MIF17

TP2 : Réingénierie et design patterns en JAVA

Rapport

Berthier Olivier - Groupe 3

Sommaire :

1. Introduction

L'objectif principale de ce TP de génie logiciel est de réaliser une application en utilisant les concepts d'UML 2, c'est à dire qu'au lieux de développer sans phase d'analyse, des diagrammes devront être réalisés et réfléchis avant la phase de développement. La contrainte de ce TP est l'utilisation d'un modèle Model, Vue, Controler.

La réalisation final du TP sera un petit jeu de balle entre plusieurs équipes de tortues. Contrairement au TP précédent, les tortues devront être plus « intelligentes ». L'utilisateur pourra ainsi changer de tactiques de jeu pour chaque équipe au cours du jeu et le comportement de leurs membres devra s'adapter en fonction de cette tactique.

Enfin, des patrons de conceptions seront utilisés au cours de la réalisation de l'application, l'utilisation de ceux-ci sera justifiée dans ce rapport.

Le logiciel de conception UML utilisé est Umbrello, les différentes interfaces seront réalisées sous NetBeans, et enfin les développements seront fait sous Eclipse.

2. Le modèle MVC

La principale contrainte de ce TP est l'utilisation du modèle MVC, cette premier partie aura donc pour objectif de montrer comment adapter la réalisation du TP précédent pour en obtenir une version MVC.

La mise en place de MVC consiste à séparer la vue (ce avec quoi l'utilisateur interagit) des traitements. Dans notre cas, la vue sera donc l'interface de contrôle et d'affichage des tortues de l'application, et le modèle sera tout ce qui concerne le déroulement du match, les actions des tortues (passes, collisions, etc), la gestion des équipes, des rôles, etc.

Le contrôleur ne fait presque rien, il sert principalement à instancier la vue et le modèle et à faire le lien entre eux.

Plutôt que d'adapter le code existant, il a été décidé de tout recommencer, cela permet d'avoir un meilleur contrôle sur l'ensemble du projet.

3. Diagramme de classes

Du fait de l'utilisation de nombreux design patterns, le diagramme de classe final de l'application se révèle assez complexe. D'autant plus que l'application a été pensée pour être un maximum générique, c'est à dire que l'on peu facilement ajouter par exemples de nouvelles interfaces, de nouvelles tactiques, de nouveaux rôles, de nouvelles tortues, etc.

Le premier patron visible dans ce diagramme est MVC entre ControlerJeu, _Model et _Vue. Son utilisation ne se justifie pas que parce qu'il a été imposé. En effet celui-ci permet de garantir une excellente souplesse d'adaptation entre la vue et le modèle lorsque ceux-ci sont bien développés.

Ensuite le second design pattern utilisé est le design pattern Observateur entre la vue et le modèle. La vue et le modèle étant indépendants, cela permet de garantir que la vue sera toujours à jours

lorsque les données du modèle sont modifiées.

Le design pattern Décorateur est utilisé pour la création de tortues ayant différentes caractéristiques. Celui-ci se trouve entre la classe Tortue et AbstractCaracteristique ainsi que tous ses descendants. Chaque classe de caractéristique redéfini certaines méthodes qui sont propres aux tortues pour leur apporter certaines spécificités (ex : vitesse).

Le design pattern de la Fabrication a été utilisé entre la création des modèles (Match ou Solo), et la création des Vues (VueJeuEquipe, VueSolo). Cela permet de créer des vues adaptées pour chaque modèles tout en respectant les contraintes de leur interface respective.

Le design pattern Abstract Factory aurait pu être utilisé entre la création des tortues du modèle et la création des logos de la vue. Cependant il aurait pour cela fallu créer différents logos pour chaque vues. Cela n'est pas impossible à mettre en place, mais dans ce cas le diagramme des classes risque d'être encore plus surchargé pour ne pas apporter grand chose de plus à l'application.

