RAPPORT JAVA - URBAN JUNGLE

Implémentation d'un jeu multijoueur en réseau

Mathieu CANZERINI - Omar EDDASSER - Florent GERARD

SOMMAIRE

1. Introduction
2. Présentation du jeu
3. Structure du projet
4. Implémentation
5. Difficultés rencontrées
6. Conclusion
7. Annexes

1. Introduction

Dans le cadre de notre cours de Java avancé, nous avons été amenés à réaliser un programme permettant de jouer à un jeu de société sur son ordinateur. Malgré la mise a disposition d'une liste de jeux sur lesquels nous aurions pu travailler, nous avons choisi de réaliser notre propre jeu : **UrbanJungle**. En effet, nous avions déjà une idée assez précise d'un jeu pouvant être entièrement créé et programmé par nos soins et étant suffisamment consistent pour mettre en pratique toutes les notions abordées dans le cadre du cours de java avancé. Nous avons choisi ce jeu, car il était plus motivant pour nous de travailler sur un projet personnel que d'essayer de créer une "copie" d'un jeu déjà existant.

2. Présentation du jeu

UrbanJungle est un jeu de stratégie fonctionnant au tour par tour dont l'objectif pour les joueurs est de prendre le contrôle d'une ville. On peut y jouer seul (en local) face à une intelligence artificielle ou à plusieurs en réseau (jusqu'à quatre joueurs).

Le jeu dispose d'une interface graphique pratique et clair permettant de très rapidement prendre en main le jeu et d'effectuer les différentes actions dans le jeu assez facilement (construire des bâtiments, construire des unités etc.).

Une partie de jeu pouvant être assez longue, il est possible de sauvegarder celle-ci et de la reprendre plus tard (cela aussi bien en local face à l'IA qu'en réseau à plusieurs joueurs)

Compte tenu du fait que le jeu est une création à part entière, son fonctionnement détaillé est donné en annexe.

3. Structure du projet

Contenu de l'archive

L'archive de notre projet contient 5 répertoires et 2 fichiers (le rapport et le script d'importation de la base de données) :

```
UrbanJungle
|__ /Rapport_canzerini_eddasser_gerard.pdf
|__ /urbanJungle.sql
|__ /bin
|__ /build
|__ /library
|__ /ressources
|__ /src
```

/bin : contient les sources compilés (.class)
/build : contient les exécutables (Client.jar , Server.jar ainsi que les librairies décompressées)
/library : contient l'ensemble des librairies utilisées dans notre projet (au format .jar)
/ressources : contient l'ensemble des ressources utiles à la fois au client (images, traductions,...) qu'au serveur (configuration, ...)
/src : contient les fichiers sources de notre projet (.java)

Répartition des packages

Notre projet étant assez conséquent, nous avons donc naturellement subdivisé celui-ci sous forme de package :

- **client** (ce package contient les classes spécifiques au client)
  - **command** (contient les classes représentants les commandes exécutées à partir des informations reçues du serveur)
  - **controller** (contient les contrôleurs des vues)
  - **view** (contient les vues des menus et des parties)
    - **jeu** (contient les vues propres au jeu en lui-même)

- **common** (ce package contient les classes communes au serveur et au client)
  - **ia** (contient les classes constituant l'IA)
  - **partie** (contient les classes du modèle)
    - **batiment** (contient les classes spécifiques aux bâtiments)
    - **plateau** (contient les classes spécifiques au plateau de jeu)
    - **unite** (contient les classes spécifiques aux unités)

- **server** (ce package contient les classes spécifiques au serveur)
    - **command** (contient les classes représentant les commandes exécutées à partir des informations reçues des clients)
    - **view** (contient les vues de l'interface du serveur)

Compilation de l'application

La compilation du client et du serveur s'effectue avec les commandes suivantes :

```
cd src
javac client/Client.java
javac server/Server.java
```

Attention : pour compiler et exécuter l'application, il est nécessaire soit d'avoir les librairies dans son "classpath" ou soit de déplacer dans le répertoire **/src** l'ensemble de librairies (Vous pouvez tout simplement récupérer les dossiers **com**, **junit** et **org** présent dans le répertoire **build** pour les placer directement dans **src** pour la compilation et l'exécution)

