Hockedu

Document d'architecture logicielle

Version 1.0

Historique des révisions

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Auteur** |
| 2013-02-08 | 1.0 | Version initiale | Mathieu M-Gosselin |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table des matières

[1. Introduction 6](#_Toc348028914)

[2. Objectifs et contraintes architecturaux 6](#_Toc348028915)

[2.1 Séparations (façades) entre les différents modules 6](#_Toc348028916)

[2.2 Structure en arbre pour les éléments de jeu 6](#_Toc348028917)

[2.3 Machine à états pour les différentes situations du client et du serveur 6](#_Toc348028918)

[2.4 Utilisation d’usines pour la création d’éléments spécifiques 6](#_Toc348028919)

[2.5 Utilisation de visiteurs pour la modification des nœuds 6](#_Toc348028920)

[2.6 Architecture multi-fils 6](#_Toc348028921)

[2.7 Les paquets reçus sont « exécutables » sur le thread principal 7](#_Toc348028922)

[2.8 Exceptions et système de journal 7](#_Toc348028923)

[2.9 Authentification de l’utilisateur 7](#_Toc348028924)

[2.10 Utilisation de *shared\_ptr* 7](#_Toc348028925)

[2.11 Objective C++ 7](#_Toc348028926)

[2.12 Échéancier 8](#_Toc348028927)

[2.13 Taille et performance 8](#_Toc348028928)

[3. Vue des cas d’utilisation 9](#_Toc348028929)

[4. Vue logique 15](#_Toc348028930)

[4.1 Description des paquetages 15](#_Toc348028931)

[4.2 Diagramme de paquetages 20](#_Toc348028932)

[4.3 Diagrammes de classes 21](#_Toc348028933)

[4.3.1 Couche Présentation 21](#_Toc348028934)

[4.3.1.1 Interface Lourd 21](#_Toc348028935)

[4.3.1.2 Interface Léger 21](#_Toc348028936)

[4.3.2 Couche Modèle 22](#_Toc348028937)

[4.3.2.1 Jeu 22](#_Toc348028938)

[4.3.2.1.1 Facade 23](#_Toc348028939)

[4.3.2.1.2 Réseau 23](#_Toc348028940)

[4.3.2.1.3 Partie 24](#_Toc348028941)

[4.3.2.1.4 Terrain 24](#_Toc348028942)

[4.3.2.1.5 Modèle3D 25](#_Toc348028943)

[4.3.2.2 Éditeur Lourd 25](#_Toc348028944)

[4.3.2.3 Éditeur Léger 26](#_Toc348028945)

[5. Vue des processus 27](#_Toc348028946)

[5.1 Gerer un Tournoi 27](#_Toc348028947)

[5.1.1 Choisir les joueurs 28](#_Toc348028948)

[5.1.2 Choisir une carte 29](#_Toc348028949)

[5.1.3 Choisir le nombre de joueurs 30](#_Toc348028950)

[5.1.4 Sauvegarder tournoi 31](#_Toc348028951)

[5.1.5 Charger tournoi 32](#_Toc348028952)

[5.2 Gérer les cartes 33](#_Toc348028953)

[5.2.1 Créer une nouvelle carte 34](#_Toc348028954)

[5.2.2 Modifier une carte 35](#_Toc348028955)

[5.2.2.1 Manipuler des objets 36](#_Toc348028956)

[5.2.2.1.1 Ajouter un objet 37](#_Toc348028957)

[5.2.2.1.2 Supprimer un objet 38](#_Toc348028958)

[5.2.2.1.3 Dupliquer un objet 39](#_Toc348028959)

[5.2.2.1.4 Déplacer un objet 40](#_Toc348028960)

[5.2.2.1.5 Mettre un objet à l’échelle 40](#_Toc348028961)

[5.2.2.1.6 Modifier les propriétés physique d’un objet 41](#_Toc348028962)

[5.2.2.1.7 Sélectionner un objet 42](#_Toc348028963)

[5.2.2.2 Annuler une action 43](#_Toc348028964)

[5.2.2.3 Refaire une action 44](#_Toc348028965)

[5.2.2.4 Manipuler la surface de jeu 45](#_Toc348028966)

[5.2.2.5 Modifier la vue 46](#_Toc348028967)

[5.2.2.6 Réinitialiser la carte 47](#_Toc348028968)

[5.2.3 Charger une carte 48](#_Toc348028969)

[5.2.4 Sauvegarder une carte 49](#_Toc348028970)

[5.2.5 Soumettre une carte 50](#_Toc348028971)

[5.2.6 Télécharger une carte 51](#_Toc348028972)

[5.2.7 Énumérer les cartes disponibles 52](#_Toc348028973)

[5.2.8 Tester une carte 53](#_Toc348028974)

[6. Vue de déploiement 54](#_Toc348028975)

[7. Taille et performance 54](#_Toc348028976)

Document d'architecture logicielle

# Introduction

Le présent document présente l’architecture du projet Hockedu. À cette fin, le document contient les objectifs et contraintes architecturaux, une vue des cas d’utilisation, une vue logique, une vue des processus, une vue de déploiement ainsi que les caractéristiques de taille et performance.