La Fabrique Simple a été utilisée pour la création des terrains et des zones. Dans ce cas la classe implémentant _Model est client, Terrain est la fabrique, et les classes implémentant _Zone sont les produits. L'utilisation de ce design pattern est justifié si l'on souhaite de nouveaux types de zones, il ne sera pas nécessaire de faire beaucoup de modifications.

Le design pattern Facade est visible dans le cas de l'utilisation des tactiques. La méthode bouger() par exemple permet de lancer de nombreuses actions qui sont propres au tortues et aux équipes (par l'intermédiaire de Role).

L'utilisation du Singleton pour la classe VueControle permet d'assurer son unicité. L'application n'utilise en effet qu'une seule interface de contrôle quelque soit le modèle.

Enfin le dernier patron de conception utilisé est le Patron de méthode pour l'application des tactiques. Chaque tactique définissant ses propres règles pour une même méthode et une équipe n'ayant qu'une seul tactique, son utilisation ici est donc justifiée.

Pour ne pas trop surcharger de diagramme de classes, pas toutes les classes du package pGUI de la vue n'ont été modélisées. Celui-ci étant trop volumineux pour pouvoir être intégré au rapport est disponible via le fichier tp2.xmi ou dans le dossier diagrammes.

4. Diagrammes de séquences

Trois diagrammes de séquences seront ici développés :

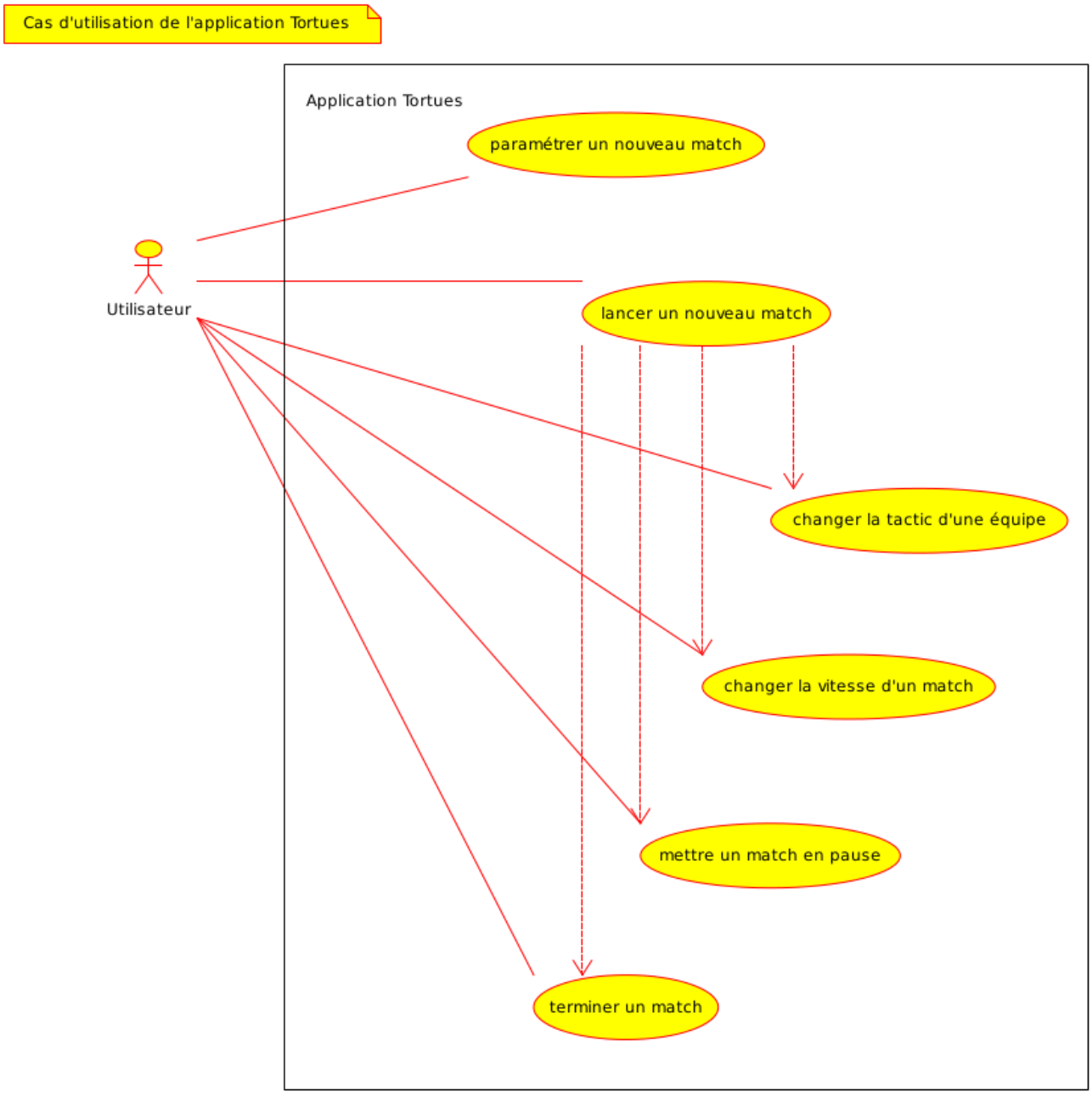
Nouveau match : voir Annexe A.