Exécution de l'application

Pour pouvoir lancer l'application, il est nécessaire :

- D'initialiser la base de données MySQL : pour cela, utilisez le script **urbanJungle.sql** pour créé la base de données (et tables)
- Correctement configurer le fichier **/ressources/config.xml** : par défaut, le serveur contenant la base de données est en "**localhost**", le port d'accès à mysql est le "**3306**", le nom d'utilisateur est "**root**" (sous WAMP, le mot de passe est vide par défaut, sous MAMP, le mot de passe est "**root**")
- Bien vérifier la présence du répertoire **/ressources**" ainsi que des librairies (jdom, junit, hamcrest et mysql-connector) au même niveau que les sources (ou alors précisées dans le classpath)

Une fois, ces étapes réalisées, on peut lancer l'application :

- soit en utilisant la console :

```
java server/Server
java client/Client
```

- soit en utilisant les JARs présents dans le répertoire "**/build**" (vous pouvez directement double-cliquer dessus) ou alors en utilisant la console :

```
java -jar Server.jar
```

```
java -jar Client.jar
```

Note : il n'est pas nécessaire de lancer le serveur si l'on souhaite jouer en solo.

4. Implémentation

Prologue

La réalisation de ce projet devant se faire en équipe (par groupe de trois), il était nécessaire pour nous d'utiliser un outil nous permettant de travailler tous ensemble de manière efficace (travail en parallèle) sur les différentes fonctionnalités du jeu.

Nous avons choisi d'utiliser un logiciel de gestion de versions à savoir GIT associé à un service web d'hébergement et de gestion de développement de logiciels GitHub (https://github.com/). Notre choix s'est porté sur GIT, car d'une part, c'est un outil que nous maîtrisons tous les trois et d'autre part, il nous permet d'avoir un bon suivi de l'avancement du projet, du travail des autres membres du groupe et de pouvoir récupérer facilement le code des autres développeurs.

Implémentation générale du jeu

Afin de garantir une certaine qualité et cohérence dans notre code, nous avons établi un ensemble de règle de programmation et nous nous sommes pliés à des conventions de nommage et d'indentations communes.

Nous avons également décidé de mettre en place les différents Design Pattern (MVC, Command, Factory, Singleton, ...) vu dans le cadre du cours de génie logiciel afin d'avoir un code de qualité et plus facilement maintenable.

Notre jeu s'articule autour de deux classes principales, le client et le serveur (un schéma complet de la structure est disponible en annexe).

Le client

La classe Client permet de représenter un utilisateur de l'application. Elle possède un attribut de type Partie permettant de représenter une partie jouée et un attribut Joueur représentant l'utilisateur. Elle compose donc le modèle dans l'implémentation de notre MVC.

La partie Vue de notre modèle est composée de multiple JPanel mais plus particulièrement par

deux classes :

- **JeuPanel** : Cette classe (qui hérite de JPanel) permet de gérer grâce à son layout (un CardLayout) le passage d'un écran à un autre dans l'application.
- **EcranJeu** : Cette classe permet de gérer l'affichage d'une partie. Elle dispose d'un JLayeredPane, permettant de superposer différentes JPanel et de gérer l'affichage d'une partie en jouant sur la notion de profondeur d'affichage des différents écrans, en les cachant ou en les affichant.

Chaque vue du programme possède son contrôleur afin de respecter au mieux le modèle MVC.
Afin d'avoir des interfaces agréables et jolies nous avons sous-classé certains composants de base de java afin de les rendre plus jolis. Ainsi, nous avons par exemple défini une classe JCoolBoutton qui hérite de JButton, mais qui redéfinit sa méthode d'affichage (paintComponent()) pour avoir des boutons plus jolis.
Le client possède également une classe ServeurListener lui permettant de dialoguer avec le serveur, celle-ci implémente l'interface Runnable et s'exécute dans un processus léger à part.

Le serveur

La classe Serveur permet de lancer le serveur de notre jeu permettant à plusieurs utilisateurs de jouer à plusieurs en réseau. Le serveur fonctionne avec une base de données SQL permettant de mémoriser les comptes utilisateurs enregistrés. Le design pattern Singleton a été utilisé pour l'accès à la base de données, permettant ainsi de n'avoir qu'une (et une seule) connexion effective à la base. La bibliothèque "mysql-connector" est utilisé pour la gestion de la connexion à la base de données.

