Génie Logiciel et Patrons de Conception
Regis.Clouard@ensicaen.fr
Julien.Mahier@ensicaen.fr
Sylvain.Vernois@ensicaen.fr
2019-2020

Jeu en réseau de tir à l'arbalète

Ce projet vise à la mise en œuvre des techniques de génie logiciel pour la réalisation d'un projet concret. Une attention particulière sera donc accordée à la qualité de la conception de votre logiciel. Il est difficile dans le cadre de séances de travaux pratiques de proposer un projet d'une taille suffisante pour se rendre compte qu'une conception mal structurée conduit inévitablement à un logiciel rigide, fragile et immobile qui vire à l'usine à gaz. Compte-tenu des contraintes de temps, nous ne pouvons que proposer un projet modeste et qui pourrait être conçu sans compétences particulières en génie logiciel. Néanmoins, dans un but pédagogique, il vous est demandé de bien vouloir jouer le jeu de la gestion de projet et de livrer un logiciel démontrant des qualités de robustesse, extensibilité et réutilisabilité, accompagné de ses tests unitaires. Un projet ne présentant pas toutes les fonctionnalités demandées mais conçu selon les règles de l'art sera beaucoup mieux évalué qu'un projet complètement fonctionnel mais bricolé.

Éléments du sujet

1 Le sujet....2
1.1 Description du jeu....2
1.2 Exigences minimales du client....2
2 Organisation du projet....3
2.1 Contraintes sur la réalisation du projet....3
1.1 Méthode de gestion de projet....3
1.2 Rôles et fonctions des coéquipiers....4
2.2 Livrables....5
2.3 Soutenance....5
2.4 Évaluation....6
3 Ressources....7
1.3 Exemples d'un projet complet....7
1.4 Documentations sur les outils à utiliser....7

1 Le sujet

Vous venez de créer une entreprise dans le domaine du jeu vidéo avec une offre spécialisée dans les jeux en réseau. Un client (rôle joué par votre encadrant de TP) qui représente une célèbre entreprise de fabrication d'accessoires et d'outils pour l'archerie vous demande de concevoir pour son activité de promotion, un jeu vidéo de tir à l'arbalète en réseau.

1.1 Description du jeu

Une compétition de tir à l'arbalète fait s'affronter deux archers en deux fois 10 volées de 3 flèches, soit 60 flèches. La cible est découpée en 10 cercles concentriques. La valeur 1 est attribuée au premier cercle et la valeur 10 au centre. Toute flèche en dehors de la cible ne sera pas comptabilisée. Le comptage des points est la somme des valeurs des cercles atteints par chaque flèche. Plusieurs distances et tailles de cible existent dans le règlement officiel, par exemple une cible de 122 cm de diamètre placée à une distance de 90 m.

1.2 Exigences minimales du client

Le client souhaite les fonctionnalités suivantes :

1. Chaque joueur accède au jeu via un ordinateur en réseau. Un ordinateur prend le rôle du serveur où viennent se connecter les deux participants en tant que clients.

2. Une partie consiste en 5 volées de 2 flèches soit 10 flèches par joueur.

3. Pour tirer sa flèche, le joueur doit construire le vecteur vitesse de la flèche à l'aide d'un moyen d'interaction qui peut être la souris ou le clavier.

4. Au moment de tirer sa flèche, le joueur doit avoir un affichage adéquat lui permettant d'ajuster son tir.

5. Pendant qu'un joueur tire sa flèche, l'autre joueur doit avoir un affichage rapproché de la cible de son adversaire avec les impacts des flèches.

6. L'application est responsable des lancers, de l'affichage du jeu, de l'affichage du score et de l'application du règlement.

2 Organisation du projet

La réalisation et l'organisation du travail sont soumises à des contraintes que chaque groupe devra respecter.

2.1 Contraintes sur la réalisation du projet

- ☑ **UML :** Les modélisations seront réalisées avec le langage UML.
- ☑ **JAVA** : Le langage d'implémentation sera le Java.
- ☑ **JAVAFX** : L'interface graphique se basera intégralement sur la bibliothèque JavaFX.
- ☑ **Test** : Le code source du projet devra s'accompagner de **tests unitaires** et de **tests d'intégration** développés avec JUnit et potentiellement Mockito.
- ☑ **Gradle** : le projet sera géré par le moteur de production Gradle.
- ☑ **Git** : La gestion des versions sera réalisée avec Git. Le dépôt central sera localisé sur gitlab.ecole.ensicaen.fr. La récupération des livrables finaux sera faite par votre encadrant directement à partir du dépôt Gitlab dont vous lui communiquerez l'adresse[1].
- ☑ **Gitlab CI** : L'intégration continue sera gérée par Gitlab[2]. Pour cela, un fichier yaml (`.gitlab-ci.yml`) vous est fourni dans la documentation. L'intégration devra être faite avec le « runner » installé sur le Gitlab de l'école.