Voir match : voir Annexe B.

Changer de Stratégie : voir Annexe C.

5. Cas d'utilisation

Le cas d'utilisation est assez simple, celui-ci ne comprend qu'un seul acteur, l'utilisateur, et 6 cas d'utilisation. Certains cas pourraient être décomposées en plusieurs sous cas (pour le paramétrage des matches par exemple), mais cela surchargerai inutilement le cas.



- **Titre** : Utilisation de l'application Tortue V2
- **Objectifs** : Un utilisateur souhaite utiliser l'application tortue pour simuler des matchs entre des équipes de tortues. L'utilisateur peut paramétrer ses matchs ou utiliser les paramètres par défaut.
- **Acteurs** : L'utilisateur de l'application
- **Date** : 20/10/2012
- **Version** : v2.0

- **Pré-conditions** : L'utilisateur a déjàlancé l'application, il dispose pour cela d'une machine virtuelle Java.
- **Post-conditions** : L'utilisateur à pu simuler un ou plusieurs matchs, il peut fermer l'application.
- **Scénario nominal**:

1. L'utilisateur a déjà lancé l'application, l'interface de paramétrage des matchs s'affiche sur son terminal.
2. L'utilisateur saisi la configuration qu'il souhaite simuler :
    1. Il choisi le nombre d'équipes
    2. Il choisi le nombre de joueurs
    3. Il saisi un nom d'équipe pour chaque équipes
    4. Il choisi une couleur pour chaque équipes
5. L'utilisateur clique sur le bouton « Valider »
6. Deux nouvelles fenêtres sont créées : l'une permettant de visualiser le match, l'autre affiche la couleur de l'équipe qui a la balle. Sur l'interface de contrôle un nouvel onglet est créé, celui-ci s'affiche à la place de l'interface de paramétrage des nouveaux matchs.
7. L'utilisateur suit le match
8. L'utilisateur termine le match
9. L'utilisateur ferme l'application

- **Scénario alternatifs** :
    - L'utilisateur souhaite mettre en pause le match :

L'enchaînement démarre au point 7 :

L'utilisateur clique sur bouton « Pause » pour mettre le match en pause

L'utilisateur clique sur le bouton « Start » lorsqu'il souhaite reprendre le match

L'enchaînement revient au point 7

- L'utilisateur souhaite lancer un nouveau match :

L'enchaînement démarre aux points 7 et 8 :

L'utilisateur clique sur l'onglet « Nouveau match »

L'enchaînement reprend au point 2

- L'utilisateur souhaite changer la vitesse de déroulement du jeu :

L'enchaînement démarre au point 7 :

L'utilisateur utilise le slider pour modifier la vitesse de déroulement du match

L'enchaînement reprend au point 7

- L'utilisateur souhaite changer la tactique d'une équipe

L'enchaînement démarre au point 7 :

L'utilisateur sélectionne une tactique dans la liste des tactiques de chaque équipe

L'enchaînement reprend au point 7

- **Scénario d'exceptions** :
  - L'utilisateur souhaite quitter l'application sans avoir lancé de match :

Le scénario démarre au point 1 :

L'utilisateur ferme l'application.