Le serveur possède une interface graphique très simple permettant de voir la liste des logins connectés ainsi que la liste des parties crées sur le serveur. La classe Serveur crée une classe ServeurListener qui va se charger d'accepter ou refuser les connexions de clients (ici on ne limite pas le nombre de joueurs sur le serveur). Cette classe ServeurListener crée un thread ClientListener pour chaque demande de connexion qui est chargé du dialogue avec le client.

Le dialogue client/serveur

Comme nous l'avons précisés précédemment, le client et le serveur possèdent tout deux une classe assurant le dialogue entre les deux qui est exécuté dans un thread. Afin de facilité ce dialogue nous avons appliqué les Design Pattern "Command", "Factory" ansi que "Singleton". Ces trois patterns ont été utilisés à la fois pour le client et pour le serveur. Ainsi, dans le cas d'un échange client vers le serveur :

- Le client envoi un ensemble de paramètres (dans notre cas sous la forme d'un tableau d'objet) au serveur, le premier paramètre correspondant à un identifiant de commande et le reste des paramètres étant des arguments pour la commande. Le serveur quant à lui reçoit le

tableau de paramètre, récupère le premier paramètre (identifiant de la commande) et le fournit à sa Factory pour récupérer la bonne commande. Ensuite, il exécute la commande en lui donnant au préalable les autres paramètres reçus.

Nous avons également couplé le pattern "Factory" au pattern "Singleton" afin de ne créer qu'une seule instance de chaque commande.

L'interface de jeu

L'interface permettant de disputer une partie se compose d'une succession de JPanel formant des couches gérées par un JLayeredPane. Une image de fond et un JPanel sur lequel est dessiné la grille représentant la ville où se déroule la partie forment les couches les plus profondes. Viennent ensuite les couches avec le contenu des différents onglets autour du plateau.

- Les onglets Bâtiment et Unite : permettent au joueur de sélectionner un type de bâtiment ou d'unité à construire. Les différents types de bâtiment sont définit dans une énumération (classe TypeBatiment) et les types d'unités dans une seconde énumération (classe TypeUnite), toutes deux implémentent l'interface TypeElementPlateau. On peut noter d'ailleurs l'utilisation assez avancée que l'on a fait des énumérations (avec notamment la création de méthodes liée aux éléments de l'énumération, ou encore l'héritage)

- Les onglets Ville et Joueur : permettent au joueur d'acheter des améliorations d'unités et de bâtiments (on notera que tous les onglets sont de type OngletPanel, héritage de JPanel avec un design retravaillé)

- L'onglet Menu : permet de changer la langue de l'interface (deux langues sont pour l'instant disponible : l'anglais et le français). Des fichiers de traduction au format XML contiennent les différentes traductions de tous les textes de l'interface. Une classe Translator contient un objet Configuration (qui stocke les paramètres de configuration du programme), contenant lui-même une table de hachage avec les traductions de la langue sélectionnée par l'utilisateur. Ces traductions sont récupérées dans les fichier XML à l'aide de la bibliothèque JDOM.

Les couches les plus hautes sont constituées de vue appelées occasionnellement et cachées le reste du temps comme la vue de sauvegarde par exemple qui permet de sauvegarder une partie en la sérialisant dans un fichier ou encore la vue d'attente lorsque ce n'est pas au tour du joueur de jouer.

Pour conclure sur cette partie implémentation, on peut souligner le fait que beaucoup de Design Pattern et d'éléments présenté en cours ont été utilisés dans ce projet : Thread (avec de la synchronisation pour éviter d'avoir des incohérences), Timer (schedule permettant par exemple de créer un petit temps d'attente lors d'une partie solo quand l'IA joue), Enum, Classes Anonymes, Sérialisation, Héritage, Interface, Polymorphisme ...

5. Difficultés rencontrées

La principale difficulté rencontrée lors de ce projet a été le manque de temps pour réaliser notre jeu. En effet, malgré les précautions que l'on a prises en commençant le projet très tôt, il nous a manqué du temps pour pouvoir implémenter la totalité des fonctionnalités que l'on souhaitait (entre les autres projets et les différents partiels).