# Objectifs et contraintes architecturaux

**Dans cette section, il sera question des choix architecturaux du programme accompagnés des raisons pour lesquelles ces choix ont été faits ainsi que les objectifs à réaliser pour respecter ces choix.**

## ****Séparations (façades) entre les différents modules****

Afin de bien définir les limites entre les différents modules, il est impératif de créer des façades pour les communications. Par exemple, le module de réseautique peut être réutilisé facilement dans le code du serveur de jeu ainsi que dans le client lourd. La réutilisation de code en est donc grandement améliorée ainsi que la portabilité du code et cela permettra de faciliter les communications entre les différents langages de programmation.

## ****Structure en arbre pour les éléments de jeu****

Afin de permettre un rendu facile et une manipulation hiérarchique des objets de jeux, la structure en arbre est utilisée. En utilisant cette architecture, il est facile d’effectuer la modification d’objets enfants sur la table de jeu. Par exemple, le déplacement d’un muret entraîne facilement le déplacement des deux points qui le compose.

## Machine à états pour les différentes situations du client et du serveur

Notre programme utilisera également le patron de conception state afin de bien gérer les différents modes de souris ainsi que bien gérer les événements reçus par les clients ou les serveurs. Cela simplifie grandement la gestion des actions des différents éléments de notre architecture et permet une meilleure réutilisation de notre code afin d’ajouter de nouvelles fonctionnalités.

## Utilisation d’usines pour la création d’éléments spécifiques

Afin de permettre de facilement changer le type d’un certain objet à utiliser dans tout le logiciel, il est utile d’utiliser une factory qui s’occupe de créer ce type d’objets dans l’ensemble de notre logiciel. Cela est donc utile pour les objets à placer sur la table. Par exemple, si l’utilisation du même code C++ pour le iPad (discuté plus bas) est choisie, il sera facile de modifier uniquement l’usine d’objets afin d’utiliser des objets différents (pas en 3D par exemple) pour la version du client léger.

## Utilisation de visiteurs pour la modification des nœuds

Pour permettre de séparer les opérations à effectuer sur les nœuds et les nœuds comme tel, il sera question d’utiliser le patron de conception *visitor*. De cette façon, le code de notre logiciel est beaucoup plus portable et réutilisable. Les modifications à apporter sont plus facilement réalisable et il y a moins de duplication de code puisque le même visiteur peut être utilisé pour plusieurs types de nœuds différents.

## Architecture multi-fils

L’architecture multi-fils est une nécessité dans le cas d’une application en réseau afin de permettre de ne pas bloquer l’interface utilisateur lorsque le programme est en attente d’une communication réseau. De cette façon, la réception et l’envoi d’information sont situés dans deux fils différents qui utilisent des files (*queue*) d’actions afin de transmettre les informations au fil d’exécution principal du jeu.

De plus multiples fils d’exécution sont utilisés afin de permettre le chargement des modèles 3D de manière asynchrone en arrière-plan lorsque le client lourd est dans le menu du jeu.

## Les paquets reçus sont « exécutables » sur le thread principal

À la réception d’un paquet, le fil de réception ne peut pas appliquer les informations immédiatement sur les objets de l’arbre de rendu. Pour ce faire, il doit y avoir une structure afin de mettre les actions en « attente » pour qu’ils soient exécutés sur le fil d’exécution principal avant le prochain affichage. De cette façon, les informations relatives aux objets ne sont pas modifiées pendant l’affichage. De plus, on s’assure que toutes les informations relatives à un « tic » en particulier sont effectuées en même temps et ne sont pas séparées.

## Exceptions et système de journal

Puisque le débogage de l’application devient plus difficile avec les multiples fils d’exécution, l’implémentation d’un système de journal avec des indices de temps sera nécessaire afin de comprendre les problèmes sans modifier la synchronisation de l’application. De cette façon, un type d’exceptions spécial pour la réseautique est utilisé dans notre application. De cette façon, il est plus facile de savoir exactement tout ce qui s’est passé puisque toutes ces exceptions sont envoyées directement dans le journal. De plus, une entrée de journal est ajoutée pour chaque événement reçu par le réseau. L’objectif est d’appliquer ce modèle à l’ensemble de l’application et d’ajouter la pile des appels en cas de plantage de l’application.