  - L'utilisateur souhaite quitter l'application sans avoir fini les matchs en cours :

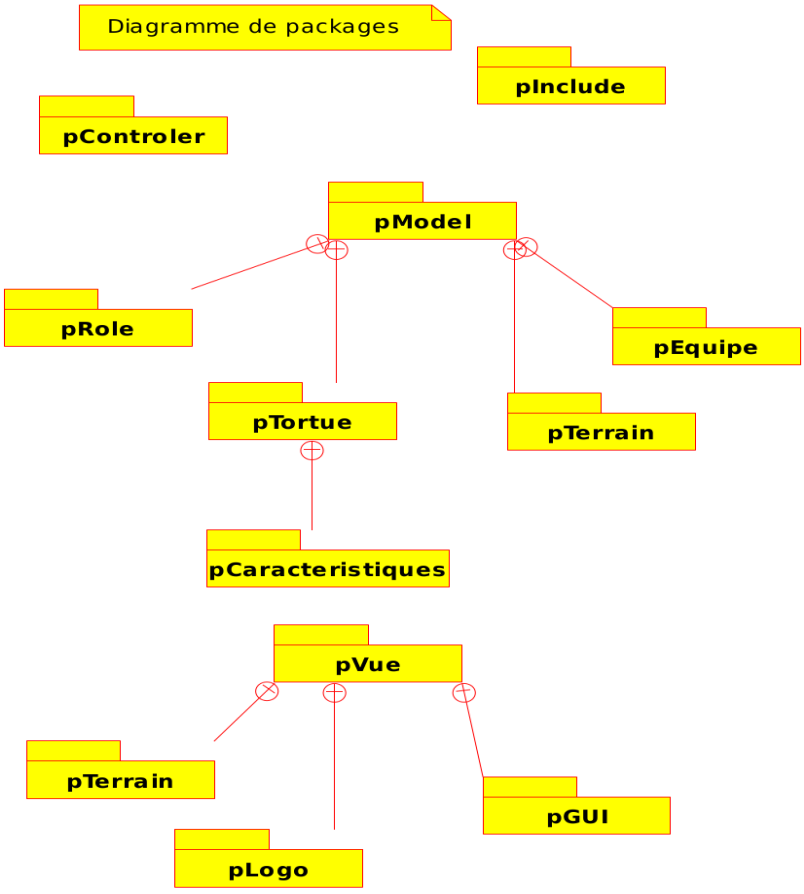
Le scénario démarre au point 7 :

L'utilisateur ferme l'application via l'interface de contrôle.

6. Les packages

Pour plus de clarté, les différentes classes et interfaces on été regroupées en plusieurs packages. On trouve principalement un package pModel, qui contient l'ensemble des classes du modèle, un package pVue, qui regroupe les classes de la vue, un package pControler qui ne contient que la classe du contrôleur, et un package pInclude qui regroupe toutes les autres classes. Les package pModel et pVue sont divisés en plusieurs sous packages, par exemple pTerrain, pTortue, pEquipe, etc pour le modèle et pLogo, pGUI, etc pour la vue.

Ce regroupement a été fait pour regrouper les classes de même structures ou ayant un fort lien entre elles.



