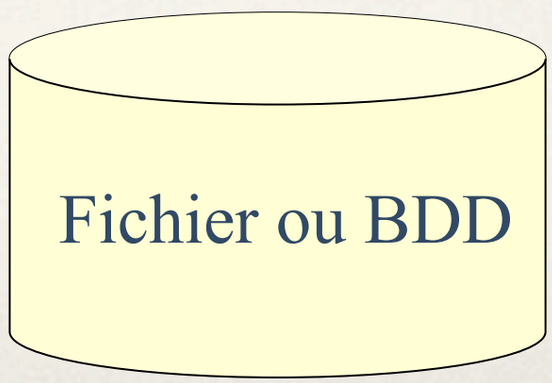
Présentation



Couches (Layers)

Applicatif

Persistance



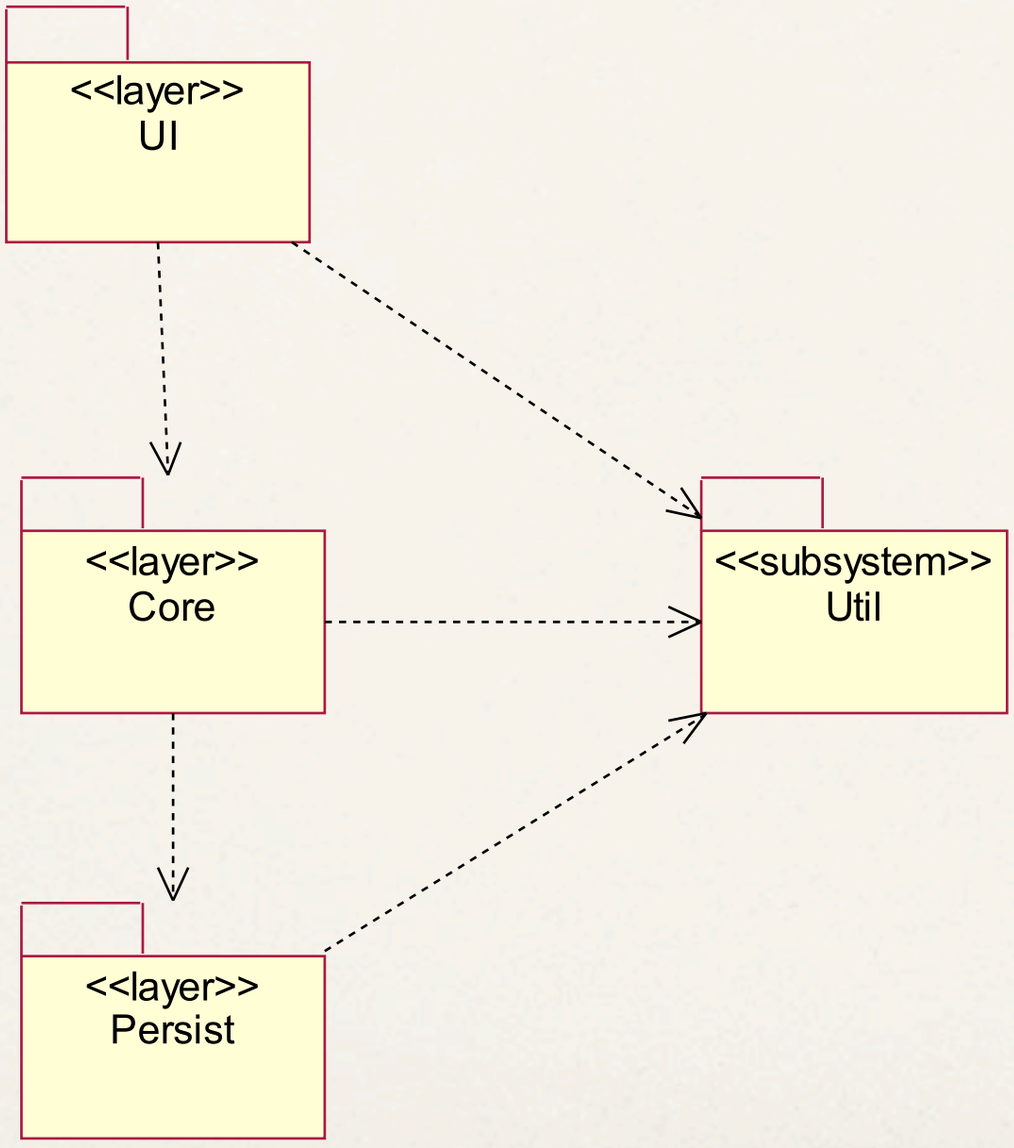
Architecture en couche...

- Une architecture possible...
- Les couches doivent être le plus indépendantes possible
- « Découpler » les couches en s'appuyant sur des interfaces et des classes abstraites

Des solutions à des problèmes récurrents : les «patterns»

d'après Pascal Molli, molli@loria.fr

Découpage en « packages » logiques



- Mapper l'architecture sur des packages « layer »
- Exprimer les dépendances

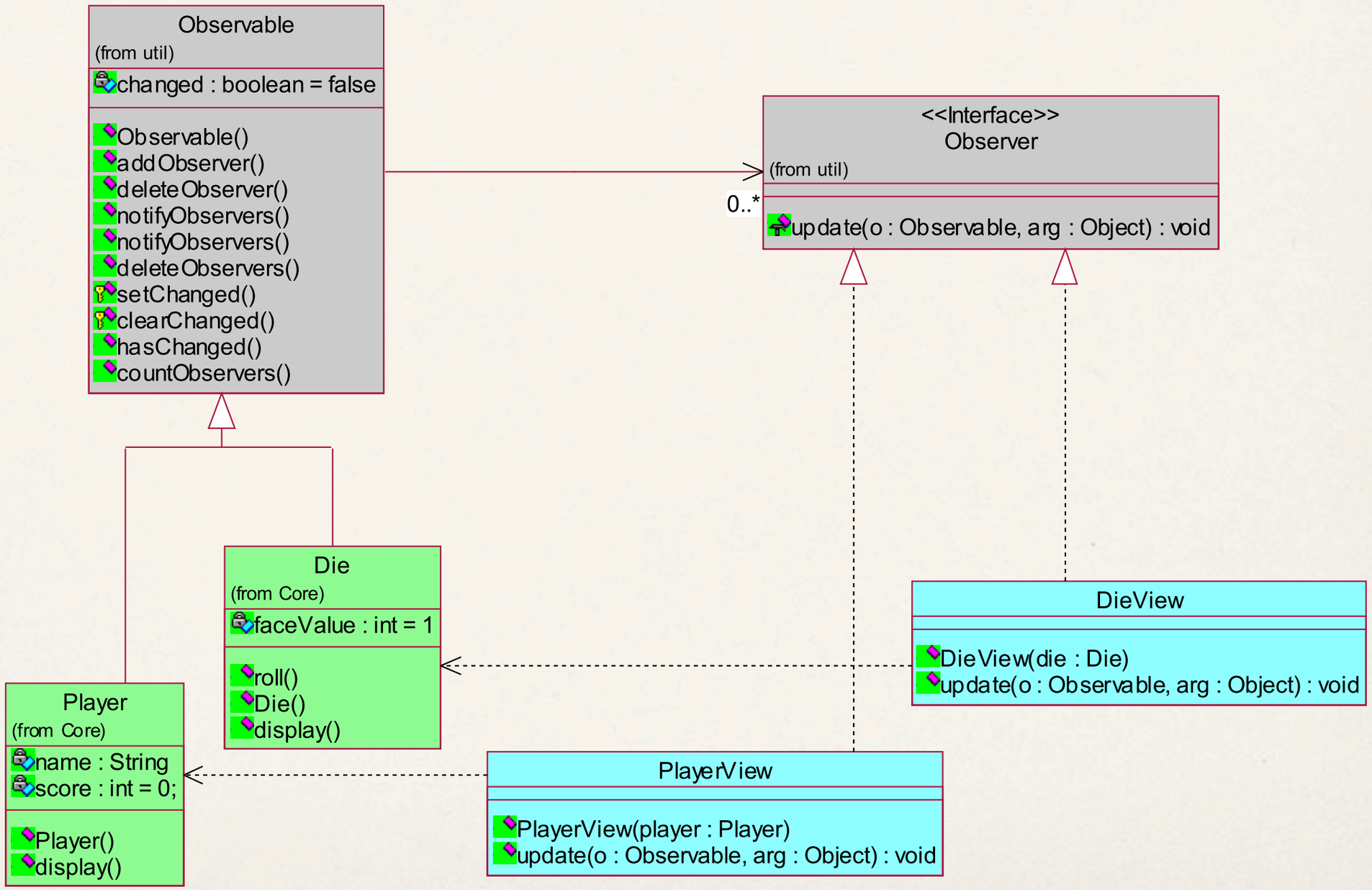
Layer « core »

- Les classes représentant la logique de notre application.

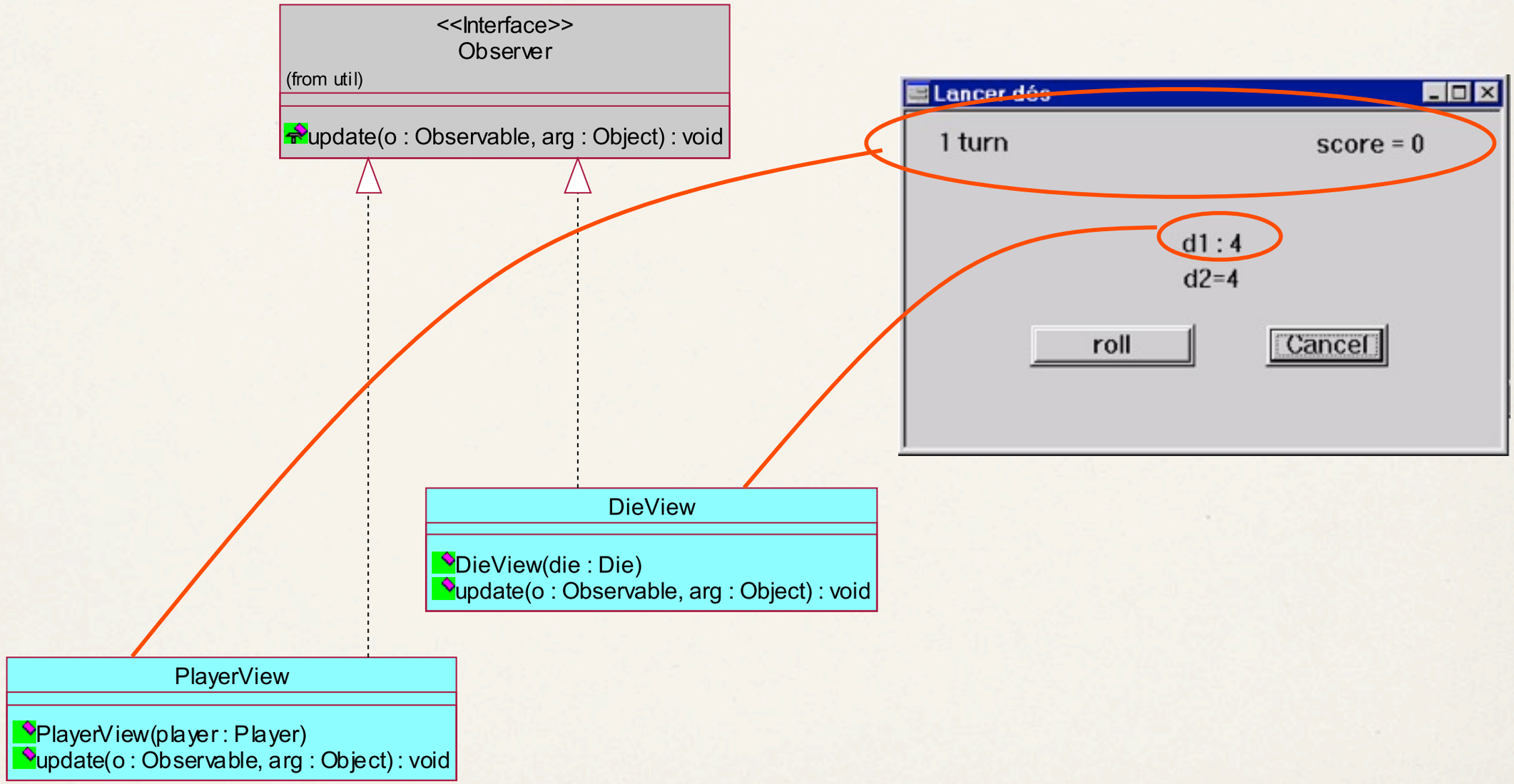
- En fait, les classes d'analyses «revisitées» en vue de la réalisation

Démarche abordée l'an dernier-
Nous la conforterons dans la suite des cours et TDs
dans une approche «Agile» et «Pragmatique».

