Rapport n°2
Interface Homme-Machine
L3S5 Informatique

Groupe 12
ARMAND Lyne

HOCHBERGER Dylan

KNAPP Liesse

LEGROS Mélanie

MASSE Robin

INTRODUCTION

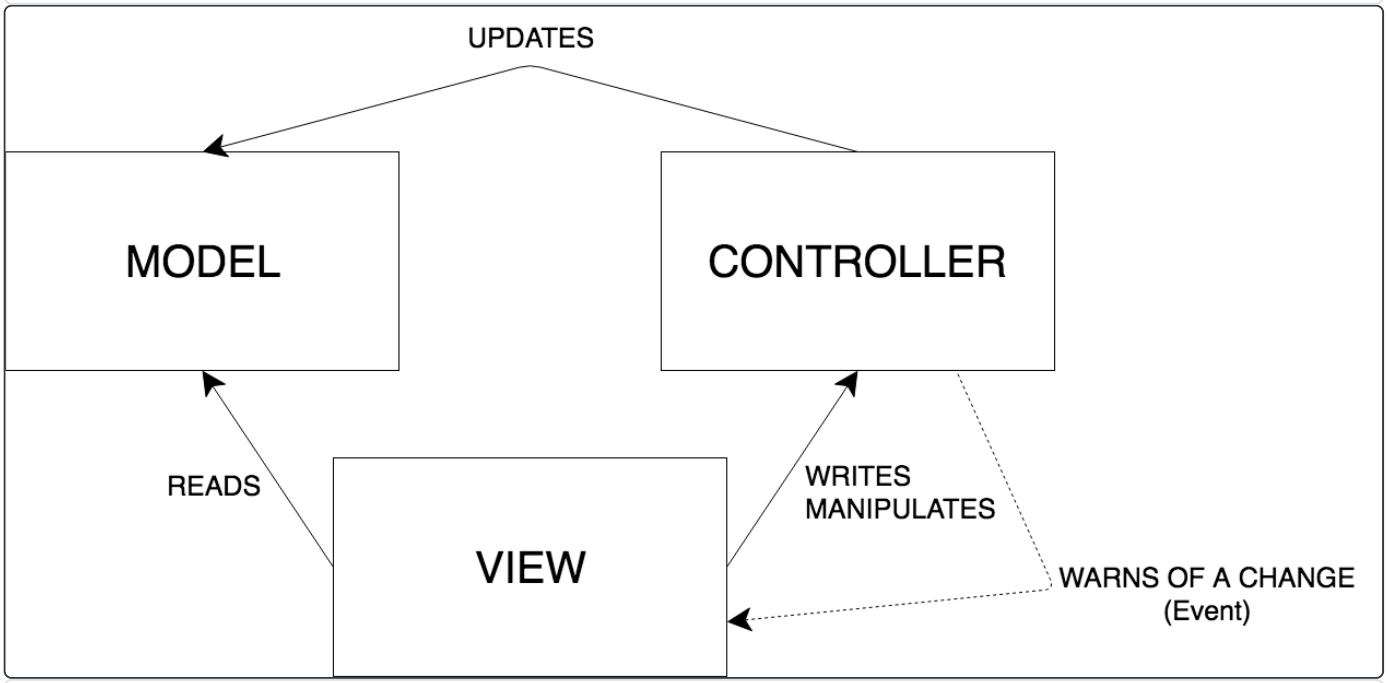
L'objectif de ce projet est de réaliser un jeu de carte nommé Codenames en proposant un noyau de jeu et une interface graphique afin de le mettre en forme.
Pour remplir cet objectif, nous avons choisi comme implémentation une architecture MVC.
Nous allons expliquer notre choix d'implémentation et nos liens entre les interfaces en séparant la version multijoueur locale et la version multijoueur en ligne.

Le projet est donc constitué d'une architecture en trois interfaces.

- Un classique Modèle-Vue-Contrôleur pour la version locale.
- Pour la version en ligne, l'ensemble de la version local servira en quelque sorte de vue pour le serveur, ayant lui même son propre modèle et contrôleur.

SECTION CLIENT

Pour réaliser l'architecture de la section client, nous auront donc nos multiples interfaces qui communiqueront comme présenté ci-dessous :



MODELE

Le Modèle conservera le plateau de jeu, les cartes qui l'occupent ainsi que les options de l'application tel que le volume, les couleurs des équipes, mais également les options des serveurs de jeu dans le cas de la version en ligne.
Les changements envoyés par le Contrôleur seront sauvegardés dans le modèle afin d'être lus par la Vue.

VUE

La Vue servira d'interface à l'utilisateur afin qu'il puisse jouer, regarder et sélectionner les cartes, ainsi que modifier les réglages et les options de son application ou de sa future partie en ligne.
Elle utilisera le contrôleur pour récupérer les informations présentes dans le modèle, ou pour y appliquer les modifications faites par l'utilisateur.