7. Outils de conception

Le principal logiciel utilisé pour la modélisation est Umbrello. Il s'agit d'un logiciel libre assez performant dans le domaine de la modélisation. Il permet en effet de réaliser divers diagrammes de la norme UML 2 qui seront enregistré dans un même fichier XMI. Il est ainsi possible de réutiliser les classes et méthodes qui ont déjà été défini dans d'autres diagrammes. De plus Umbrello permet d'enregistré le code à implémenter pour les divers méthodes, ainsi que des commentaires. Cependant j'ai assez vite renoncé à cela car lors de la génération du code Java, si le code et les commentaires ajoutés sont bien présents, le code n'est pas vraiment utilisables. En effet, par exemple certaines relations précédemment supprimées apparaissent toujours. Il faut donc à chaque génération du code depuis le diagramme revoir tout le code, ce qui n'est pas efficace au final. De plus ce logiciel semble parfois rencontrer certaines erreurs de segmentation, il faut donc tout rouvrir. Je pense que je utiliserai un autre logiciel à la prochaine occasion, car au lieu de faire évoluer mon code à partir de mes diagrammes, il m'a souvent fallu faire l'inverse à partir d'un certain momment.

Pour ce qui est de l'IHM, celle-ci a entièrement été maquetté sous NetBeans. Le code a ensuite été adapté pour essayer d'obtenir une génération dynamique en fonction du nombre d'équipes, mais cela

n'est pas aussi facile que pour une page WEB, cela n'a pas pu être terminé. La version du programme livrée ne pourra donc avoir que deux équipes. Enfin le code de l'application a été implémenté sous Eclipse.

8. Évolutions

Bien que la version du programme fournie ne soit pas complètement finie, de nombreuses pistes d'améliorations ont été explorées au cour du développement.

Premièrement, comme cela a été expliquée précédemment, il est possible d'utiliser un design pattern Abstract Factory pour pouvoir créer différents types de logo pour les différents types de tortues de chaque modèle.

Les autres améliorations explorées sont plutôt techniques. Par exemple il serait possible de gérer les paramètres dans un fichier XML à part, cela permet d'éviter de stocker des données en dur dans le code. Il serait aussi possible de gérer les rôles et les tactiques en Prolog par l'intermédiaire de la librairie GNU Prolog for Java. Cependant cela demanderai en plus du temps passé pour le développement un certain temps pour la création des règles de gestion.

Pour plus de confort, il serait par exemple possible d'utiliser une règle logarithmique pour le slider de sélection de la vitesse d'exécution. Il aurait aussi été possible de gérer des tournois (d'où la présence des boutons Enregistrer et Ouvrir du menu Fichier, permettant de reprendre un tournoi en cours).

Enfin, de nombreuses autres améliorations pourraient encore êtres prévus, comme le fait de ne pas mettre l'enbut au milieu de la zone de l'équipe, pouvoir afficher la balle qui se déplace lorsque celle-ci est lancée d'une tortue à une autre, revoir la collision entre les tortues, ...