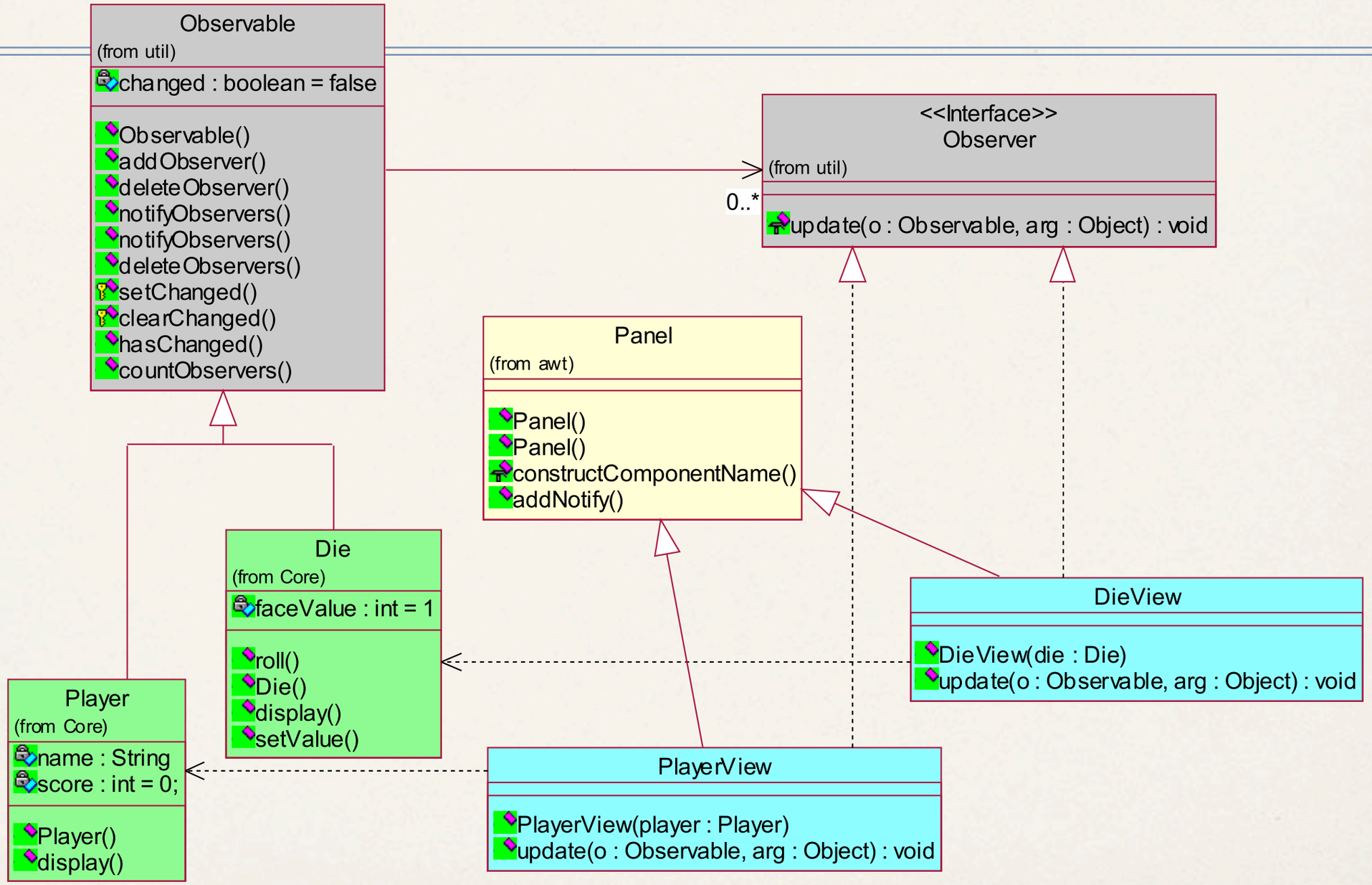
Découplage interface graphique : MVC (par exemple)



Vues ?

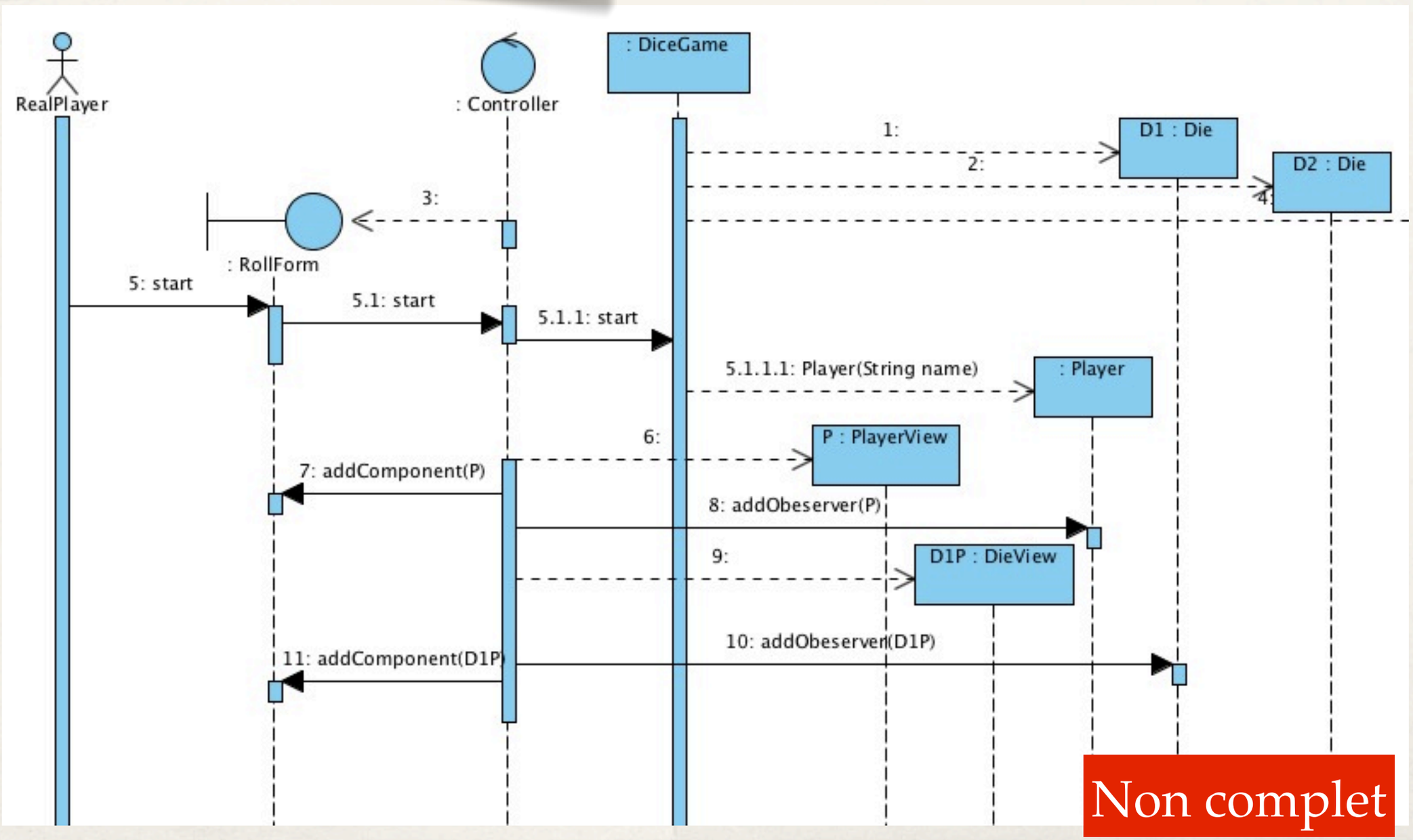

<<Interface>>
Observer
(from util)
update(o : Observable, arg : Object) : void
Lancer dés
1 turn
score = 0
d1 : 4
d2=4
roll
Cancel
DieView
DieView(die : Die)
update(o : Observable, arg : Object) : void
PlayerView
PlayerView(player : Player)
update(o : Observable, arg : Object) : void

Héritage et Réalisation ...



MVC en action:1 mise en place

Approfondissement de ce type d'architecture dans un prochain cours dédié.



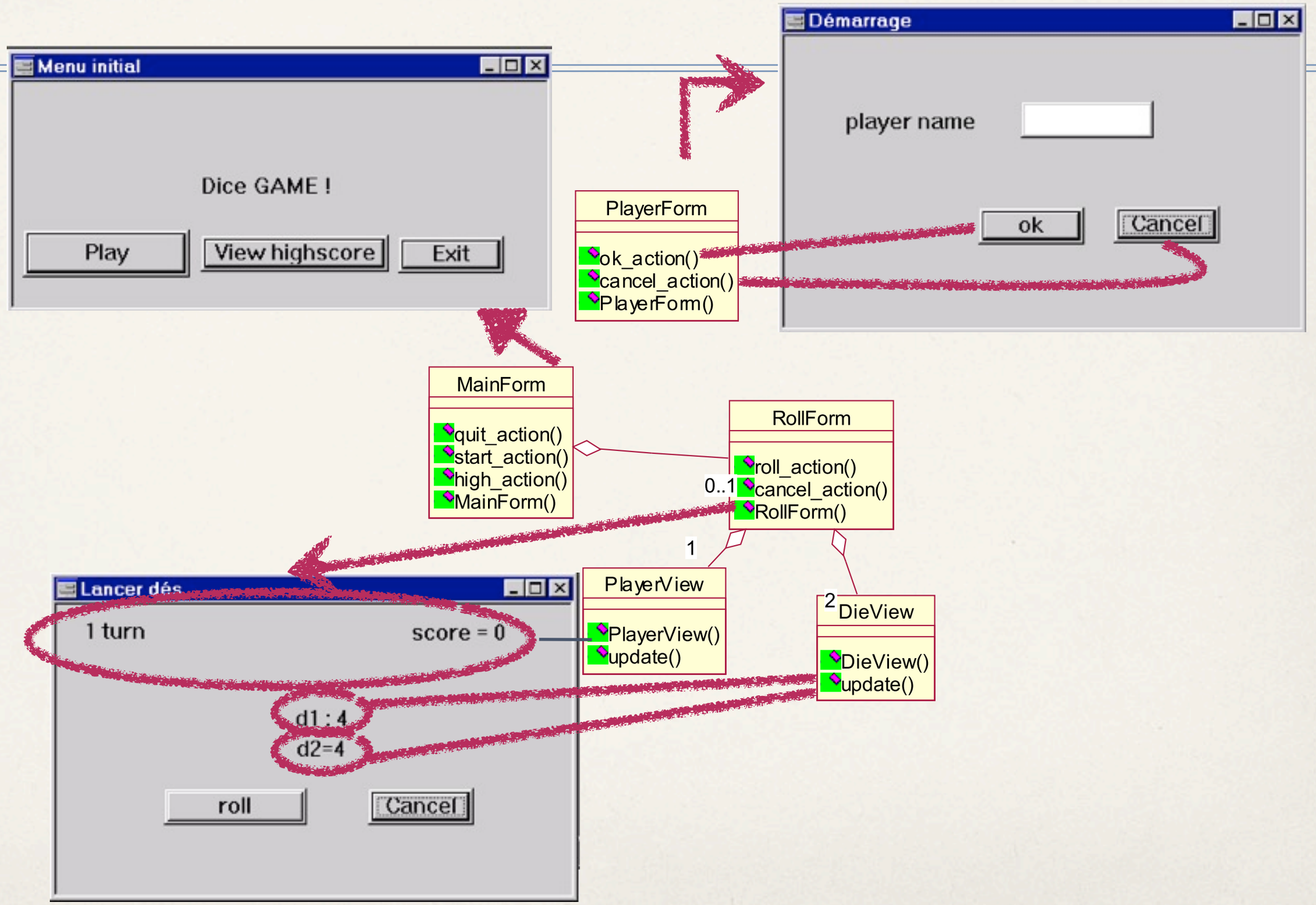
Non complet

Mettre les vues dans des « forms »

- Réaliser des écrans graphiques contenant des vues s'il y a lieu…
- Le « layer » UI…

Pascal Molli, molli@loria.fr

Mapping classe UI, UI



Pascal Molli, molli@loria.fr

Architecture en couche...