La coordination entre les membres de l'équipe a été grandement simplifiée grâce à l'utilisation de GIT et la répartition du travail s'est faite naturellement. Chaque membre du groupe a choisi les fonctionnalités qu'il souhaitait développer. GIT permettant d'avoir un bon aperçu du travail effectué par les autres membres, nous n'avons pas rencontré de problèmes malgré cette liberté que nous nous sommes octroyés dans l'organisation et la répartition du travail.

Enfin, concernant le langage Java, étant tous trois issus de l'IUT Charlemagne, nous avons quelques années d'expériences de ce langage qui nous ont permis de ne pas avoir de réel problèmes au niveau de la programmation en elle-même. De plus, pour certains d'entre nous, nous avons eu la chance d'assister à des cours sur la programmation de jeux vidéo avec M. Vincent Thomas, ce qui nous a été particulièrement utile dans la réalisation de notre jeu.

6. Conclusion

Ce projet nous a permis de mettre en application de nombreux concepts vu à la fois en cours de java avancé et de génie logiciel. Nous souhaitions dès le départ avoir une application souple et facilement maintenable, c'est pourquoi, nous nous sommes orientés vers les designs pattern et avons utilisés des normes de codages communes.

Nous avons trouvé dommage de manquer de temps pour finalisé notre application avant de la rendre, mais nous la continuerons sûrement plus tard. Nous pensons améliorer notre IA et ajouter un fond à la ville où se déroule la partie. Nous pensons également ajouter des bâtiments spéciaux tels que des commissariats qui auraient une influence de zone sur les profits des bâtiments appartenant aux joueurs ou des hôpitaux capables de soigner les unités à proximité.

Pour finir, nous pouvons dire qu'il est regrettable, compte tenu du grand nombre de points communs et de la complémentarité des cours de java avancé et de génie logiciel, que ces deux cours n'aient pas fait l'objet d'un projet commun, ce qui aurait permis d'avoir un peu plus de temps pour effectuer ces deux projets.

7. Annexes

1. Dossier de présentation d' Urban Jungle

Notre jeu étant une création à part entière et ne se basant pas sur les règles d'un autre jeu existant, nous joignons ce dossier de présentation du jeu afin de présenter la manière de jouer à UrbanJungle et montrer les différentes fonctionnalités de notre application.

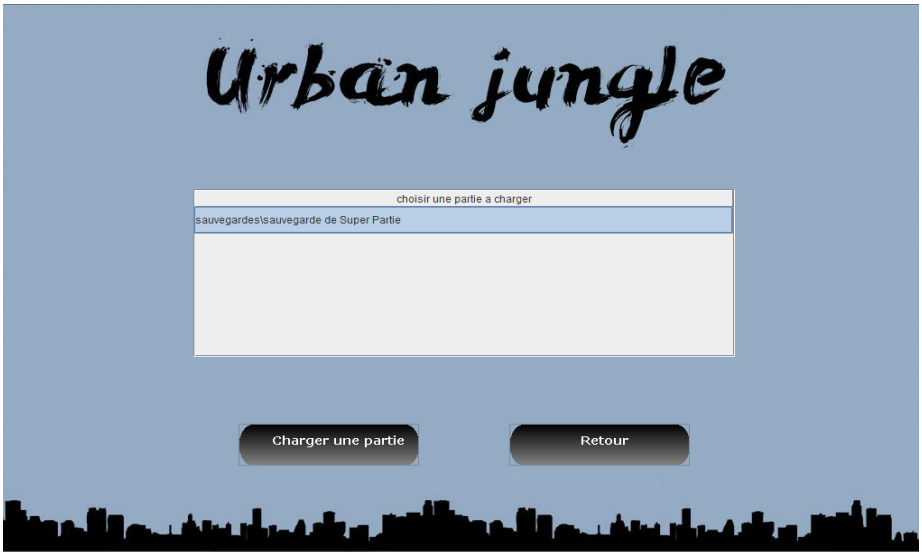
Pour pouvoir profiter pleinement du jeu et de l'aspect multijoueur, il est nécessaire d'avoir un serveur actif avec la partie serveur de l'application de lancé. Dans la présentation du jeu qui va suivre, nous supposons que la base de données urbanJungle est installée sur le serveur et que ce dernier exécute la partie serveur de notre jeu.