CONTROLEUR

Le Contrôleur servira à réaliser toutes les actions que l'utilisateur peut effectuer grâce à

l'interface.
Voici les fonctions du Contrôleur utilisées pour intéragir avec le jeu :
(La variable 'c' est utiliser à titre d'exemple)

- `Controller c=Controller()` qui permet de générer le contrôleur. (A faire impérativement avant de pouvoir en utiliser d'autres)
- `c.isTeamOnePlaying()` qui retournera un booléen pour savoir si l'équipe 1 joue actuellement ou pas. Si la partie n'a pas encore commencé le booléen indique si l'équipe 1 commence ou non.
- `c.blockColor(int idBlock)` qui peut retourner 0,1,2 ou 3 si la carte correspondant à l'idBlock fourni est respectivement un témoin neutre,un agent de l'équipe 1, un agent de l'équipe 2 ou l'assassin.
- `c.blockWord(int idBlock)` qui retourne une chaîne de caractère, celle contenu sur la carte d'idBlock.
- `c.isBlockSelected(int idBlock)` qui retourne un booléen afin de déterminer si une carte a déjà été jouée, la carte indique son idBlock.
- `c.playBlock(int idBlock)` sert à valider le choix de la carte, qui renvoie 0 si la carte n'appartient à aucune équipe, sinon 1 ou 2 selon l'équipe.
- `c.newMap()` qui regénère un plateau de jeu si on décide de rejouer.

Un ensemble de getters `c.get...()` et de setters `c.set...(...)` afin d'intéragir avec les options de l'application :

- la langue avec `Language` de type `string` .
- le volume principal avec `MasterVolume` et `OldMasterVolume` de type `int` .
- le volume de la musique avec `MusicVolume` et `OldMusicVolume` de type `int` .
- le volume des effets avec `EffectVolume` et `OldEffectVolume` de type `int` .
- la couleur de l'équipe 1 et 2 avec `ColorTeamOne` et `ColorTeamTwo` de type `string` .

autres fonctions intéragissant avec les options de l'application :

- `c.mute()` qui met le jeu en sourdine, `c.unmute()` qui restaure les niveaux des différents volumes.
- `c.isVolumeMuted()` et `defaultSettings()` qui sont explicite.

Un autre ensemble de getters et de setters afin de configurer les options d'une partie en ligne :

- le mode de jeu (Versus ou Coop) avec `VersusMode` de type `bool` .
- les critères de votes avec `VotingStandard` de type `int` (chaque critères étant numéroté de 0 à 4).
- le nombre de joueurs actuel et maximum avec `NbrPlayerMax` et `NbrPlayerCurrent` de type `int` .
- la liberté de choix des rôles avec `FreeRoleChoice` de type `bool` .

- le nom, le mot de passe ainsi que la langue du serveur avec `ServeurName` , `ServeurPassword` et `ServerLanguage` de type `string` .