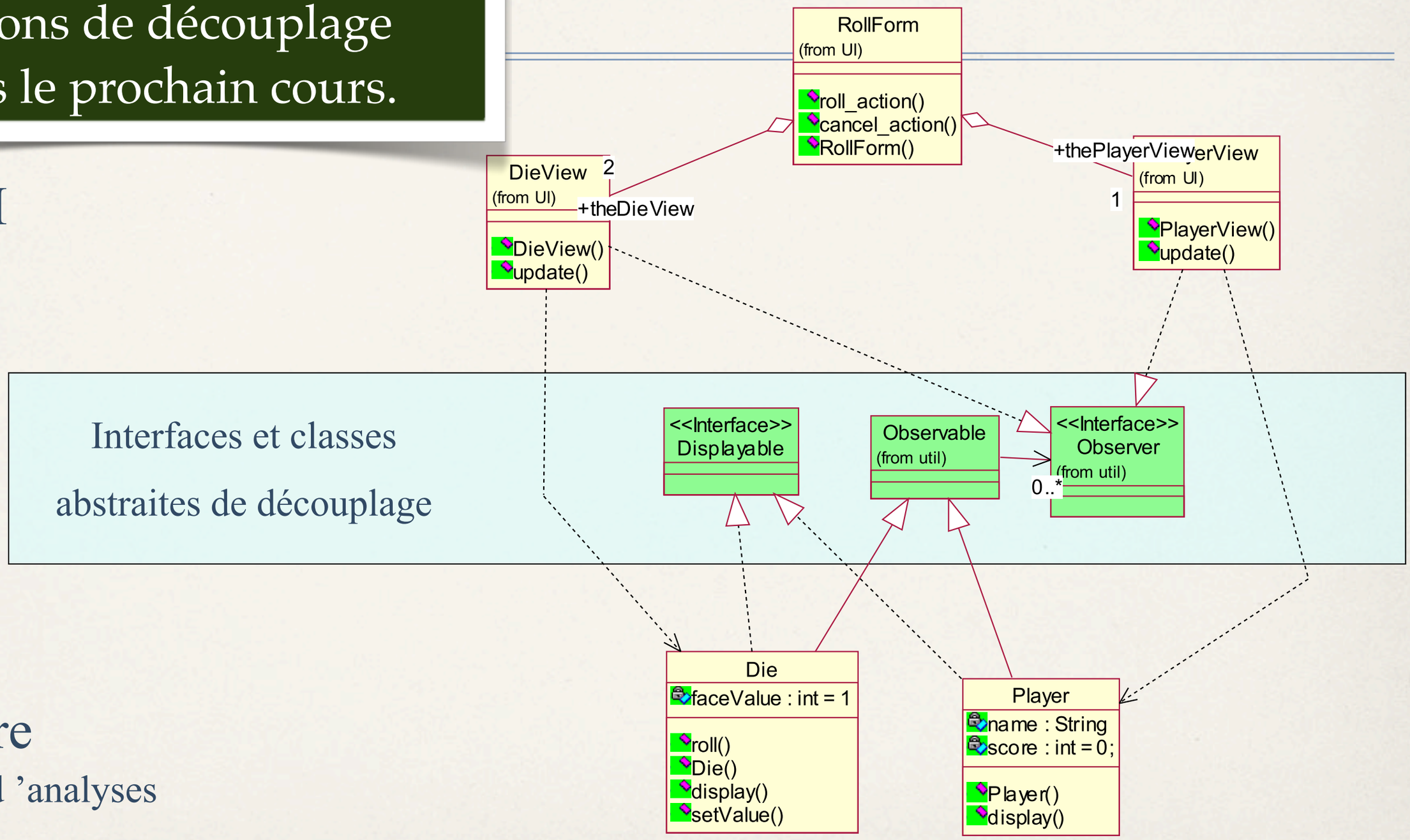
Approfondissement des notions de découplage dans le prochain cours.

UI

Interfaces et classes abstraites de découplage

Core

Classes d ’analyses



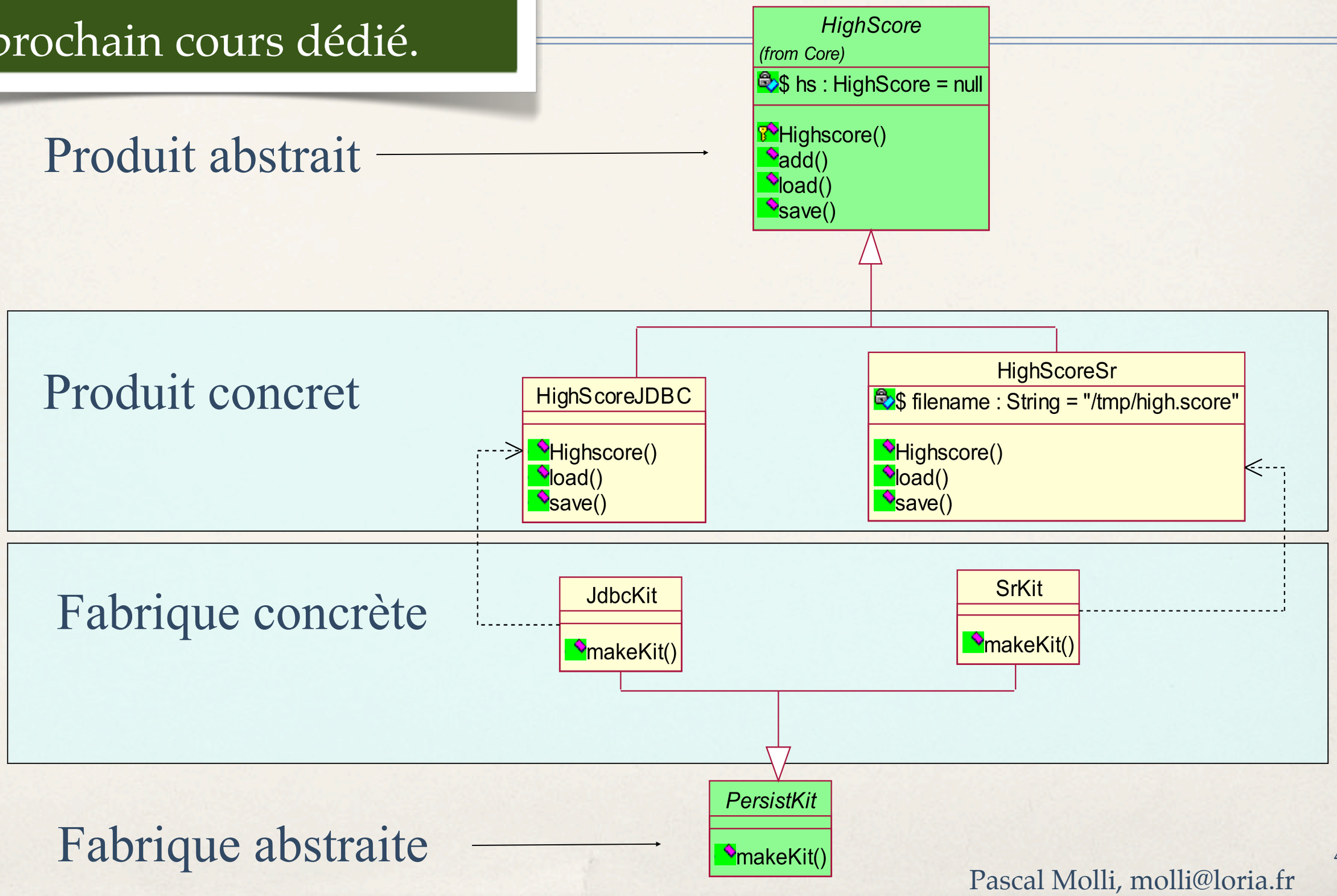
Layer « Persist » : Couche de persistance

- Classes techniques de persistance
- Assurer l'indépendance Core/Persist
  - pouvoir changer de « persistent engine »
- Par exemple:
  - Persistance par « Serialisation »
  - Persistance via une base de données relationnelle (JDBC).

Découplage : Pattern Fabrication

Approfondissement de ce type d'architecture dans un prochain cours dédié.

Produit abstrait



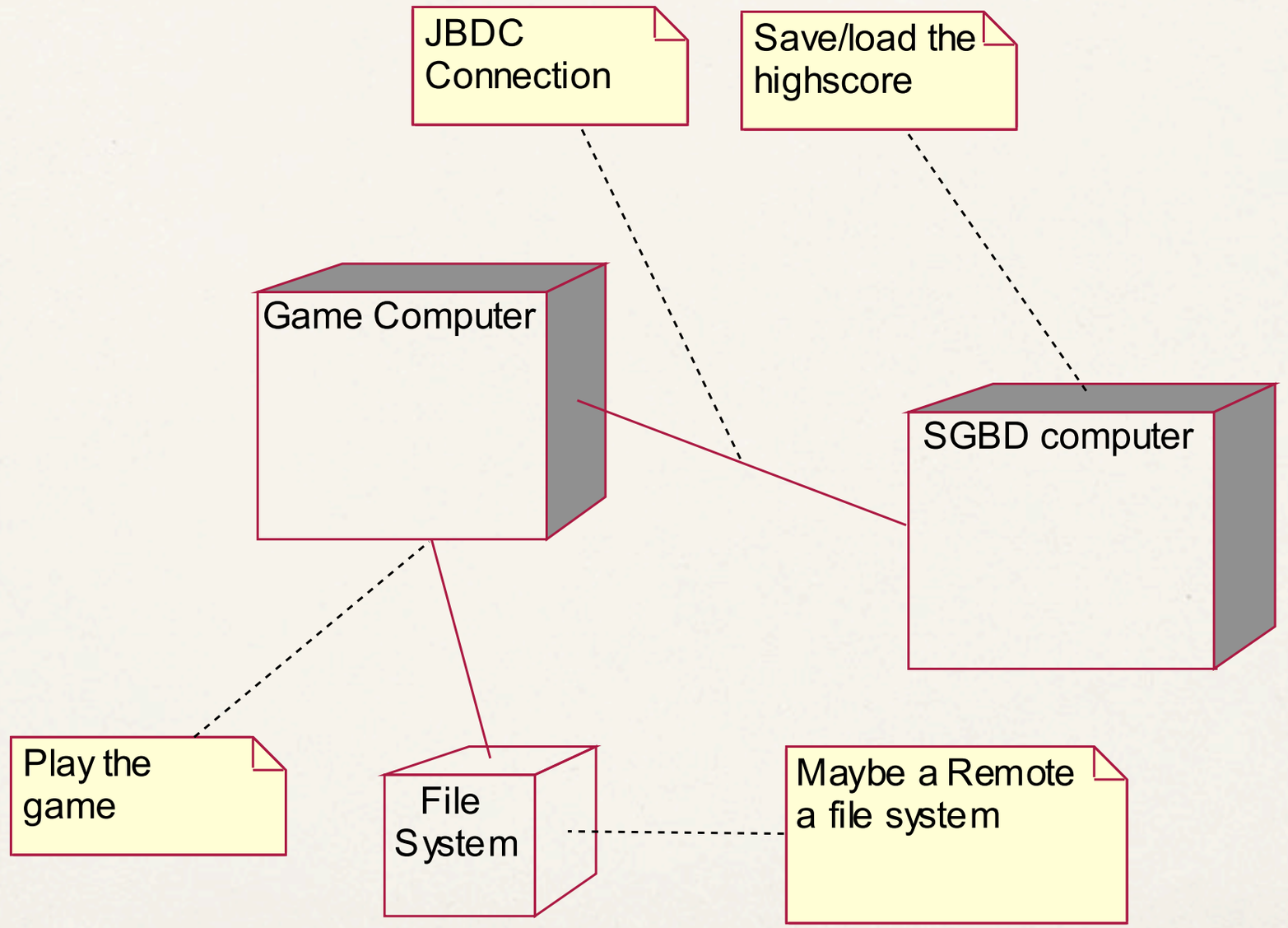
Pascal Molli, molli@loria.fr

Diagrammes de déploiement

- Visualiser l ’architecture physique
- Associer les unités d ’exécution aux traitements
- Identifier les connexions entre les unités d ’exécution

Au delà des enseignements pour cette année.

Diagramme de déploiement



Pascal Molli, molli@loria.fr

Design terminé ?

- Couverture des fonctionnalités : comparer Use-case et activity diagram…
- Cohérence entre les diagrammes ?
  - Quelques incohérences… UI vs Core ; gestion des changements d'états?
  - Indépendance Core/Persist partiellement atteinte...