Le joueur lambda dispose lui de la partie client de l'application qu'il a juste à exécuter. Les paramètres permettant de se connecter au serveur sont stocké dans le fichier XML de configuration et permettent par défaut de lancer le serveur en local et de s'y connecter.

Présentation de l'interface

Notre jeu possède une interface graphique permettant de rendre l'expérience de jeu agréable.
Le menu principal du jeu permet au joueur de choisir entre jouer en local contre une intelligence artificielle (particulièrement agressive), ou jouer en ligne avec d'autres joueurs grâce au serveur.

Le joueur a également la possibilité de charger une partie préalablement enregistré en local.



Présentation de l'interface multijoueurs

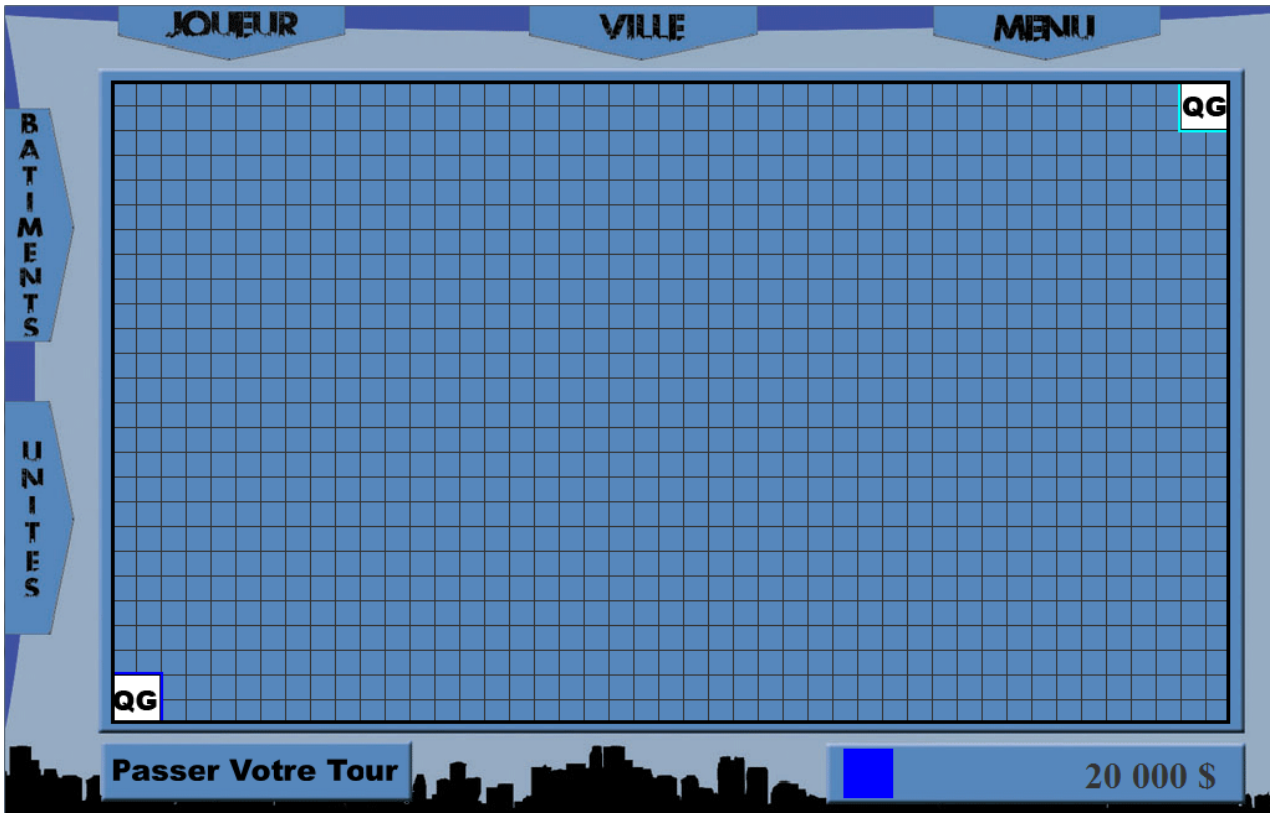
Si le joueur choisis de disputer une partie en ligne il doit d'abord se connecter au serveur en saisissant ses identifiants ou en créant un nouveau compte.