1.1 Méthode de gestion de projet

Le projet sera réalisé par équipe de 8 personnes et se déroulera sur **8 séances de 2 heures**.

Il vous est demandé d'utiliser une méthode de gestion de projet Agile à base de **cycles de développement itératifs et incrémentaux**.

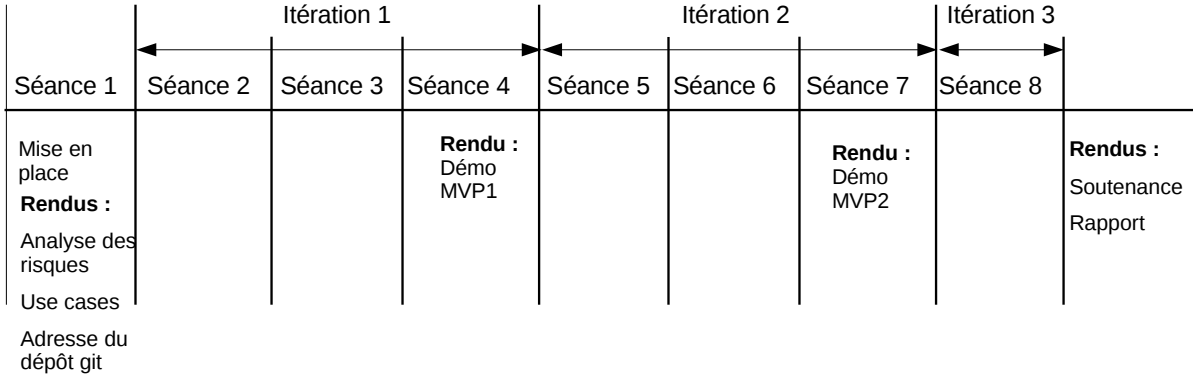
- ➢ Une itération durera 3 séances et donc il n'y aura que 2 itérations.
- ➢ Le projet commence par une première séance de définition du premier MVP avec la liste de ses fonctionnalités. Cette phase associe normalement pleinement le client ; ici, votre client viendra simplement valider ce que vous aurez prévu.
- ➢ Chaque itération commence par la définition d'un sous-ensemble de tâches à réaliser (de granularité assez fine) qui correspondent aux fonctionnalités identifiées pour cette itération : le MVP. La liste des tâches doit être ajustée pour 2 séances de TP !

1 N'oublierez pas d'ajouter votre encadrant votre dépôt avec le rôle développeur afin qu'il puisse suivre votre progression.
2 Voir la documentation relative sur la plateforme pédagogique.

- Au cours de l'itération, les tâches sont réalisées par les développeurs de l'équipe avec une étanchéité la plus élevée possible entre les développeurs.

- Chaque itération se termine par une **démonstration** du prototype opérationnel auprès de votre client. Cette démonstration est un moment d'évaluation par votre encadrant de votre capacité à concevoir le logiciel demandé. Elle doit donc être préparée et réalisée de manière formelle. Le trop fréquent « effet démo » est ici une erreur qui sera sanctionnée.

- La démonstration s'accompagne ensuite d'une rétrospective qui vise à réviser le plan de développement en supprimant, ajoutant ou changeant les fonctionnalités de la liste prévue avant d'entamer l'itération suivante.



1.2 Rôles et fonctions des coéquipiers

Chaque équipe doit répartir les responsabilités en nommant des personnes aux postes suivants :

- ☑ **Chef de projet** : son rôle est de coordonner les activités de l'équipe, de superviser les travaux et d'interagir avec le client.

- ☑ **Architecte** : son rôle est de concevoir l'architecture du logiciel, l'organisation de la conception en paquets et de prendre les décisions tactiques relatives à cette réalisation.

- ☑ **Développeur** : son rôle est de concevoir les classes et leur organisation pour réaliser les fonctionnalités qui lui sont attribuées. Le développeur doit aussi écrire les tests unitaires correspondants (*pourquoi pas en TDD*).