Générer le code : code mapping

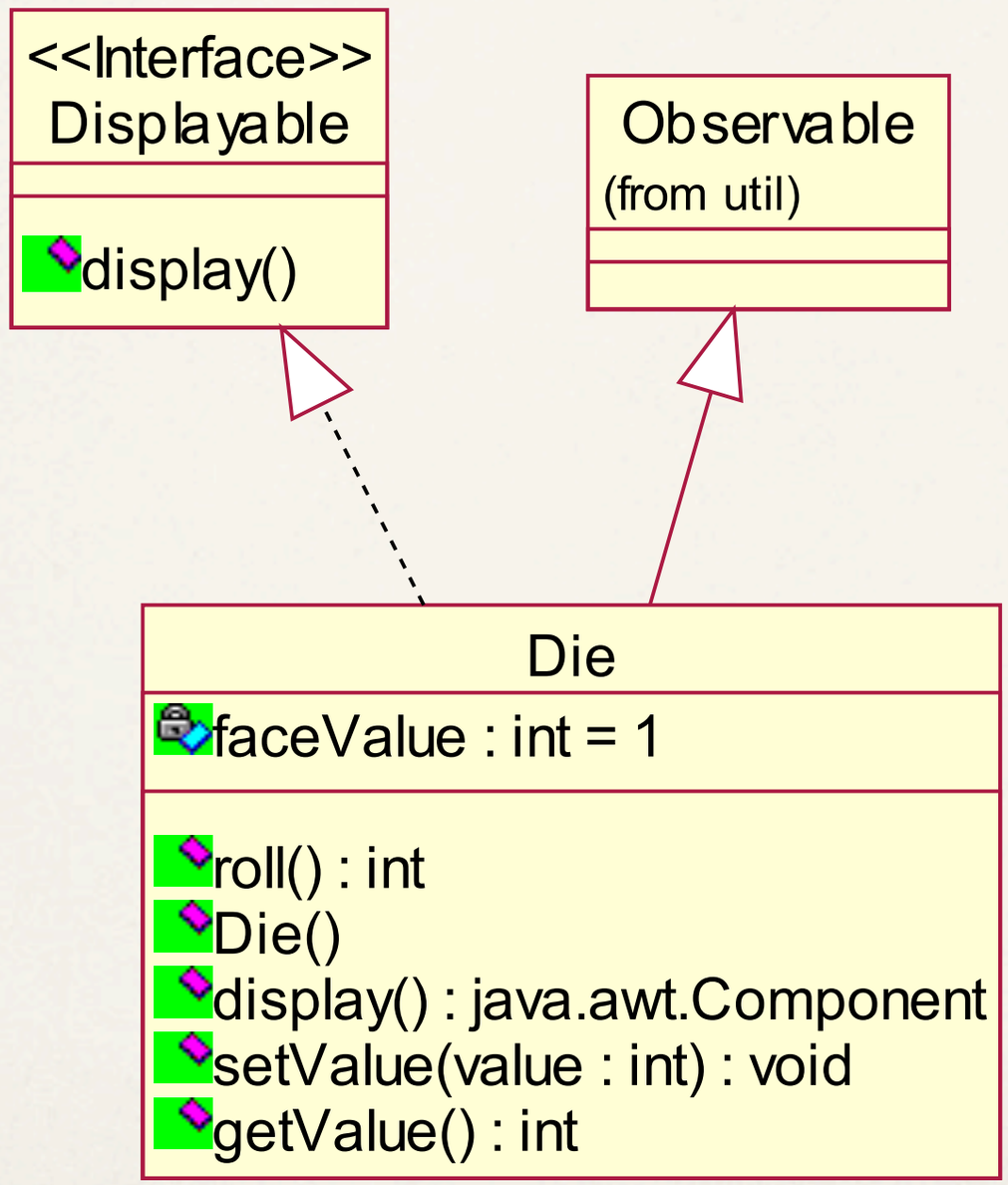
- Mapper vers n'importe quel langage !*
  - Les langages objets : java, C++, smalltalk
  - Mais aussi les autres: VB, C, Fortran, Php
  - Voir aussi: SQL...

Déjà vu l'an dernier

* automatiquement, encore faut-il que l'environnement le supporte !

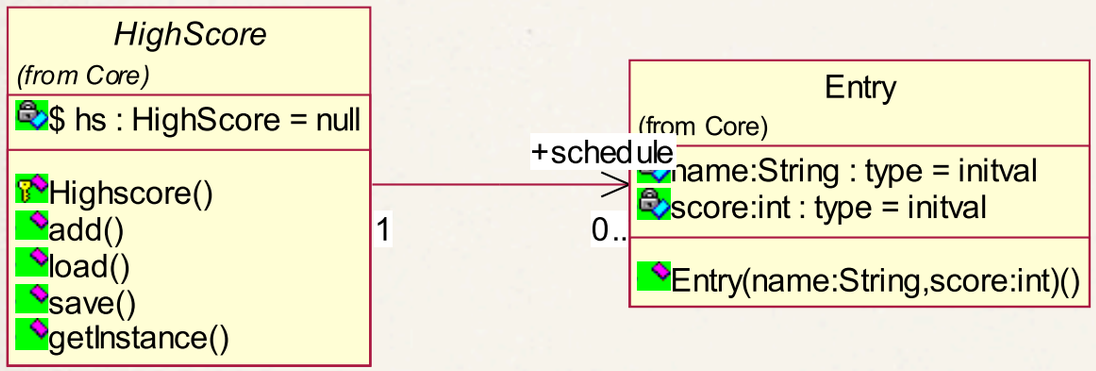
d'après Pascal Molli, molli@loria.fr

Mapping java... (Rappels)




```
package Core;

import Util.Randomizer;
import UI.DieView;
import java.util.*;
import java.awt.Component;

public class Die extends
Observable implements Displayable
{
private int faceValue = 1;

public int roll() {

setValue(Randomizer.getInstance().
    getValue());
    return getValue();
    }
public java.awt.Component
display() {
    Component c=new DieView(this);
    this.addObserver((Observer)c);
    return c;
    }
public void setValue(int value) {
    faceValue=value;
    this.setChanged();
    this.notifyObservers();
    }
public int getValue() { return
faceValue;}
}
```

Mapping Java : Relations (Rappels)




```
package Core;
import java.util.*;
import java.awt.Component;
import UI.HighScoreView;
public abstract class HighScore
extends Observable implements
java.io.Serializable, Displayable {
    protected static HighScore hs = null;
    public Vector schedule=new Vector();
public void add(Entry entry) {
    entries.addElement(entry);
    this.setChanged();
    this.notifyObservers();
    }

    public Enumeration elements() {
    return entries.elements();
    }
public abstract void load();
public abstract void save();
public Component display() {
 Component c=new HighScoreView(this);
 this.addObserver((java.util.Observer)c);
 return c;
}
public static HighScore getInstance() {
 if (hs==null) {
     new Error("No Persist Kit declared");
 }
 return hs;}
```

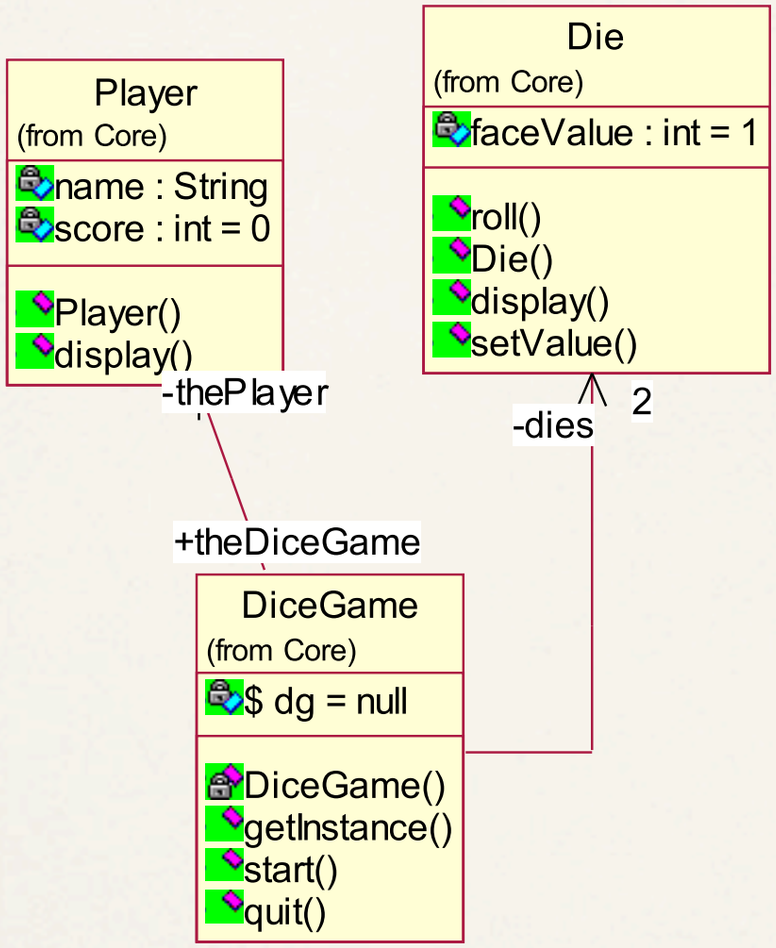
d'après Pascal Molli, molli@loria.fr

Codage…

- Utiliser les fonctionnalités de « forward engineering » des outils
- Puis de « reverse engineering »
- Mais l'idéal : « round trip engineering »
- Assurer la cohérence Code/Design/analyse…

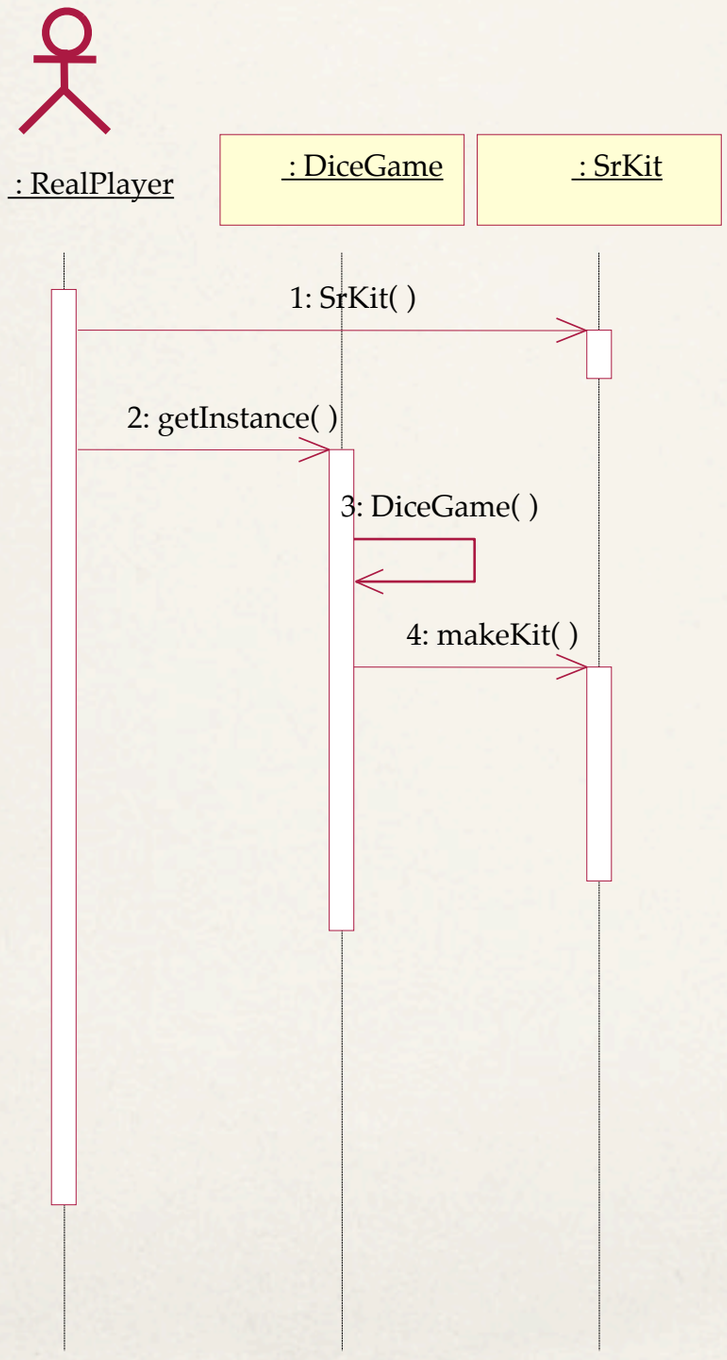
Une séance de TD consacrée au Reverse et à la qualité du logiciel

«Forward engineering»



