Compte Rendu Projet Tutoré S2

Jeu Qwirkle

Réalisé par :

Ballouhey Mathilde
Billau Elise
Boronat Alexis
Lombard-Massy Theo
Nicolas Dimitri
Rosselle Paul

1ère année DUT Informatique,
Groupe D

informatique amiens

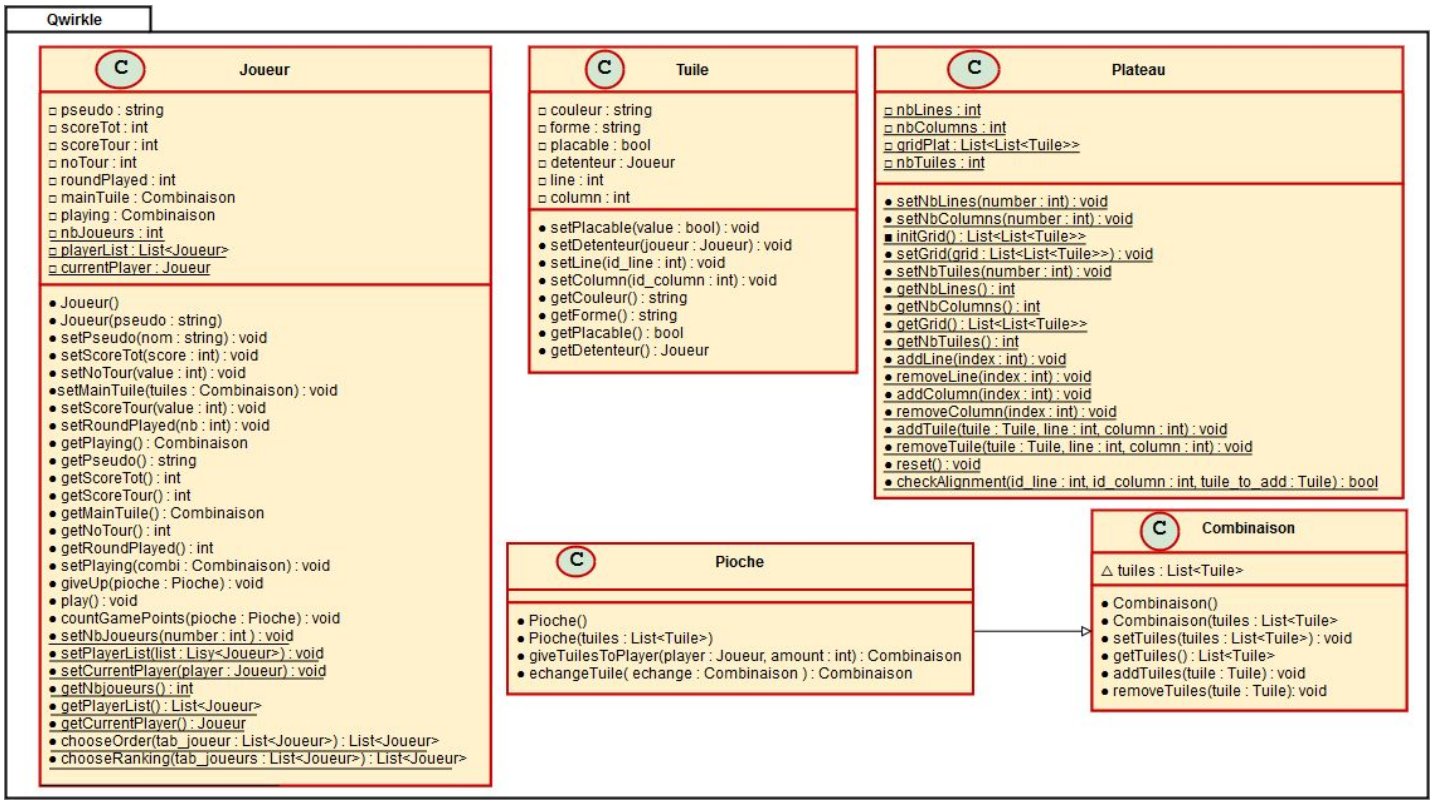
UNIVERSITÉ de Picardie Jules Verne

Le projet qui nous a été donné avait pour but de mettre en pratique les notions vues dans les modules de conception orientée objet et d'introduction aux interfaces homme-machine, ainsi que dans le module de gestion de projet informatique. Il consistait à réaliser un jeu de Qwirkle en y intégrant les fonctionnalités imposées par nos professeurs.
Date du début du projet : le mardi 19 mars 2019
Date de rendu du projet : le dimanche 02 juin 2019