- ☑ **Responsable de version** : cet ingénieur est le responsable du dépôt sur Gitlab. C'est lui qui fait l'intégration continue du travail des développeurs et réalise le prototype de démonstration qui sera présenté à l'issue de chaque itération. Il est aussi le garant de la propreté du code et de la présence des tests dans les contributions des développeurs.

Il est à noter qu'une même personne peut endosser plusieurs rôles et qu'un rôle peut être assumé par plusieurs personnes.

2.2 Livrables

Premier rendu au début de la séance 2

- ☑ **Analyse des risques** : la liste des risques majeurs de ne pas parvenir à développer le logiciel et les moyens de les prévenir ou d'y remédier.
- ☑ **Cas d'utilisation** : l'analyse des besoins sur la forme d'un diagramme des cas d'utilisation.
- ☑ **Dépôt git** : l'adresse gitlab du projet.
- ☑ **Répartition des rôles** au sein de l'équipe.

Rendu final

Différents documents, regroupés dans un rapport, devront être produits :

- ☑ **Analyse des risques** : reprise de l'analyse faite en première séance.
- ☑ **Cas d'utilisation** : le diagramme des cas d'utilisation du logiciel.
- ☑ **Conception UML** : la modélisation du logiciel en UML où les patrons de conception seront mis en exergue.
- ☑ **Diagramme de paquet** : uniquement les paquets et leurs dépendances.
- ☑ **Code source** : le projet Gradle avec le code Java du logiciel et le code des tests unitaires et des tests d'intégration.

Cette liste n'est en aucun cas exhaustive et pourra être complétée avec d'autres documents qui vous paraîtront pertinents dans la limite du raisonnable (au sens Agile du terme).

2.3 Soutenance

À l'issue des huit séances de travail, une soutenance sera organisée pour la validation finale du logiciel. La soutenance sera découpée en 10 minutes de présentation, 5 minutes de démonstration et 10 minutes de question. La soutenance devra au moins faire apparaître clairement :

- ☑ L'architecture de votre système.
- ☑ La réponse de votre conception à la robustesse, la réutilisabilité et la maintenance.
- ☑ L'organisation de l'équipe et la gestion de projet conduite.

La soutenance sera conclue par une démonstration finale.

Si la planification de votre projet s'est avérée irréaliste et que l'avancement réel en est très éloigné, il vous est demandé de prendre du recul et d'analyser les causes du dysfonctionnement. Le travail d'une équipe qui saurait analyser les dysfonctionnements

et proposer des solutions sera bien mieux apprécié que celui d'une équipe qui chercherait à dissimuler d'éventuelles déconvenues derrière un produit fini, même spectaculaire.

Lors de la présentation orale, tous les membres de l'équipe doivent présenter l'aspect du projet dont ils sont responsables. Il ne sera pas accepté qu'un seule personne présente l'ensemble du projet de l'entreprise.

2.4 Évaluation

La note de TP de chaque membre d'une équipe sera la moyenne des notes obtenues pour chacune des quatre évaluations :

- ☑ la pertinence de la conception ;
- ☑ la « propreté » du code et des tests ;
- ☑ la qualité de la soutenance ;
- ☑ la perception du client sur votre capacité à produire le logiciel demandé.

3 Ressources

Afin de vous aider dans la réalisation de votre projet vous trouverez sur la plateforme pédagogique plusieurs ressources documentaires et matérielles.

L'ensemble des ressources proposées est conforme à l'IDE IntelliJ IDEA, que nous vous encourageons à utiliser.

1.3 Exemple d'un projet intégrant toutes les technologies demandées

- ☑ Un projet Gradle avec un exemple d'interface graphique en JavaFX basée sur le patron d'architecture MVP[3] répartie sur une architecture client-serveur utilisant les sockets. Le projet intègre des exemples de test écrits en JUnit 4 avec la bibliothèque Mockito. Un fichier yml réalise l'intégration continue sur Gitlab.

1.4 Documentations sur les outils à utiliser

- ☑ Pour le responsable de version : création d'un dépôt Gitlab.
- ☑ Le travail de développement de projet avec Git (Gitlab Workflow).
- ☑ Les tests unitaires avec JUnit.
- ☑ Les doublures avec Mockito.

3 Modèle - Vue - Présentation