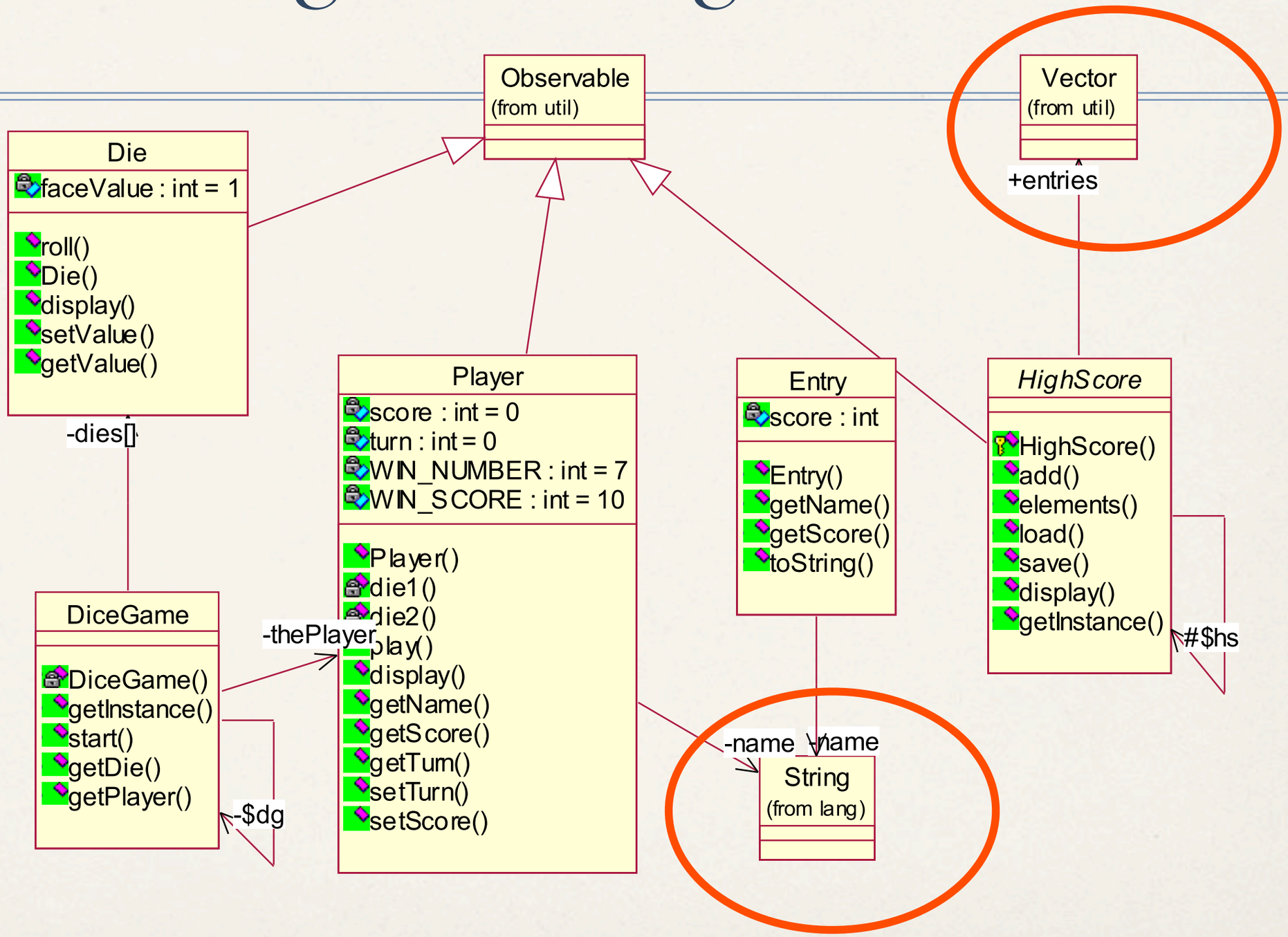

```
package Core;
public class DiceGame {
    private static int dg = null;
    private Die dies[];
    private Player thePlayer;
    DiceGame() {
    }
    /**
       @roseuid 37F877B3027B
     */
    private DiceGame() {
    }
    /**
       @roseuid 3802F61403A0
     */
    public void getInstance() {
    }
    /**
       @roseuid 37F8781A014D
     */
    public void start() {
    }
    /**
       @roseuid 38074E7F0158
     */
    public void quit() {
    }
```

Forward Engineering




```
package Core;

import UI.MainForm;
import Persist.*;
import java.awt.*;

class Main {
    public static void main(String args[]) {
    // SrKit srk=new SrKit();
    JdbcKit srk=new JdbcKit();
    DiceGame dg=DiceGame.getInstance();
    Frame f=MainForm.getInstance();
    f.setSize(300,300);
    f.show();
    }
}
```

«Reverse Engineering...»



Reverse engineering

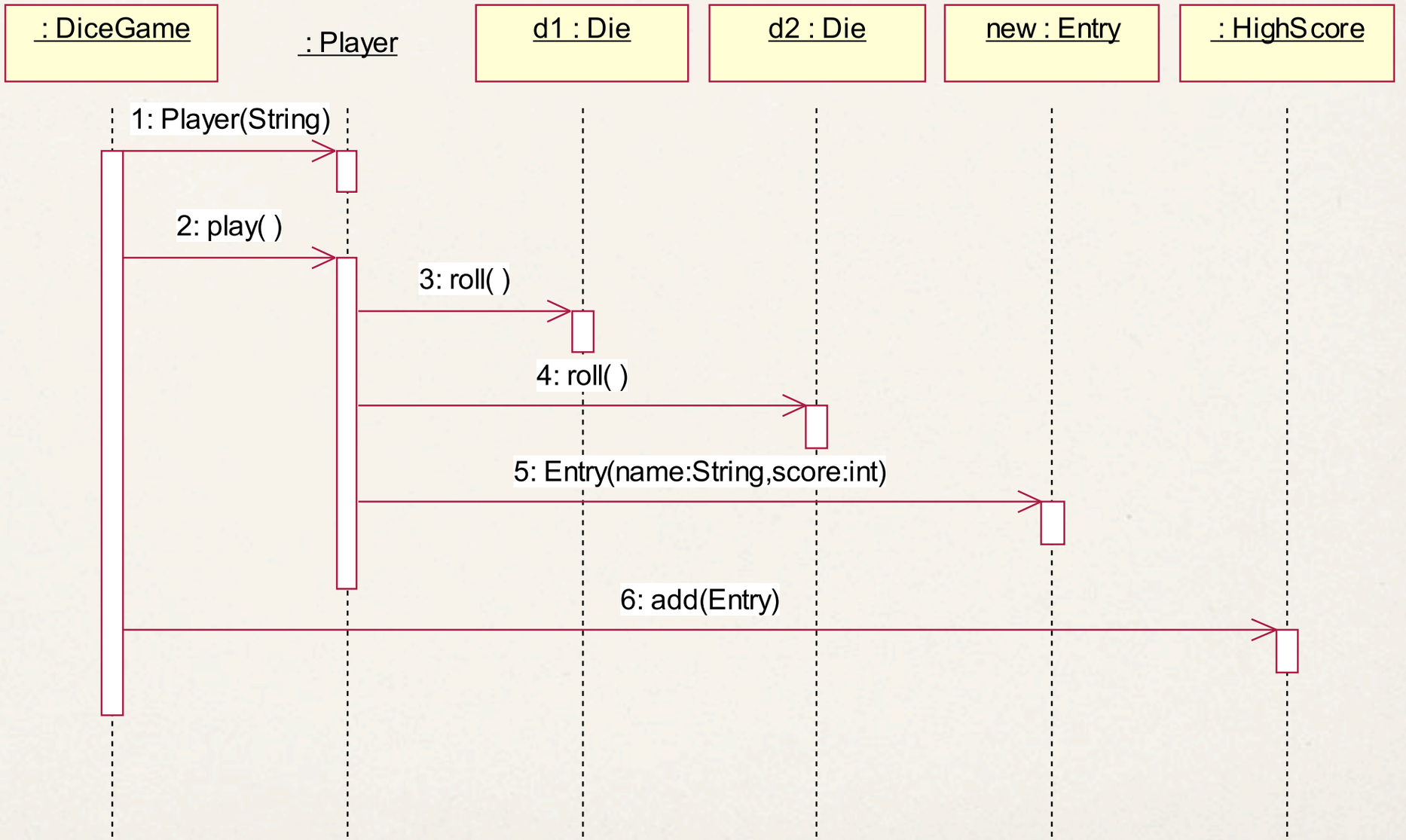
- Peu sur la dynamique (mais vous pouvez faire du reverse sur les diagrammes de séquence)
- Gère les aspects forward+modification+reverse
- Pas miraculeux !

d'après Pascal Molli, molli@loria.fr

Problèmes rencontrés

- Dynamique de la gestion des tours mal conçue !
- Qui véritablement fait le test de fin de jeu ?
- Faiblesse dans le design !

Ici ! Diagramme d'analyse !!

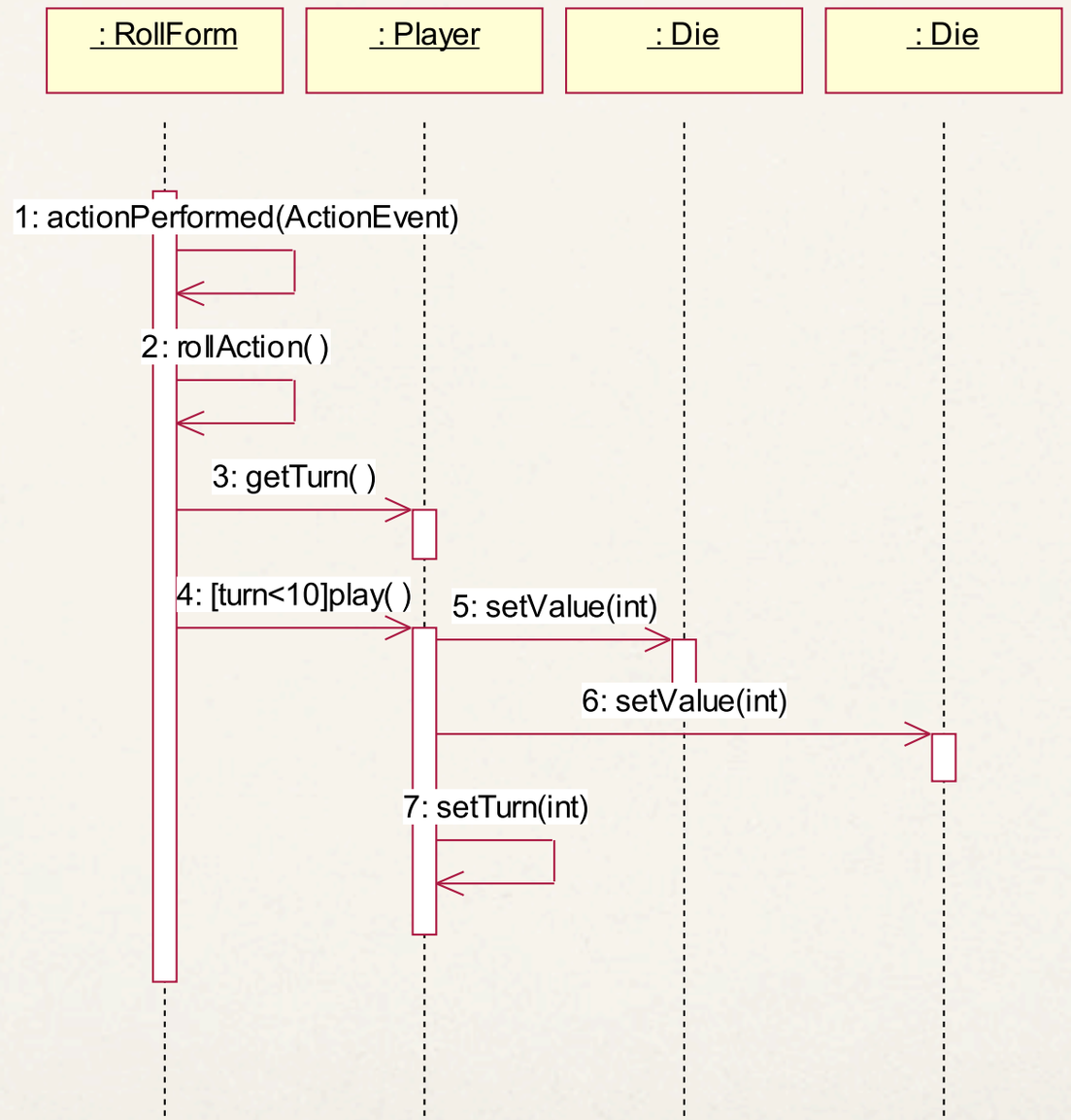


Pascal Molli, molli@loria.fr

Problème !

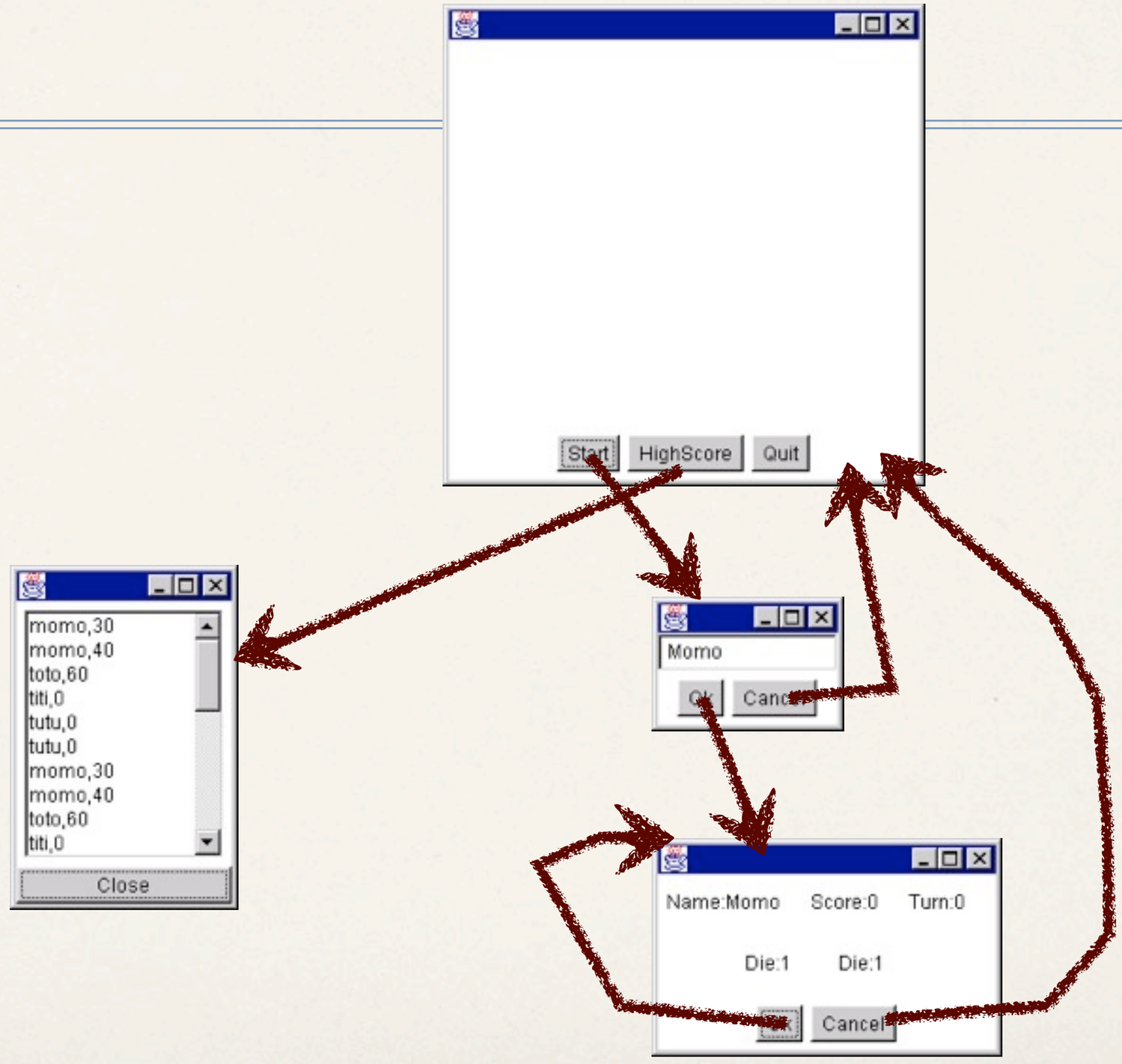
- Pas assez rigoureux !
- Ce diagramme d'analyse n'a pas été revu au design !!!
- (-4)

Refaire !