Les fonctionnalités imposées sont :

- Recueil, dans un formulaire, des informations nécessaires à la création d'une partie et à la détermination du 1er joueur;
- Création de la partie en utilisant un plateau de jeu statique de 29*29 cases, afin qu'il y ait une case centrale;
- Le placement du 1er mot;
- Gestion de la pioche;
- Gestion des tours de jeu :
    - Affichage des tuiles du joueur en cours;
    - Placement d'une tuile avec vérification de possibilité de placement (vertical et horizontal) puis comptage des points par rapport à la tuile posée et reconstitution de la main;
    - Échange d'une ou plusieurs tuiles de la main du joueur;
    - Changement de joueur.
- Détermination et affichage du classement en fin de partie.

Après avoir cherché en groupe, comment répondre aux exigence du cahier des charges, nous avons pu créer le schéma UML suivant, il répertorie les fonctions dont nous avons eu besoin dans nos programmes.



1. Description des classes et de leurs fonctions

a. Joueur

La classe Joueur permet de gérer les participants du Qwirkle. On leur a associé : un pseudo, leur score total, leur score durant le tour en cours, le tour où ils sont rendus, le nombre de tours joués, une main de tuiles, une combinaison de tuiles jouées lors de leur tour. De plus, il y a des attributs statiques, et donc communs à l'ensemble des objets instanciés soit : le nombre de joueurs encore en partie (ceux qui n'ont pas abandonné), la liste des joueurs en partie, et le joueur dont c'est actuellement le tour.

Description des méthodes :

- get et set : elles permettent de récupérer ou de modifier les attributs de la classe.
- giveUp : cette méthode est appelée lorsque le joueur courant souhaite abandonner. Elle va décrémenter de 1 le nombre de joueur et retirer celui-ci de la liste de Joueur.
- play : appelée à la fin du tour d'un joueur, elle va incrémenter de 1 le nombre de rounds joués pour le joueur courant, elle incrémente aussi le nombre de tour en fin de tour. C'est cette méthode qui permet de passer au joueur suivant.
- countGamePoint : appelée en fin de tour lors d'un click sur le bouton valider, elle compte le nombre de points marqué par le joueur lors de son tour.
- chooseOrder : cette fonction est statique car elle fait appel à un attribut statique. Elle permet de définir l'ordre de jeu en début de partie. Elle classe les joueurs dans l'ordre par rapport à leur plus grand mot possible de façon décroissante.
- chooseRanking : appelée en fin de partie, elle classe les joueur en fonction de leur nombre de points de façon décroissante.

b. Tuile

La classe tuile, permet de gérer les tuiles de jeu et de les instancier. Elle contient les attributs suivant : la couleur, la forme, si une tuile est plaçable ou non, le détenteur d'une tuile si elle est dans la main d'un joueur, la ligne et la colonne où elle est posée si elle est sur le plateau.

Description des méthodes :

- get et set : permettent de modifier ou de récupérer les attributs de la classe

c. Plateau

Cette classe est entièrement statique car il n'y a qu'un unique plateau, on instancie donc aucun objet Plateau. Les attributs sont : le nombre de lignes, le nombre de colonnes, les tuiles posées sur le plateau sous la forme d'une liste de liste de tuiles, ainsi il est facile d'ajouter une ligne ou une colonne tout en décalant automatiquement l'indice de la position des tuiles (pour une manipulation dynamique du plateau).

Description des méthodes :

- get et set : permettent de modifier ou de récupérer les attributs de la classe;
- initGrid : permet d'initialiser le plateau en début de tour avec les tuiles posées;
- addLine / addColumn (obsolète pour notre gestion de plateau actuelle): permet d'ajouter une ligne ou une colonne à notre plateau dynamique. Elle est appelée lorsqu'une tuile est posée en bordure du plateau
- removeLine / removeColumn (obsolète pour notre gestion de plateau actuelle): permet de retirer une ligne ou une colonne à notre plateau dynamique. Elle est appelée lorsqu'il n'y a aucune tuile sur les deux lignes ou colonnes extérieures.
- reset : permet de remettre le plateau à zéro en vue d'une nouvelle partie.
- checkAlignment : vérifie que plusieurs tuiles soient bien alignées.

d. Combinaison

Cette classe gère les combinaisons de tuiles. Elle est formée d'un attribut stockant une liste de Tuile.