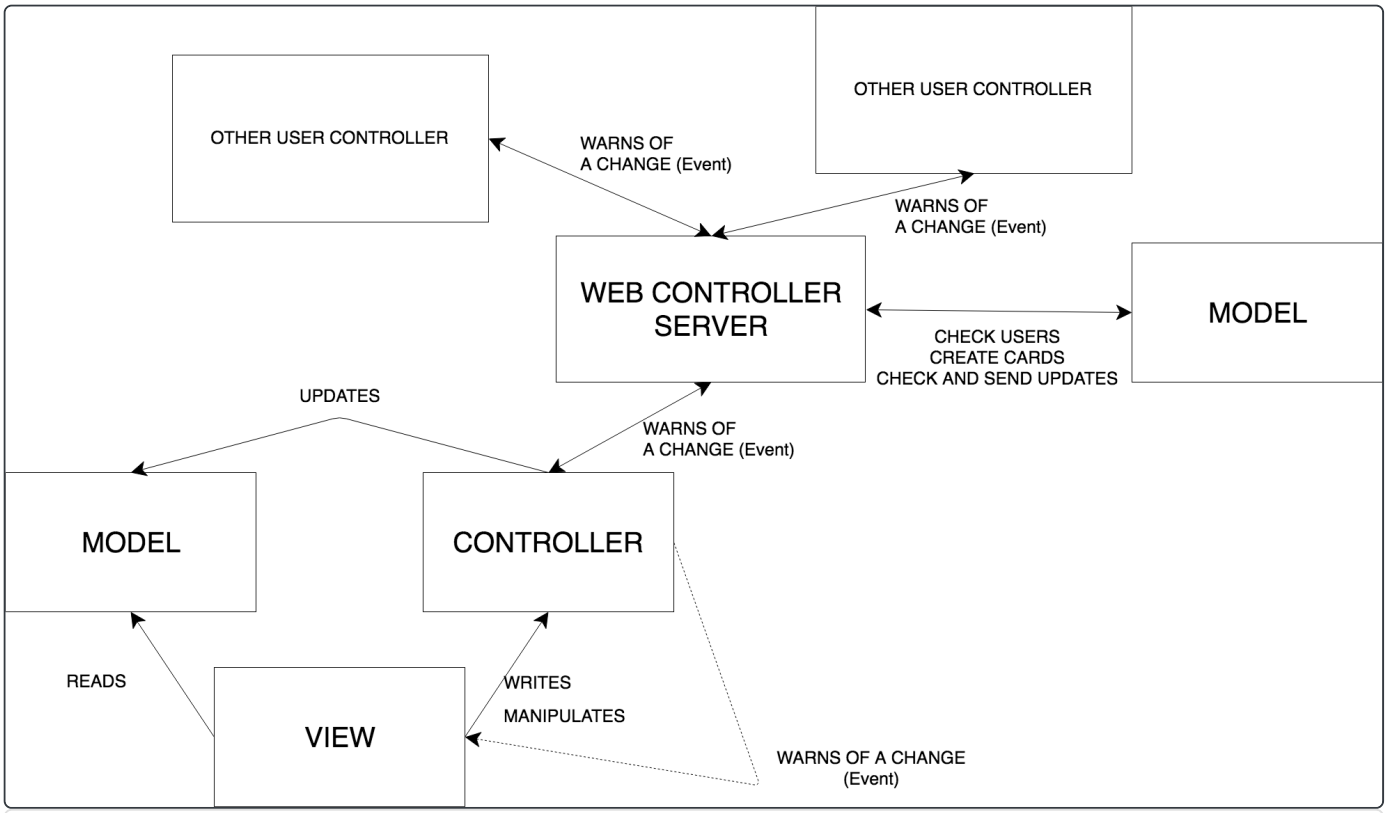
ainsi que `c.defaultServerSettings()` pour initialiser les paramètres de jeu à ceux du jeu originel.

Pour finir, des fonctions permettant de communiquer avec la section serveur par échange de paquets à l'aide du protocole TCP/IP.
Il est important de préciser que tout envoi sera sous la forme de `string` avec comme entête le nom de la fonction d'envoie afin que le serveur détermine ce qui est envoyé et adapte sa lecture.

- `c.sendMap()` pour que le créateur d'un salon envoie son tableau de jeu au serveur.
- `c.playBlockOnline(int idBlock)` qui fait appel à la fonction `playBlock(int idBlock)` puis qui envoie au serveur quelle carte a été joué.
- `c.sendServerSettingsToCreate()` afin de notifier le serveur de la création d'un "serveur" et lui envoyer ses paramètres.
- `c.sendServerSettingsToJoin(ServerSettings chosenServer)` qui envoie au serveur lequel des "serveurs" le joueur souhaite rejoindre.
- `c.sendRematchInvitation()` qui notifie le serveur que le joueur souhaiterai refaire une partie avec ses partenaires.
- `c.sendChat(string message)` pour envoyer un message aux autres joueurs.
- `c.sendPlayerRole()` qui envoie au serveur quel rôle le joueur choisit afin que le serveur puisse lui dire si celui ci est disponible.
- `c.sendNewClue()` qui envoie l'indice ainsi que le nombre de mots s'y rattachant choisi par le joueur maître espion.
- `c.sendGameStarted()` qui préviendra le serveur que la partie à commencé afin qu'il l'efface des "serveurs" disponible.
- `c.receiveSomething()` qui attend la réception des paquets et met à jour le modèle via certaines des fonctions précédentes.

SECTION SERVEUR

Pour réaliser l'architecture de la section serveur, nous auront donc nos multiples interfaces qui communiqueront comme présenté ci-dessous :



MODELE

Le Modèle du serveur en ligne conservera toutes les données importantes du serveur. Il aura ainsi une liste des différentes parties en cours, chacune contenant les liens vers les joueurs connectés.

CONTROLEUR

Le Contrôleur du serveur intéragit avec les Contrôleurs clients à l'aide de deux fonctions :

- `receiveSomething()` attend la réception des paquets. Il vérifera leurs provenances et mettra à jour les informations du serveur.
- `spreadIt(string receivedThing)` vérifiera l'objet reçu par `receiveSomething()` et le redistribuera à l'ensemble des joueurs appartenant au serveur de provenance.

CONCLUSION

En conclusion, nous avons réfléchi à une manière de traiter ce projet au niveau du code qui nous permettrait d'optimiser les liens entre les interfaces. Nous pensons qu'en traitant un maximum de requêtes de la Vue par le Contrôleur cela permettrait d'utiliser au

mieux les différents modules de notre programme et ainsi, nous simplifier leurs créations.