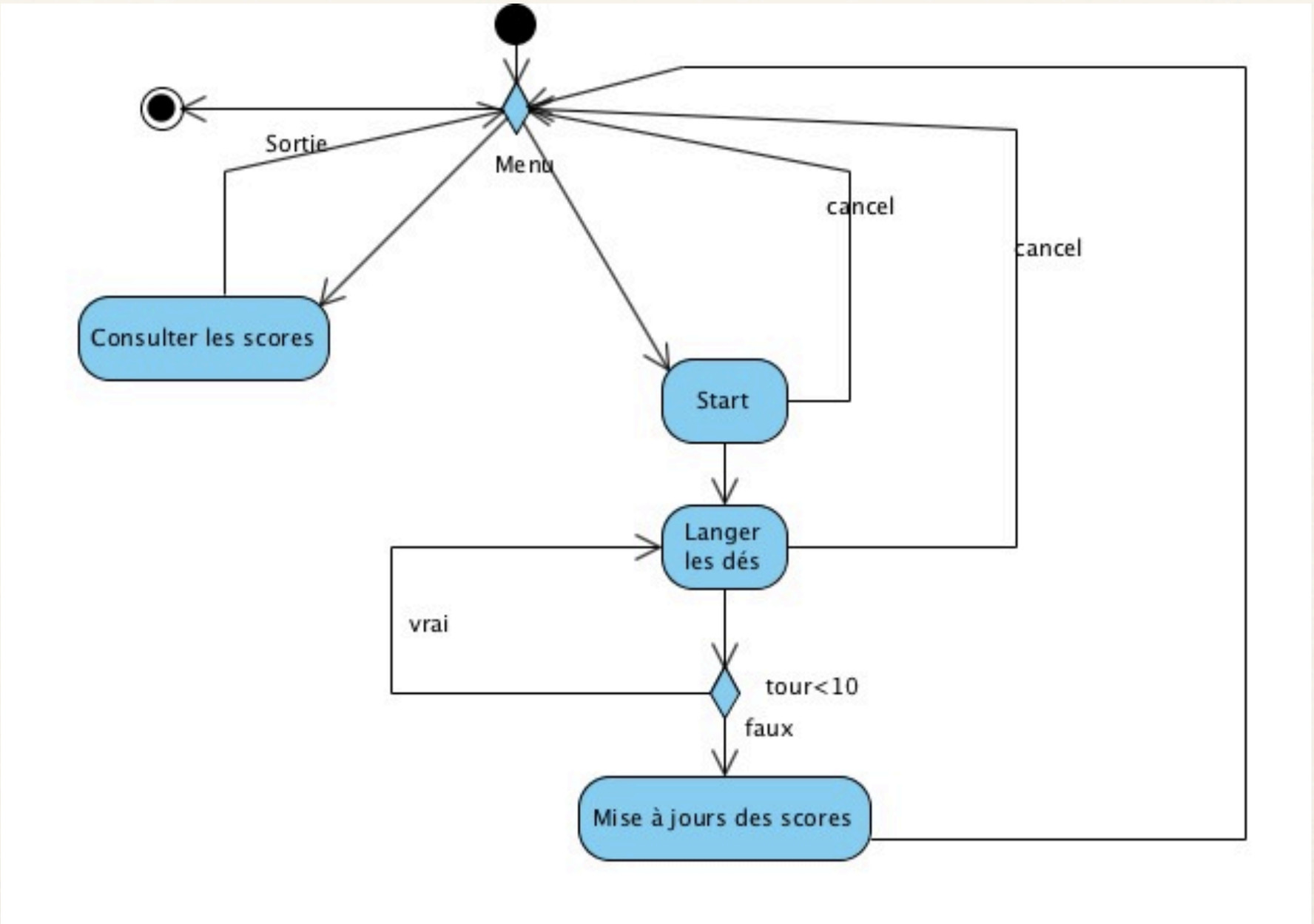
Pascal Molli, molli@loria.fr

Finalement !



Pascal Molli, molli@loria.fr

Après révision partielle !

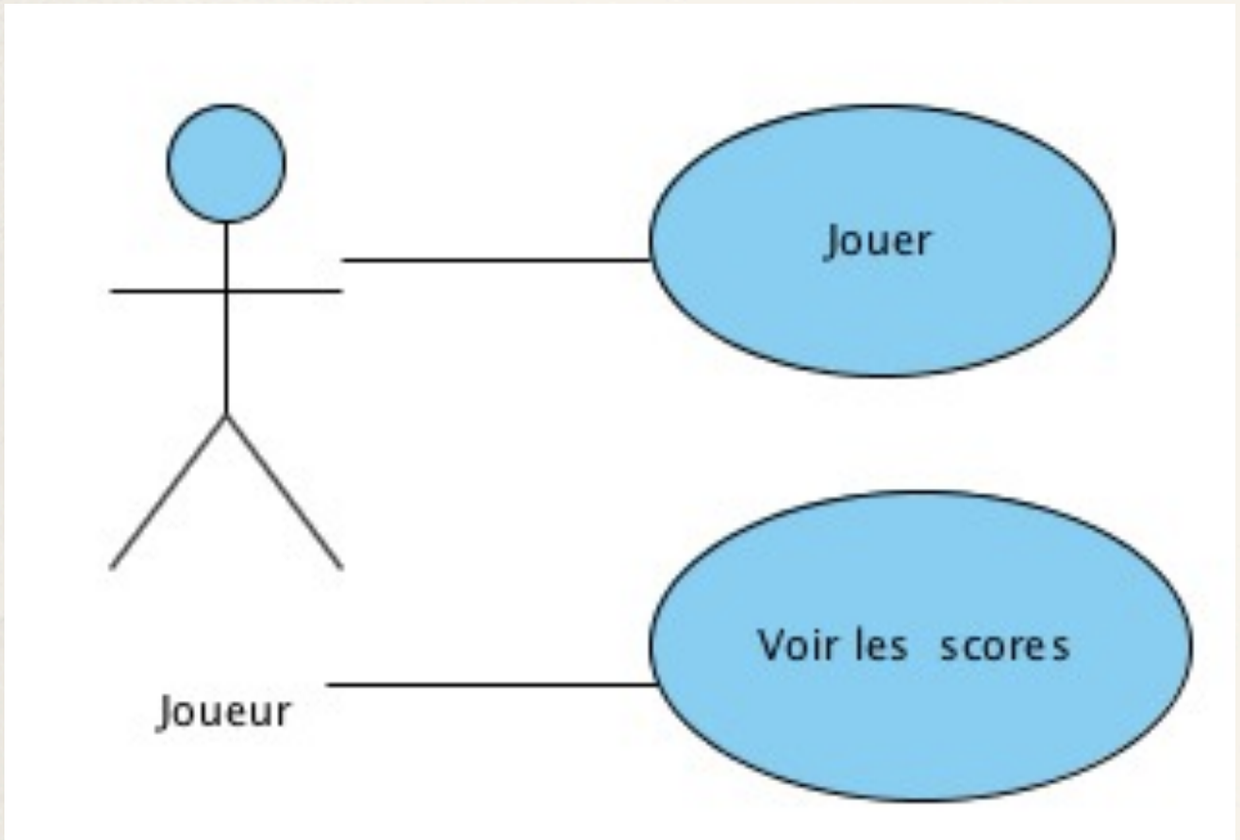


Pascal Molli, molli@loria.fr

Est-ce que ça marche ? Tester

- Tests unitaires : tester classe par classe, méthode par méthode
  - Diagramme de classes
- Tests d'intégration :
  - Nous travaillerons sur des diagrammes de classes et de packages
- Tests du système :
  - Diagramme Use Case + Activités

Test du système

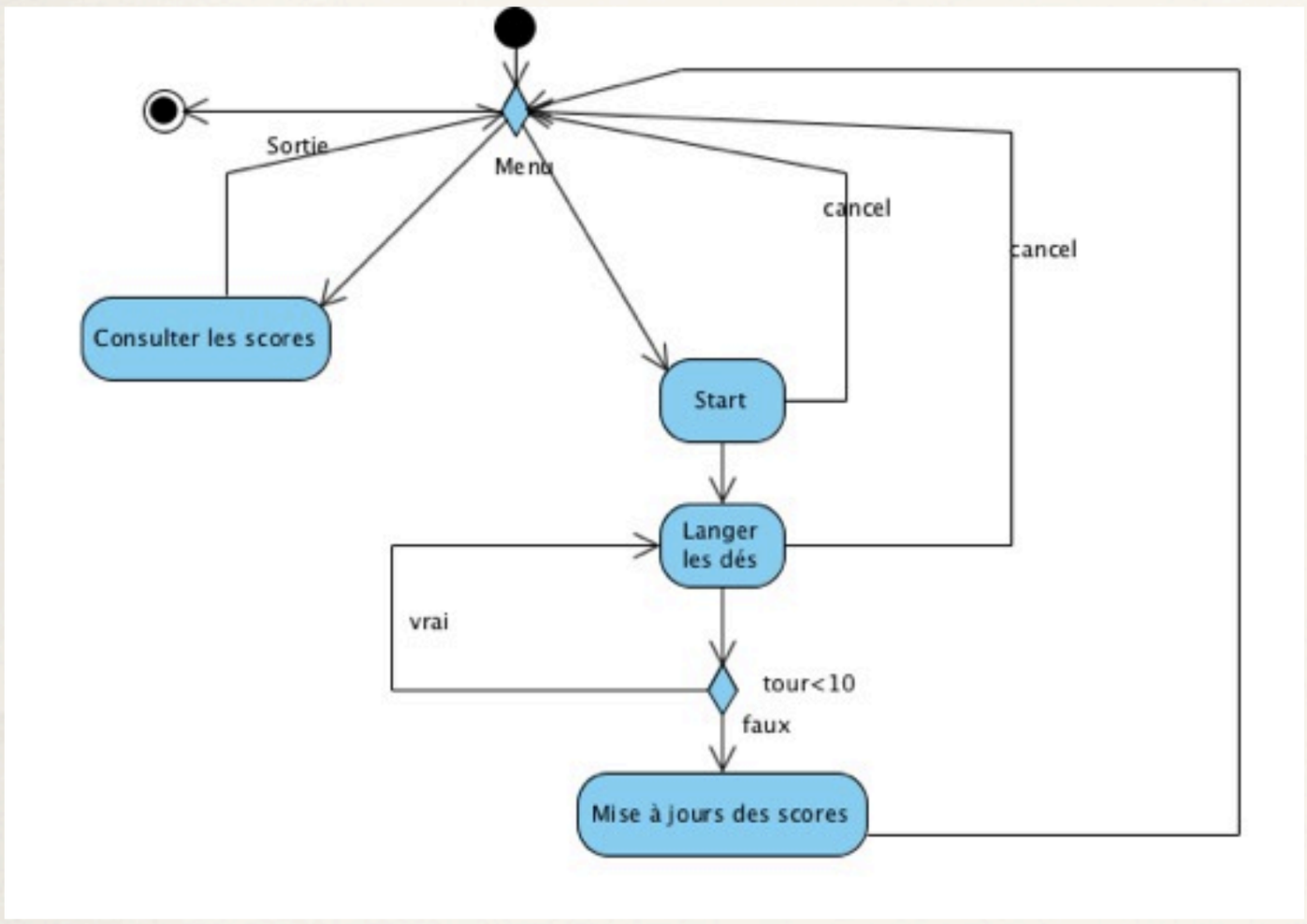


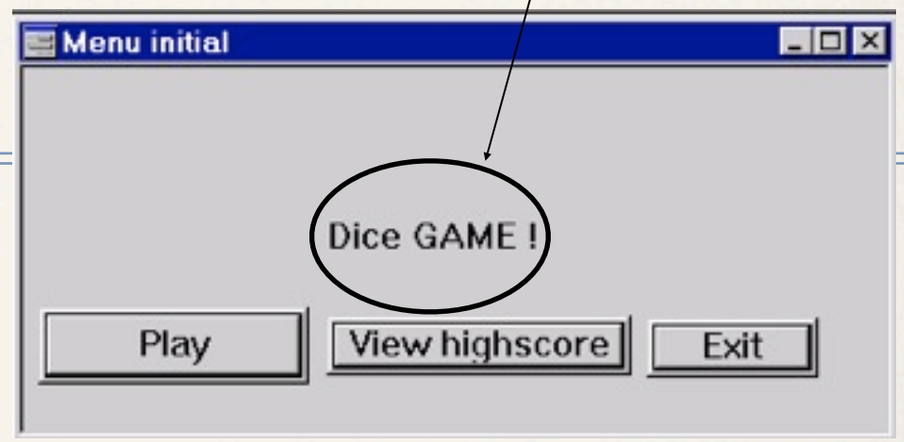
- Ok, les fonctionnalités sont là ...
- et sont conformes au descriptif associé au use case !
- >8->