## Authentification de l’utilisateur

Pour permettre la sécurité et la confidentialité des informations de l’utilisateur, il est nécessaire que ce dernier s’authentifie avec son nom d’utilisateur ainsi que son mot de passe. Cela implique d’avoir une base de données avec lequel le serveur maître doit communiquer afin d’effectuer l’authentification des utilisateurs qui veulent jouer en ligne.

Le point négatif à cette architecture est le couplage fort entre les différents serveurs et la base de données et également le fait de forcer l’utilisateur à entrer un mot de passe.

## Utilisation de *shared\_ptr*

L’utilisation de *shared\_ptr* de la librairie standard en C++ est nécessaire dans certains cas pour des objets qui sont utilisés globalement dans plusieurs parties du programme. Cela nous limite au langage C++ dans le cas du client lourd et du serveur maître puisque l’utilisation de tels pointeurs est nécessaire dans le cas des Sockets et des joueurs contenus dans une partie. En utilisant les *shared\_ptr* dans ces cas précis, cela nous permettait de moins se soucier de problèmes de libération de mémoire puisque dans les deux cas le couplage de ces objets est très élevée. Le prix à payer ici est la transmission de références à un objet à la place de la transmission de pointeur lors d’appels de fonction avec paramètres.

## Objective C++

Afin de maximiser la réutilisation du code, l’objectif est de pouvoir réutiliser un maximum du code C++ du client lourd afin d’effectuer l’adaptation iPad. Cela est réalisable avec du Objective C++ contenu dans des fichiers *.mm* qui sont compilés directement dans Xcode. De plus, cela permettrait de s’assurer d’avoir deux clients différents qui utilisent la même structure pour l’édition et le jeu ainsi que pour la sauvegarde des terrains au format xml.

## Échéancier

L’échéancier est un facteur clé dans le développement de ce logiciel. Il est donc important de bien fixer les priorités pour obtenir un résultat correspondant aux requis. Puisque le temps est limité, la sécurité et la confidentialité des informations utilisateur n’est pas une priorité, mais il sera quand même question d’un système d’authentification avec une base de donnée afin d’en avoir un minimum.

La majorité de notre architecture sera affectée par cette contrainte et ce qui sera optimisé au maximum sera les fonctionnalités.

## Taille et performance

Ce point sera abordé à la section 7 de ce document.

# Vue des cas d’utilisation











# Vue logique

## Description des paquetages

|  |  |
| --- | --- |
| **Présentation** | |
| Description: | Réprésente la couche de présentation. C’est ce avec quoi l’utilisateur interagit. |
| Classes incluses: |  |
| Relations: |  |
| Sous-paquetages: | Interface Lourd, Interface Léger, Interface Web |

|  |  |
| --- | --- |
| **Présentation :: Interface Lourd** | |
| Description: | Représente l’interface utilisateur pour le client lourd développé en C# pour la plateforme PC |
| Classes incluses: | - Chat  - Login  - MenuPrincipal  - Options  - Tournoi  - GameBrowser  - Editeur  - Partie  - Profils AI |
| Relations: | Communique avec le Jeu en passant par la FacadeC#,  Envoi les événements utilisateur à l’éditeur lourd pour traitement. |
| Sous-paquetages: |  |

|  |  |
| --- | --- |
| **Présentation :: Interface Léger** | |
| Description: | Représente l’interface utilisateur pour le client léger développé en Objective-C pour la plateforme iPad |
| Classes incluses: | - Login  - MenuPrincipal  - Editeur  - Options |
| Relations: | Envoi les événements utilisateur à l’éditeur léger pour traitement. |
| Sous-paquetages: |  |

|  |  |
| --- | --- |
| **Présentation :: Interface Web** | |
| Description: | Représente l’interface utilisateur pour la page web du jeu. |
| Classes incluses: | - Login.html  - Register.html  - LeaderBoard.html  - Achievements.html  - Stats.html  - TerrainUtilisateur.html  - TerrainPublic.html |
| Relations: | Commnunique avec le serveur web pour obtenir les informations pertinentes à afficher. |
| Sous-paquetages: |  |

|  |  |
| --- | --- |
| **Modèle** | |
| Description: | Représente la couche de logique des applications. |
| Classes incluses: | - EtatSouris  + ÉvénementClavier  + ÉvénementSouris  + GestionnaireEtat  + GestionnaireEvenement  + SoundFMod  + GestionnaireAchivement |
| Relations: |  |
| Sous-paquetages: | Jeu, Éditeur Lourd, Éditeur Léger |