Une fois connecté au serveur, le joueur voit la liste des parties avec leur statut (En attente de joueurs ou commencée),

Il peut alors rejoindre une partie en attente en double cliquant dessus ou créer sa propre partie via une interface où il peut fixer le nom de la partie, un mot de passe éventuel (servant à créer des parties privées) et le nombre de joueurs (de 2 à 4).

Présentation de l'interface de jeu

L'interface permettant de disputer une partie de UrbanJungle est la même pour une partie Locale que pour une partie en ligne (à une différence près, un chat est présent pendant les parties multijoueurs afin de permettre aux joueurs de discuter entre eux).

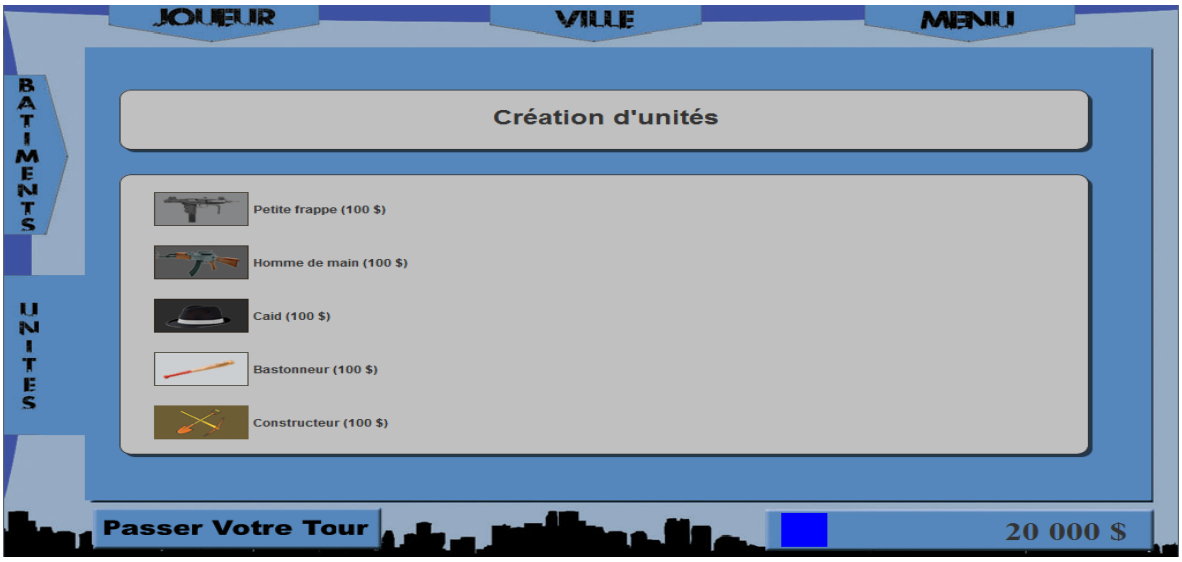


On peut voir au démarrage de la partie (ici partie a deux joueurs) que chaque joueur possède un bâtiment clef : le Quartier Général. Si un joueur perd son quartier général, il a perdu la partie. Afin de remporter la partie, il faut être le dernier joueur en vie.

On peut voir sur cette vue en permanence l'argent que possède le joueur, un bouton "passez votre tour", qui permet de mettre fin à son tour et de laisser jouer ses adversaires, ainsi que différents onglets qui permettent d'effectuer les principales actions du jeu

- L'onglet UNITES

Il permet au joueur de choisir un type d'unité à engager et à placer sur le plateau parmi cinq types d'unités



Les quatre premières unités sont des unités d'attaque possédant chacune une puissance d'attaque, un nombre possible de déplacement par tour et un nombre de point de vie qui lui est propre. Le joueur choisi le type d'unité en fonction de l'argent qu'il possède et de sa stratégie souhaitée.

Le dernier type d'unité (Constructeur) est une unité spéciale qui permet au joueur de construire des bâtiments sur la carte.