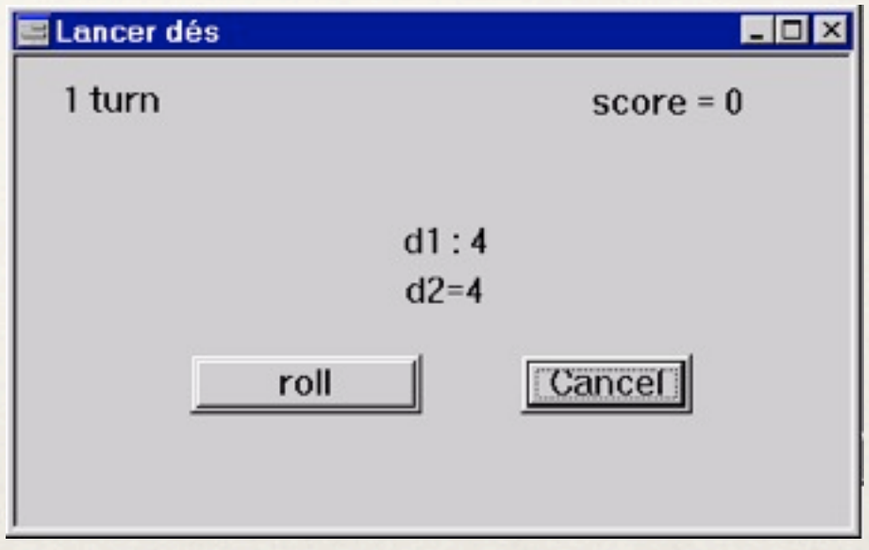
Test du système

Je l ’ai oublié

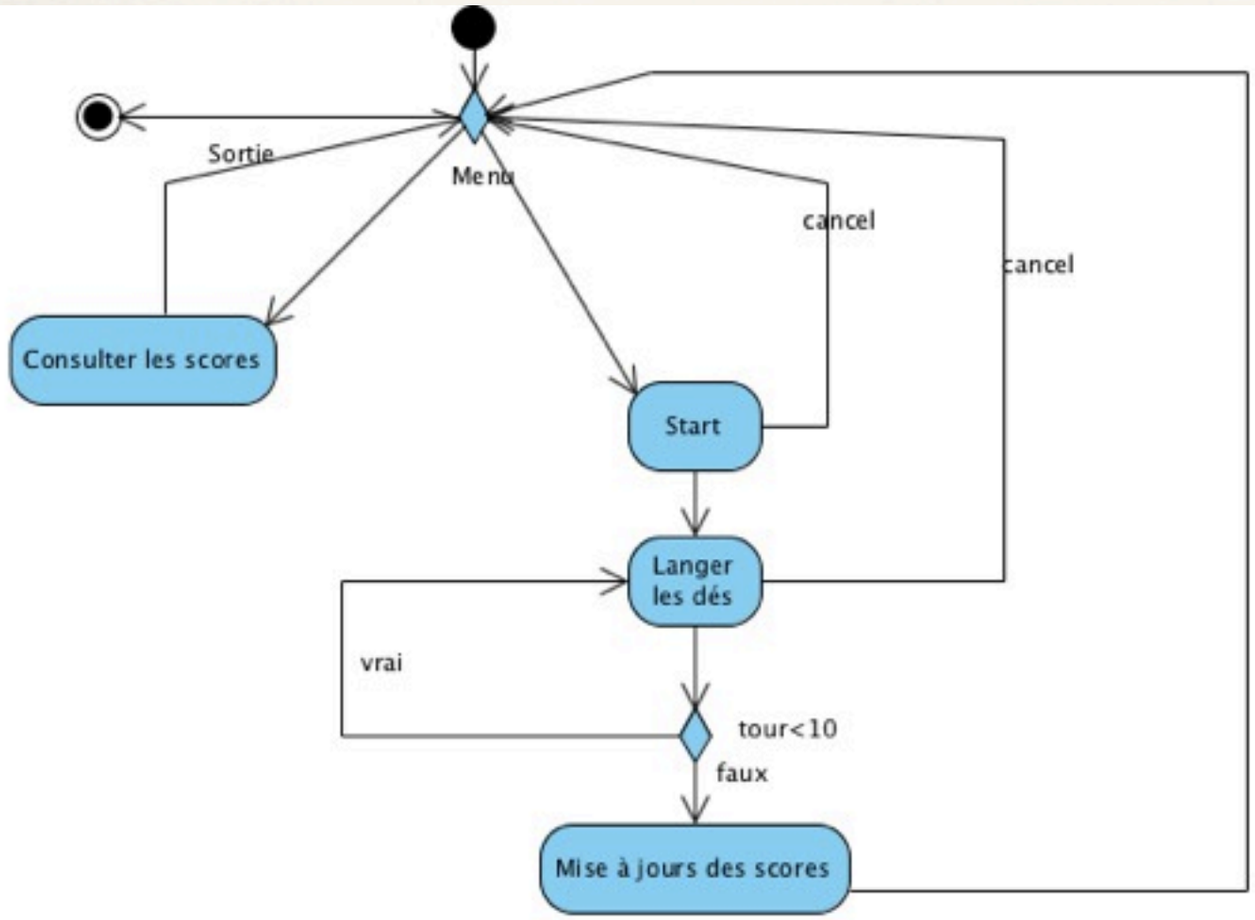
celui là !







Test du système



- Tester tous les chemins possibles !
- Ex:
  - 1/Start
  - 2/ roll
  - 3/ cancel
  - 4/ highscore
  - 5/ exit

Problème rencontré

- Scénario 1 :
  - start, roll*, highscore, quit : OK
- Scénario 2:
  - highscore, : ko ! Bug
  - Pb design :
    - DiceGame crée Highscore (start)
    - Si Highscore avant start : bug

Debriefing de cette application

- Analyse des besoins
  - Use-case + description
  - diagramme d ’activités
  - Prototypage UI
- Analyse
  - Dynamique : Sequence, state
  - Statique : Class Diagram

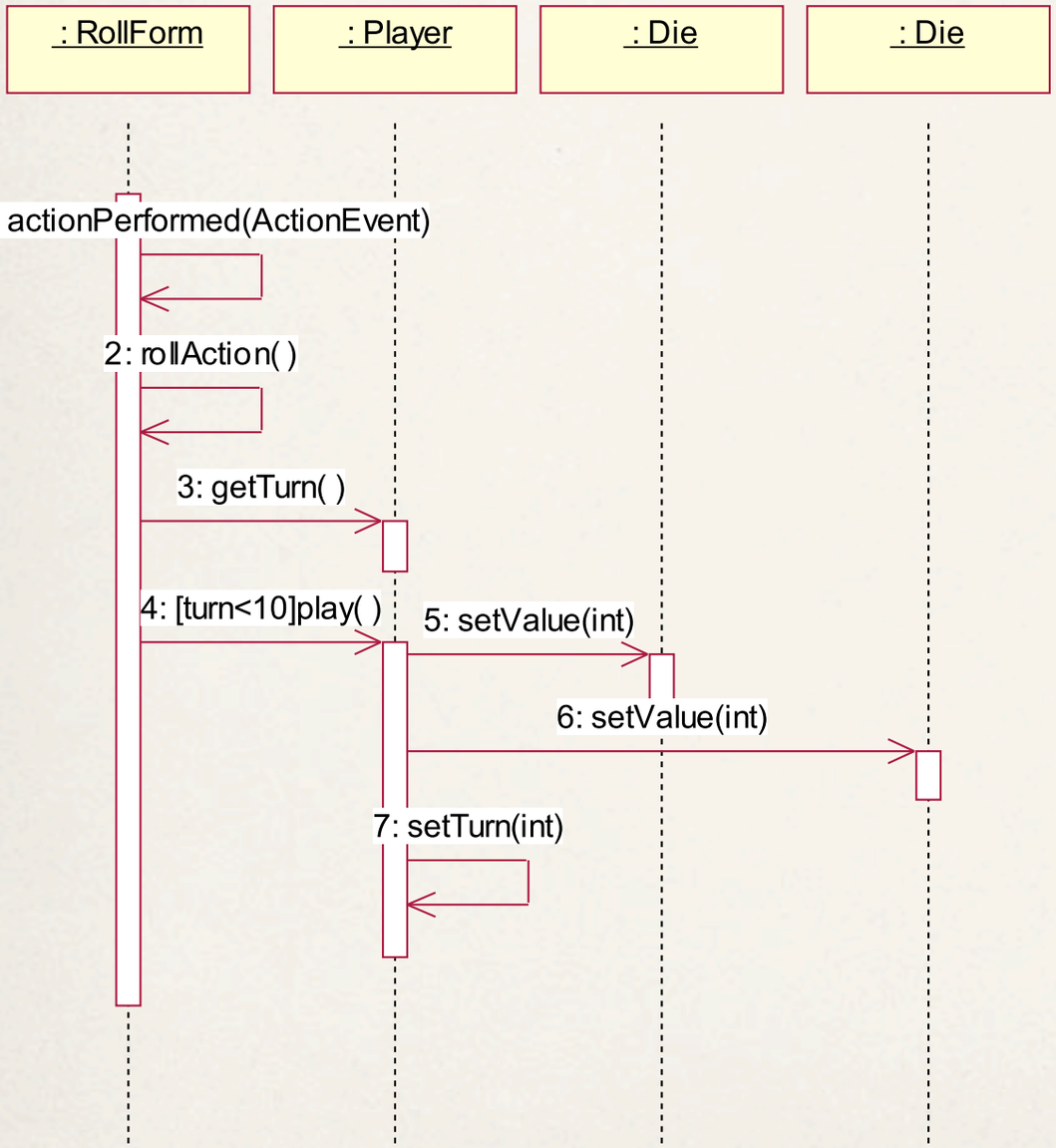
Conception

- Architecture design (layer)
  - Package diagram, deployement diagram
- Classes techniques pour assurer le découpage :
  - Pattern MVC,
  - Pattern Fabrication, ...

Codage

- Simple conversion du design vers Java
- Possibilité de construire pour chaque représentation UML, une traduction vers n ’importe quel langage cible.
- Utilisation des outils pour round-trip engineering
- PB au codage : Bien remettre à jour les document analyse/design !!!