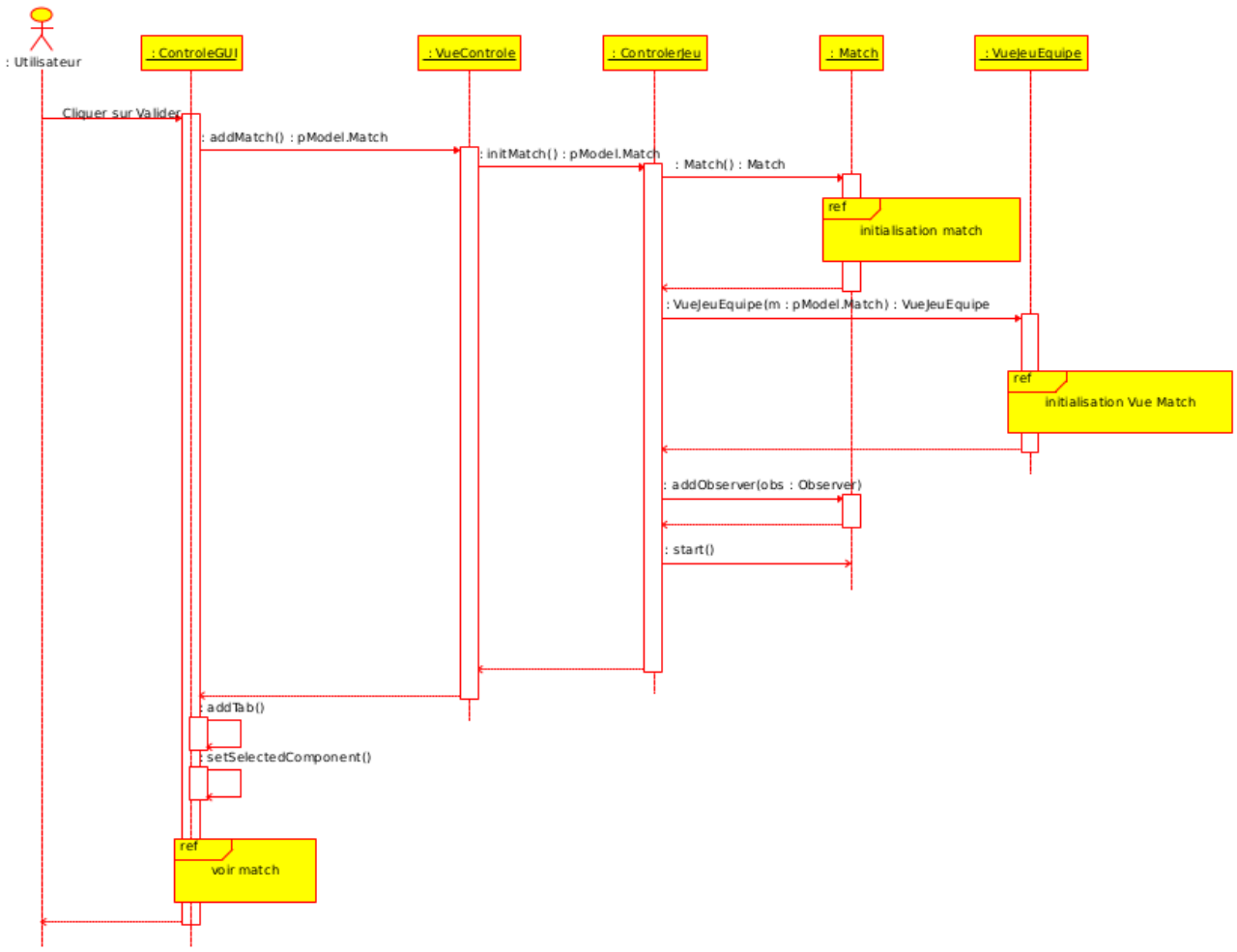
9. Conclusion

Contrairement au TP précédent, l'effort a été poussé à l'utilisation des design patterns et des techniques de conceptions d'UML 2. Malgré quelques difficultés techniques lié à l'utilisation d'Umbrello, cela a pu être fait. En effet, l'application a entièrement été repensée pour être un maximum générique.