|  |  |
| --- | --- |
| **Modèle :: Jeu** | |
| Description: | # GestionnaoreHUD  # GestionnaireEtatModeJeu  # GestionnaireAnimations  - EtatSourisJeu |
| Classes incluses: |  |
| Relations: | Communique avec le serveur de jeu pour synchroniser des parties  Communique avec le serveur maitre pour faire l’authentification des utilisateurs et enregistrer les *achievements.* |
| Sous-paquetages: | Facade, Réseau, Partie, Terrain, Modèle3D |

|  |  |
| --- | --- |
| **Modèle :: Jeu :: Facade** | |
| Description: | Représente le point d’entré sur le modèle du jeu par les autres paquetages. |
| Classes incluses: | + FacadeC#  + FacadeModele |
| Relations: |  |
| Sous-paquetages: |  |

|  |  |
| --- | --- |
| **Modèle :: Jeu :: Réseau** | |
| Description: | Représente les éléments s’occupant de la gestion du réseau pour le jeu et les serveurs Jeu et Maitre |
| Classes incluses: | + Paquet  + CommunicateurReseau  + GestionnaireReseau  + Controlleur  - ExceptionReseau  - UsinePaquet  + PacketHandler |
| Relations: | Communique avec le reste du jeu en passant par les façades du modèle. |
| Sous-paquetages: |  |

|  |  |
| --- | --- |
| **Modèle :: Jeu :: Partie** | |
| Description: | S’occupe de la gestion des éléments de jeu concret. |
| Classes incluses: | + Partie  + Tournoi  + JoueurAbstrait  - AIMaillet  - AIStrat  - Zamboni |
| Relations: | Communique avec le reste du jeu en passant par les façades du modèle. |
| Sous-paquetages: |  |

|  |  |
| --- | --- |
| **Modèle :: Jeu :: Terrain** | |
| Description: | Représente la structure des éléments de jeu utilisé dans les éditeurs et durant la partie. |
| Classes incluses: | + NoeudAbstrait  + Terrain  + ZoneEdition  + Visiteur  - UsineNoeuds  - ConfigScene |
| Relations: |  |
| Sous-paquetages: |  |

|  |  |
| --- | --- |
| **Modèle :: Jeu :: Modèle3D** | |
| Description: | Effectue le chargement et la gestion des modèles 3D utilisés pour l’affichage du jeu. |
| Classes incluses: | + GestionnaireModeles  + Modele3D |
| Relations: | Sert à donner une représentation visuelle aux différents éléments du terrain. |
| Sous-paquetages: |  |

|  |  |
| --- | --- |
| **Modèle :: Éditeur Lourd** | |
| Description: | Gère les événements de l’utilisateur pour manipuler le terrain en mode édition sur la plateforme PC. |
| Classes incluses: | - EtatSourisAjout  - EtatSourisSelection  - EtatSourisDeplacement  - EtatSourisEchelle  - EtatSourisRoation  + GestionnaireEtatModeEdition  + Camera  + LumiereAbstraite |
| Relations: | Communique avec le serveur web pour le chargement et enregistrement des terrains de l’utilisateur.  Importe la structure des terrains du paquetage Jeu. |
| Sous-paquetages: |  |

|  |  |
| --- | --- |
| **Modèle :: Éditeur Léger** | |
| Description: | Gère les événements de l’utilisateur pour manipuler le terrain en mode édition sur la plateforme iPad. |
| Classes incluses: | - EtatAjout  - EtatSelection  - EtatDeplacement  - EtatEchelle  - EtatRotation  + GestionnaireEtatModeEdition |
| Relations: | Communique avec le serveur web pour le chargement et enregistrement des terrains de l’utilisateur.  Importe la structure des terrains du paquetage Jeu. |
| Sous-paquetages: |  |

|  |  |
| --- | --- |
| **Serveur** | |
| Description: | Représente la couche des serveurs permettant la communication entre les différents clients. |
| Classes incluses: |  |
| Relations: |  |
| Sous-paquetages: | Maitre, Jeu, Web |

|  |  |
| --- | --- |
| **Serveur :: Maitre** | |
| Description: | Serveur gérant l’authentification des utilisateurs et sert de point de contact pour accédé aux différents serveurs de jeu. |
| Classes incluses: | - Request  - GameServer |
| Relations: | Communique avec la base de données pour accéder aux données sensibles.  Importe la structure des paquets du paquetage Réseau pour cohérence dans les communications. |
| Sous-paquetages: |  |