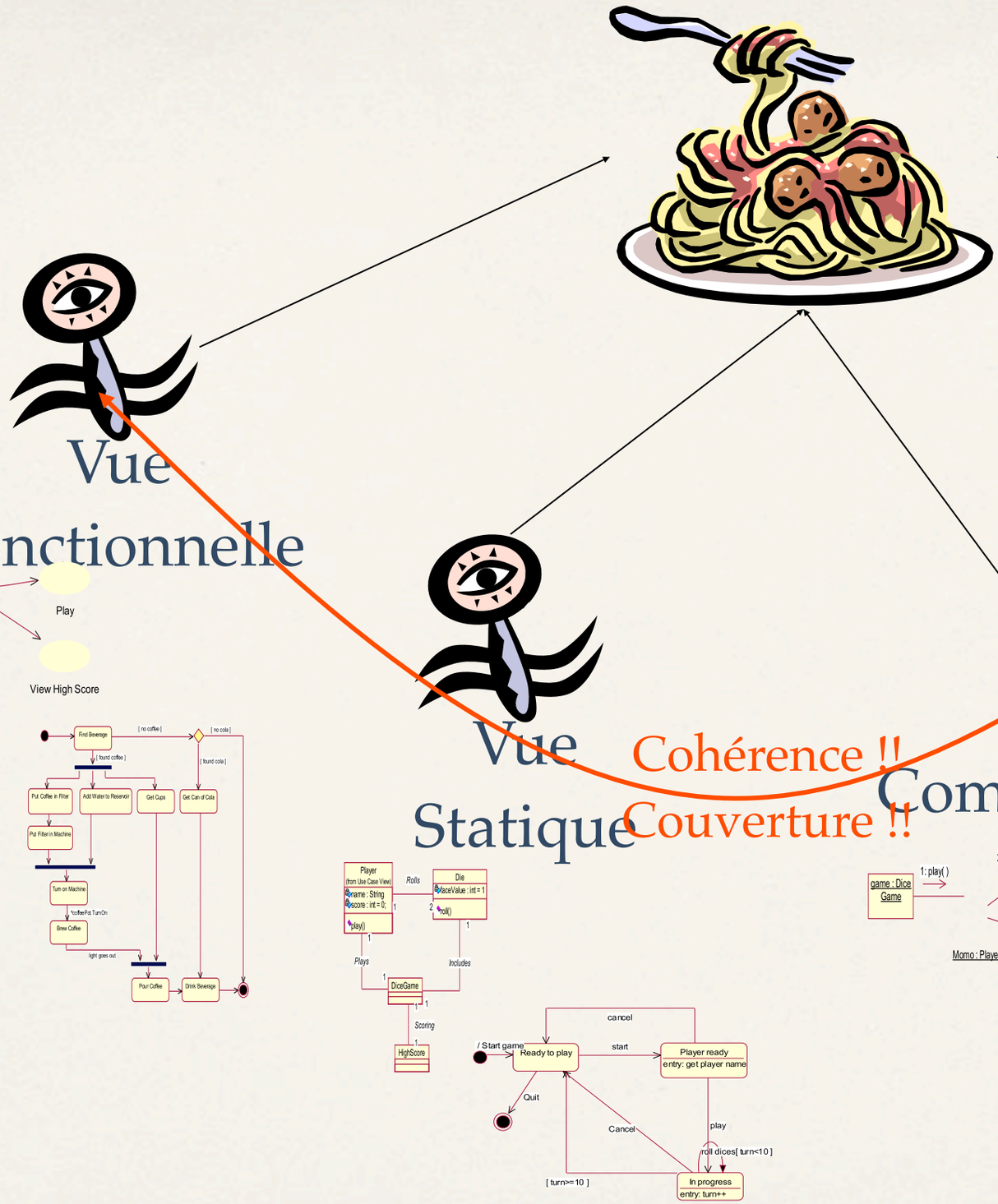
Remonter les PB vus au codage !



- Faire son auto-critique, retrouver la cause de ce pb..
- Améliorer son process pour la prochaine fois !
- Ici : les diag d ’analyse n ’ont pas été refait !
- Software process, procédure qualité !

Conclusion sur cette application

- Découpage en phases:
  - Analyse des besoins, analyse, conception, réalisation, test.
- Dans chaque phase:
  - Prise de points de vue sur le même problème
  - Vue statique, dynamique, fonctionnelle, architecturale



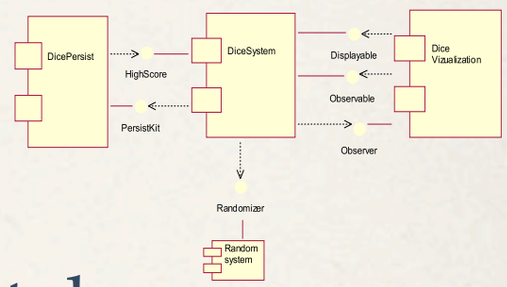
Vue Fonctionnelle

Vue Statique

Cohérence !!

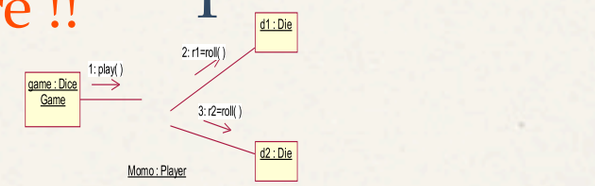
Couverture !!

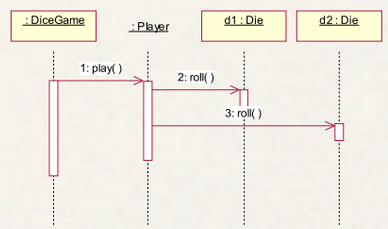
Vue

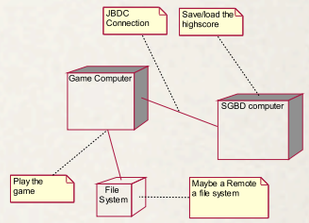


Vue architecturale

Vue Comportementale







Pascal Molli, molli@loria.fr

Cohérence/couverture

- Diagramme Use-cases / Activity
  - Toute activité est assignable à un use-case
  - Tous les use-cases sont réalisés dans les diagrammes d ’activités

d’après Pascal Molli, molli@loria.fr

Séquence/Class diagram

- Tous les objets d ’un diagramme de séquence ont un type : Classe du diagramme de classe
- Les «relations» induites par un diagramme de séquence existent ou peuvent être dérivées du diagramme de classe !
- Les messages échangés sont des méthodes du diagrammes de classes !

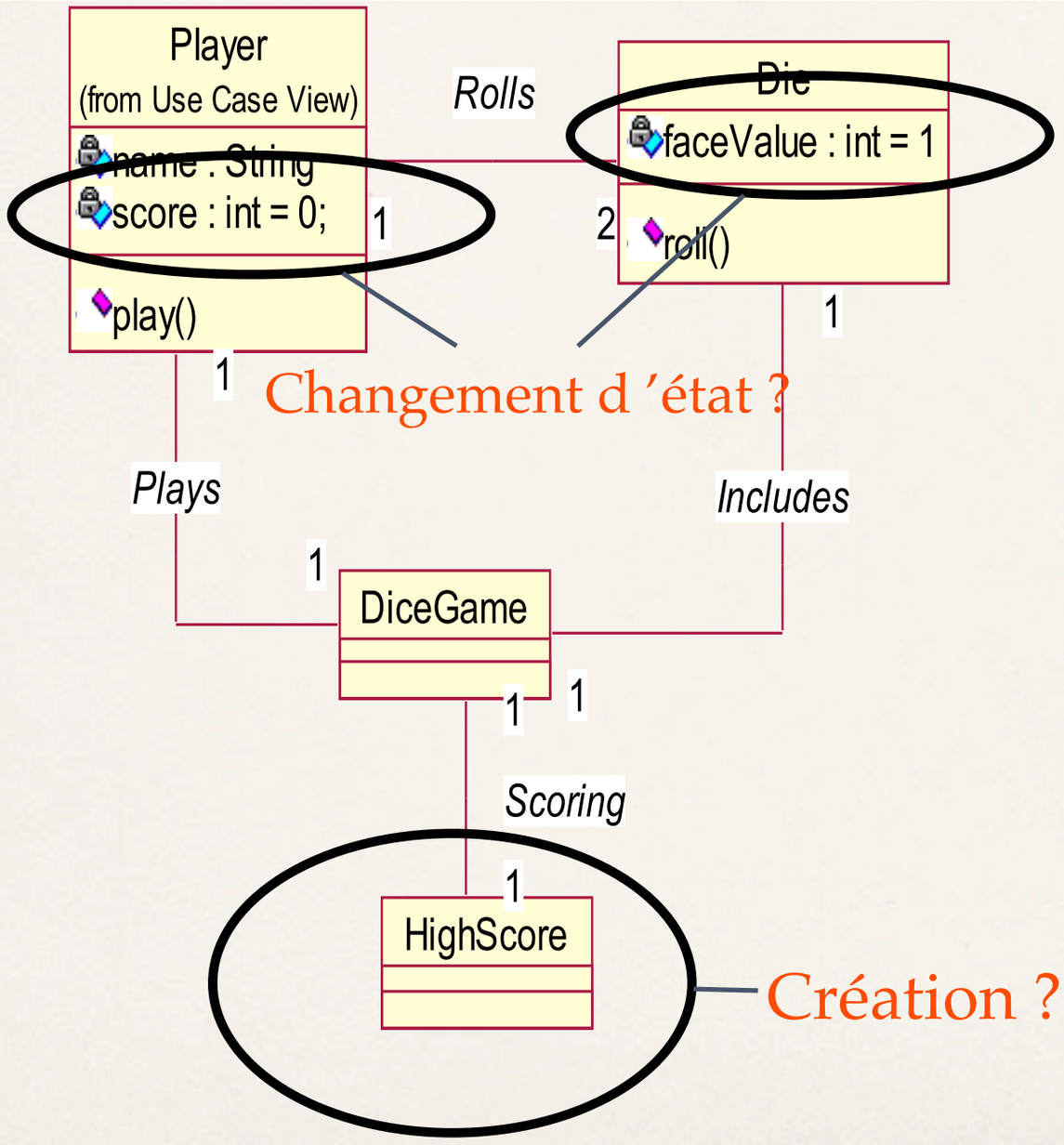
Class diagram / séquence

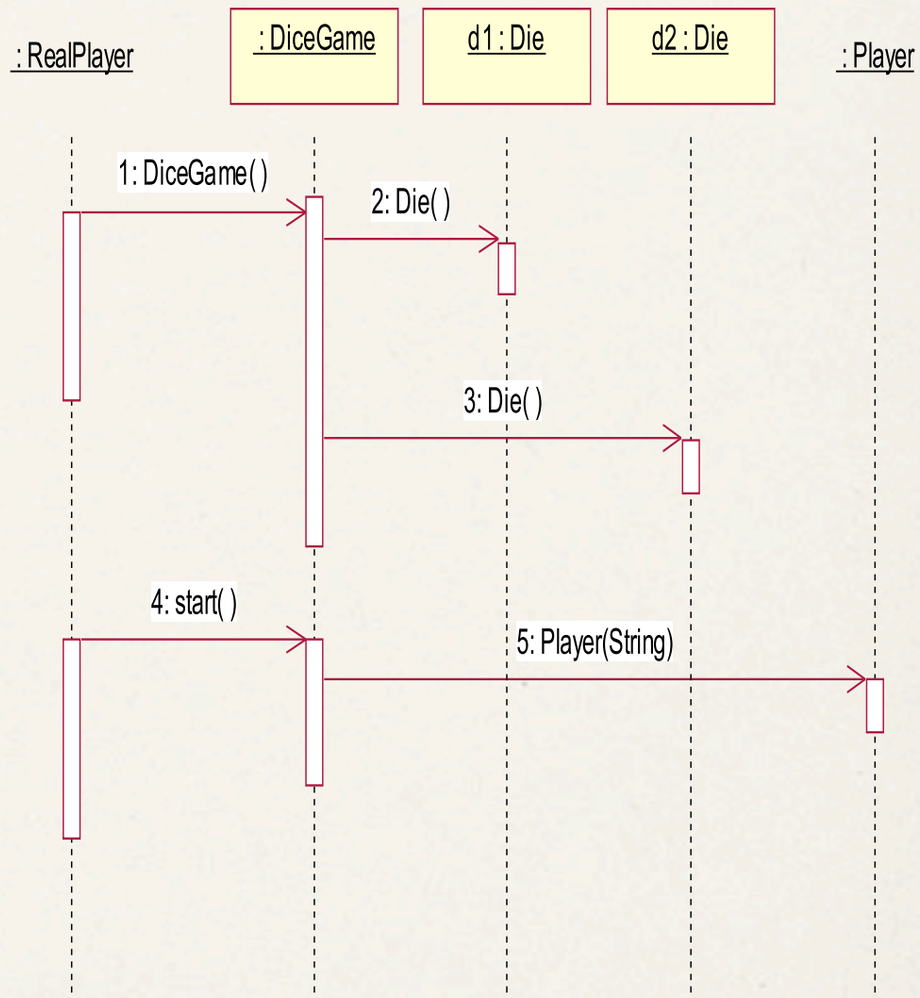
- La dynamique des relations apparaît dans au moins 1 diagramme de séquence ou d'activités
- Tout changement d'attributs est représenté dans au moins 1 diag d 'activités ou de séquence
- Toute création ou destruction d 'objet apparaît dans au moins 1 diag dynamique!

d'après Pascal Molli, molli@loria.fr

Class/Sequence

KO!





Pascal Molli, molli@loria.fr

Class/Package (Design)

- Chaque classe est affectée à un package, package lui-même partie intégrante de l ‘architecture Sinon la classe ne fait pas partie de l ’architecture !

d’après Pascal Molli, molli@loria.fr

Conclusion générale sur l'application «Dice»

- Différence avec une application juste codée ?

- ✓ Elle est documentée, les choix sont justifiés, les tests sont inclus, ...

- ✓ Elle est «évolutive» :
  - ➡ Changement d'IHM, de modèle de persistance
  - ➡ Responsabilité unique des objets (à étudier)
  - ➡ ....

d'après Pascal Molli, molli@loria.fr

Conclusion

- La suite de ce cours doit vous aider à produire des applications de qualité en améliorant vos «développements» de l'analyse à la mise en oeuvre, tel est notre objectif.