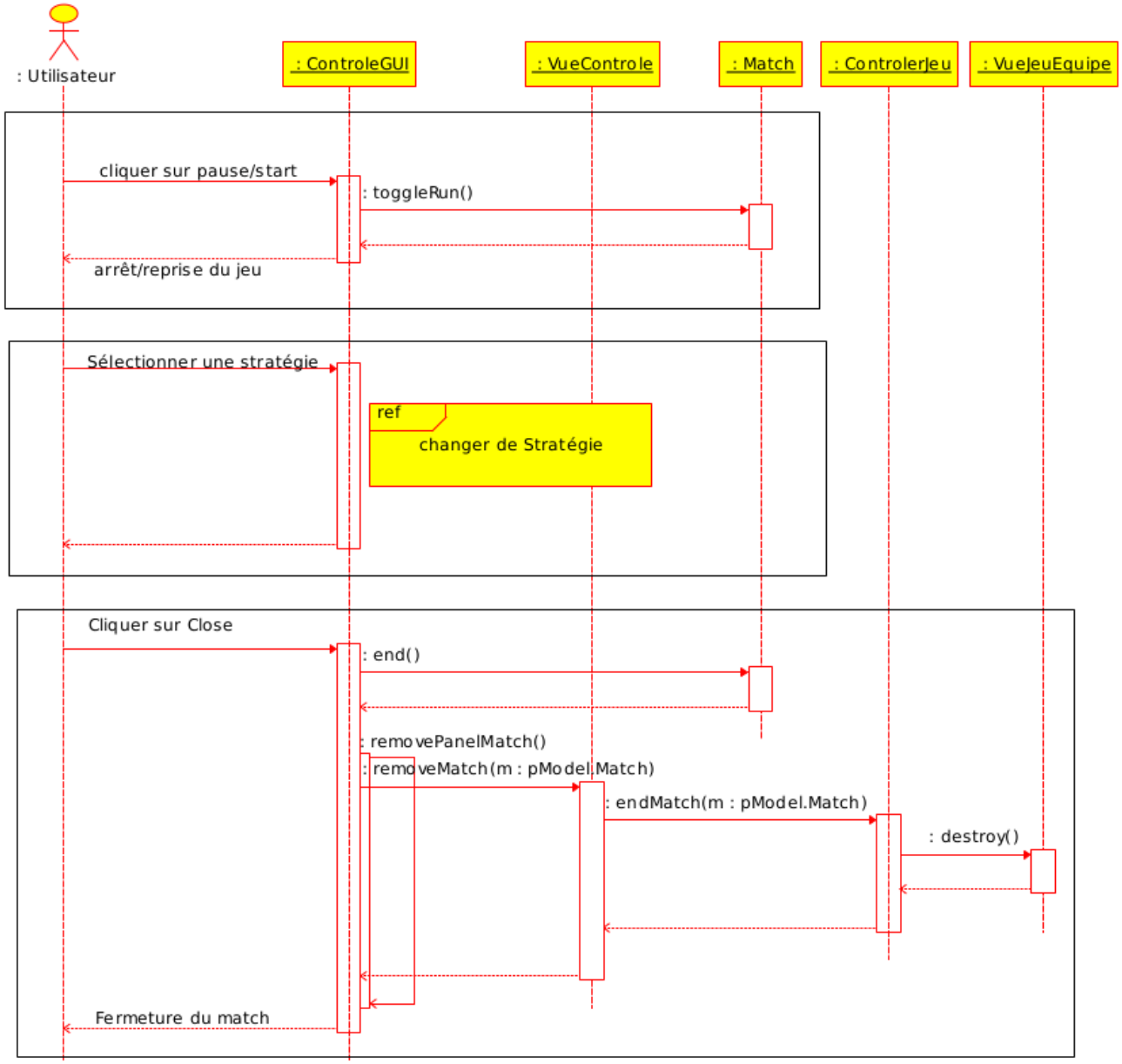
En théorie, si la génération de code était parfaitement fonctionnelle, il suffirait d'ajouter les classes souhaitées (par exemple concevoir un nouveau modèle avec de nouvelles tortues) sur le diagramme, puis d'ajouter le code nécessaire via l'application de modélisation pour obtenir une application fonctionnelle. Malheureusement l'outil de modélisation choisi ne permet pas d'arriver à cela.

Au final, l'application est en grande partie fonctionnelle, même si une partie de ce qui a été prévu au départ n'a pas pu être fait par manque de temps, comme par exemple la gestion de plusieurs équipes, ce qui nécessite une génération dynamique de l'interface de contrôle et une méthode de division du terrain en zones d'équipes plus évoluée.

ANNEXE A :


: Utilisateur
: ControleGUI
: VueControle
: ControlerJeu
: Match
: VueJeuEquipe
Cliquer sur Valider
: addMatch() : pModel.Match
: initMatch() : pModel.Match
: Match() : Match
ref
initialisation match
: VueJeuEquipe(m : pModel.Match) : VueJeuEquipe
ref
initialisation Vue Match
: addObserver(obs : Observer)
: start()
: addTab()
: setSelectedComponent()
ref
voir match

ANNEXE B :



ANNEXE C :