Description des méthodes

- get et set : permettent de récupérer ou de modifier les attributs de la classe
- addTuile : permet d'ajouter une Tuile à la liste
- removeTuile : permet de supprimer une Tuile de la liste

e. Pioche

Cette classe hérite de Combinaison et permet de gérer la pioche lors de la partie. Elle a donc les mêmes attributs et méthodes que celle-ci.

Description des méthodes

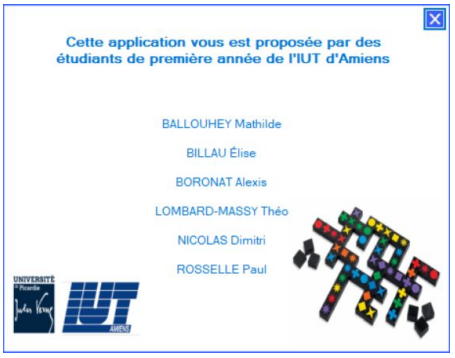
- giveTuilesToPlayer : est appelée quand un joueur doit piocher en fin de tour s'il a posé des tuiles.
- echangeTuile : est appelée lorsque le joueur change une ou plusieurs tuiles de sa main.

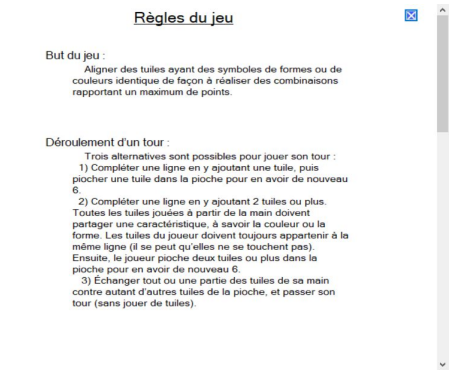
2. Description des interfaces

a. Menu principal

Rôles de l'interface :

L'interface d'accueil étant l'interface sur-laquelle arriveront les utilisateurs lors de l'ouverture de notre jeu, elle se doit d'être claire et simple d'utilisation. Les utilisateurs peuvent donc, depuis cette dernière, lancer une partie, consulter les règles afin de connaître le déroulement de la partie, consulter les crédits ou bien quitter le jeu.





Fonction Implémentée:

➔ Showdialog()

Problème rencontré :

➔ L'agrandissement de l'interface entraînait la modification de l'emplacement des contrôles à l'intérieur de cette dernière. En effet, n'étant pas responsive, les éléments de l'interface vb ne sont positionnés que de manière absolue.

Deux solutions nous faisaient face : rendre l'interface responsive ou bloquer son redimensionnement.

➔ Nous avons choisi la 2ème solution en affectant à la barre de contrôle (FormBorderStyle) la valeur ' None '. Ainsi les utilisateurs n'ont plus la possibilité d'agrandir les interfaces d'accueil (Menu, Règles, Crédits), de sélection du nombre et des noms de joueurs. L'interface du plateau est mise en

plein écran fenêtré. Afin de permettre à l'utilisateur de quitter le jeu, le bouton ' Quitter ' a été ajouté.

b. Choix du nombre de joueurs

Rôles de l'interface :

Nous avions besoin d'une interface permettant aux utilisateurs de sélectionner le nombre de joueurs souhaitant participer à une partie. Une partie ne pouvant se dérouler qu'en présence de 2 à 4 joueurs. Cette interface doit donc leur offrir la possibilité de sélectionner le nombre de personnes voulant jouer et de valider leur choix. Elle doit aussi leur permettre un retour facile au menu du jeu.

Fonctions Implémentées:

- ➔ Joueur.setNbJoueurs()
- ➔ Showdialog()

c. Choix du nom des joueurs

Rôles de l'interface :

La saisie des noms des joueurs dépend du nombre de joueurs sélectionné lors de l'étape précédente. La validation n'est possible uniquement qu'après avoir rempli tous les champs, sinon une fenêtre pop-up s'ouvre et la partie ne peut pas commencer.