Chaque unité demande un salaire par tour propre à son type et à son niveau. Lorsque le joueur a cliqué sur le type d'unité qui l'intéresse, il clique sur la carte à l'endroit où il veut faire apparaître son unité. Les unités ne peuvent être déposées que des cases à proximité de bâtiments du joueur.

- L'onglet BATIMENTS

L'onglet bâtiment permet, de la même manière que l'onglet unité, de positionner sur la carte les bâtiments que le joueur souhaite construire. Les bâtiments permettent de générer de l'argent à chaque tour afin de pouvoir survivre dans urbanJungle. L'argent permet d'acheter de nouveaux bâtiments, des unités ainsi que des améliorations d'unités et de bâtiments. Il est donc très important de gérer l'approvisionnement d'argent à l'aide des bâtiments sous peine de rapidement se trouver submerger par ses adversaires !

Chaque type de bâtiment possède un coût, un montant de points de vie et un revenu par tour qui lui est propre.

Le joueur clique sur le type de bâtiment qui l'intéresse pour ensuite le construire sur la carte. Un bâtiment ne peut être construit que sur une case à proximité d'un constructeur.

- Les onglets JOUEUR et VILLE

Ces deux vues permettent essentiellement d'acheter des améliorations pour les différents types d'unités et de bâtiments présents dans le jeu.

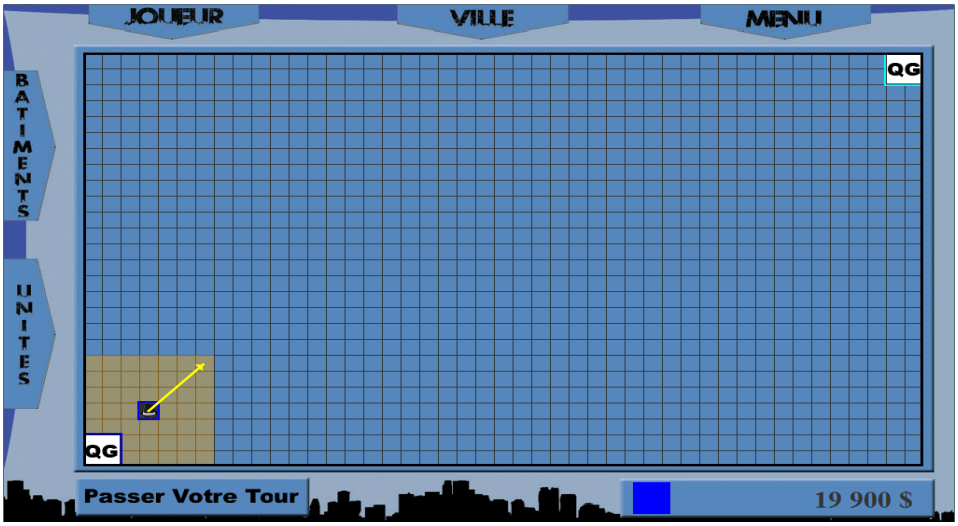
l'amélioration des unités et des bâtiments fait évoluer chacune des caractéristiques de ces éléments à chaque niveaux supplémentaire. Il est très important de bien faire évoluer ses unités afin de ne pas se faire déborder par la puissance des unités adverses.

- L'onglet MENU

Cet onglet est très utile pour l'utilisateur. Il permet de modifier la langue de l'interface (la totalité des interfaces est traduite en anglais).

Cet onglet permet également au joueur de sauvegarder la partie à tout moment et de quitter la partie s'il le souhaite.

- La gestion des déplacements d'unités et de l'attaque



pour déplacer une unité il suffit de cliquer dessus (en maintenant le clic), une zone apparaît alors pour montrer la zone ou l'unité peut être déplacée en fonction du nombre de déplacement qu'il lui reste pour ce tour. Il suffit ensuite de relâcher l'unité sur la case souhaitée pour la déplacer.

Si on déplace l'unité sur une case où une unité ou un bâtiment ennemi se trouve, alors notre unité attaque automatiquement l'élément ciblé. Une attaque de la part d'une unité porte ses points de déplacements à zéro et elle ne peut plus agir pendant ce même tour.

Une fois que le joueur en a terminé avec toutes les actions qu'il souhaite effectuer durant son tour, il clique sur le bouton "passez votre tour" afin de passer la main au joueur suivant.

2. Schéma de la structure de l'application