|  |  |
| --- | --- |
| **Serveur :: Jeu** | |
| Description: | Serveur gérant les communications entre 2 clients durant les parties. |
| Classes incluses: |  |
| Relations: | Importe la structure des paquets du paquetage Réseau pour cohérence dans les communications. |
| Sous-paquetages: |  |

|  |  |
| --- | --- |
| **Serveur :: Web** | |
| Description: | Serveur permettant la gestion des terrains utilisateurs et l’affichage des informations utilisateurs sur l’interface Web. |
| Classes incluses: |  |
| Relations: | Communique avec la base de données pour accéder aux données sensibles. |
| Sous-paquetages: |  |

## Diagramme de paquetages



## Diagrammes de classes

### Couche Présentation

#### Interface Lourd



#### Interface Léger



### Couche Modèle



#### Jeu



##### Facade



##### Réseau



##### Partie



##### Terrain



##### Modèle3D



#### Éditeur Lourd



#### Éditeur Léger



# Vue des processus

## Gerer un Tournoi



### Choisir les joueurs



### Choisir une carte



### Choisir le nombre de joueurs



### Sauvegarder tournoi



### Charger tournoi



## Gérer les cartes



### Créer une nouvelle carte



### Modifier une carte



#### Manipuler des objets



##### Ajouter un objet



##### Supprimer un objet



##### Dupliquer un objet



##### Déplacer un objet



##### Mettre un objet à l’échelle



##### Modifier les propriétés physique d’un objet



##### Sélectionner un objet



#### Annuler une action



#### Refaire une action



#### Manipuler la surface de jeu



#### Modifier la vue



#### Réinitialiser la carte



### Charger une carte



### Sauvegarder une carte



### Soumettre une carte



### Télécharger une carte



### Énumérer les cartes disponibles

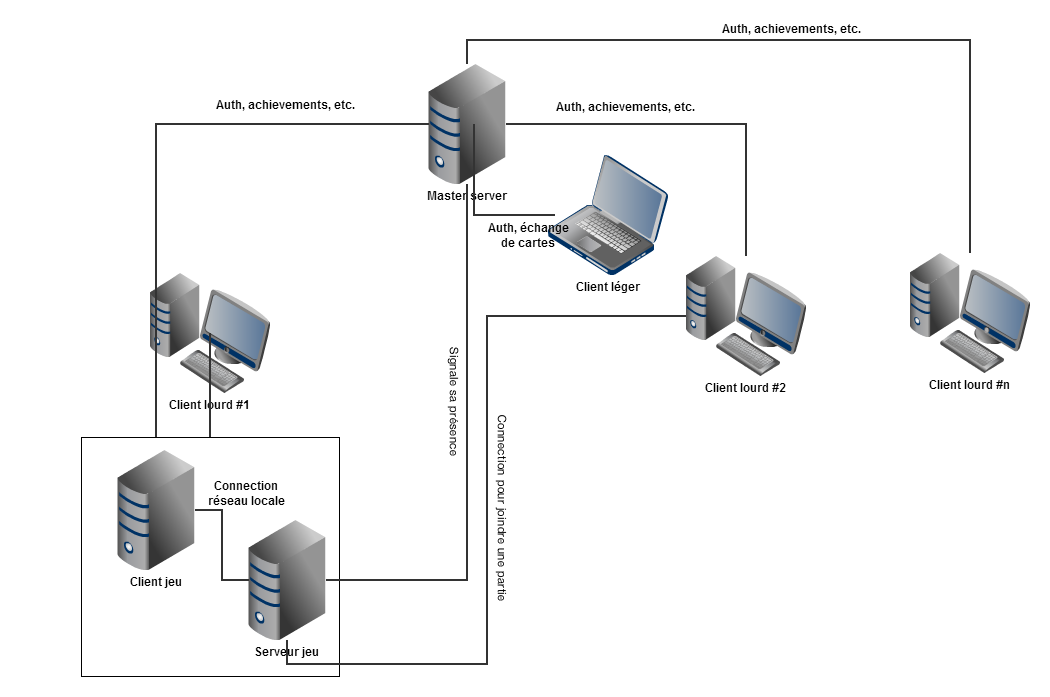


### Tester une carte



# 

# Vue de déploiement



# Taille et performance

* Se basant sur le fait que l’affichage est la tâche qui consomme le plus de temps CPU, la simulation du jeu est effectuée localement sur chacun des clients à des fins de prédiction et sur le serveur pour la synchronisation. Les données du serveur ont précédence sur celles des clients.