Fonctions Implémentées:

➔ Show()

➔ Joueur.setPseudo(pseudo)

Problèmes rencontrés :

- ➔ Ajuster l'interface graphique en fonction du nombre de joueurs sélectionnés au préalable dans l'interface précédente.
- ➔ Solution : On a affiché dynamiquement les éléments qui dépendent du nombre de joueurs afin de pallier ce problème. Ceci pour éviter un affichage statique qui rend visible certains éléments déjà positionnés.
- ➔ Si un au moins l'un des champs réservés pour la sélection des noms était vide, le/les joueur(s) se retrouvait(ent) sans pseudo.
- ➔ Solution : On a ajouté une vérification qui regarde si tous les champs sont complétés. Si c'est le cas, le jeu se charge, sinon, un message d'erreur apparaît demandant aux utilisateurs de définir tous les pseudos.

d. Définition ordre de jeu

Rôles :

Le choix du nombre de joueurs et la sélection de leur nom a permis la définition d'une liste, dont la taille dépend du nombre de joueurs sélectionnés, contenant le nom de ces joueurs. On parcourt ensuite cette liste pour leur donner à chacun 6 tuiles.

Une fois les tuiles distribuées on calcule quel joueur peut placer le mot le plus long au premier tour. Ce joueur devient le "premier joueur ". Puis l'ordre de jeu est défini en classant de façon décroissante, la taille maximale du mot que chaque joueur peut jouer à son premier tour.

Fonctions Implémentées:

➔ Joueur.setMainTuile()

➔ Joueur.ChooseOrder(Liste de Joueur non triée)

➔ PlateauJeu.createPlat()

➔ Joueur.setCurrentPlayer(Liste de Joueur triée)

➔ Show()

Problème rencontré :

- ➔ En cas d'égalité sur la taille des plus long mots des joueurs, il était impossible de choisir le premier joueur.
- ➔ Solution : En cas d'égalité, la position des joueurs possédant un mot de taille semblable est déterminée de façon aléatoire.

e. Transition

Rôles de l'interface :

Une fois l'ordre de jeu défini la partie commence avec le premier joueur. Le jeu se déroulant sur un seul ordinateur, le joueur doit confirmer son identité avant de pouvoir accéder au plateau et à sa main. Cela évite qu'un autre joueur puisse voir sa main, ou puisse jouer à sa place.

Fonctions Implémentées:

- ➔ Joueur.getCurrentPlayer()
- ➔ Joueur.getPseudo()

f. Plateau de jeu

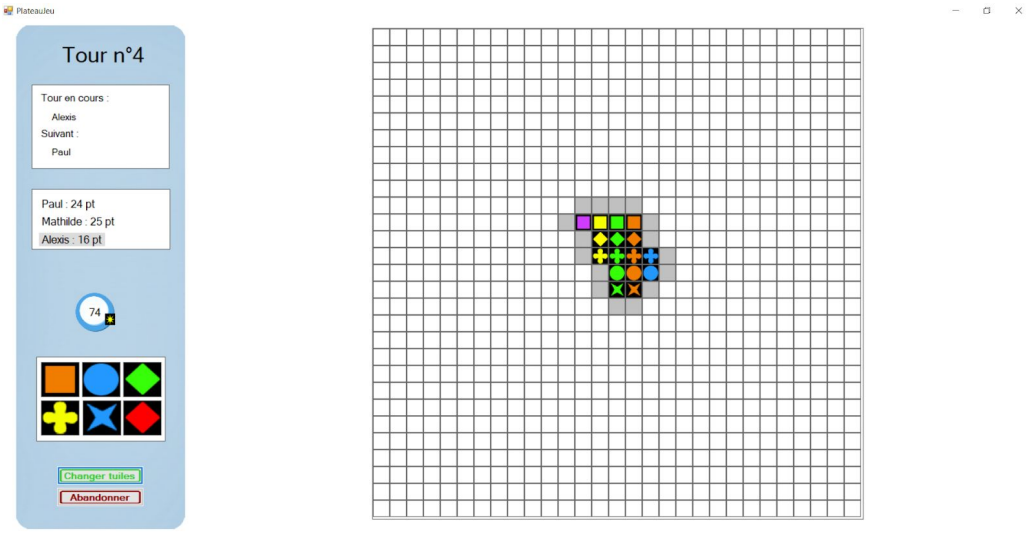
Rôles de l'interface :

Le plateau de jeu est divisé en deux parties : une première partie située sur la gauche permet aux joueurs d'accéder aux informations de la partie, ainsi il peut y voir, le nombre de tours joués, l'indication du nom du joueur qui joue et celui du joueur suivant, le nombre de points de chacun et le nombre de tuiles restant dans la pioche. Sur cette partie, se trouvent aussi la main du joueur affichant les 6 tuiles en sa possession qui peuvent être placées sur le plateau situé sur la droite de l'interface, un bouton "Valider/Passer/Changement de tuiles/Fin de partie" qui change en fonction du contexte (pioche vide, tuile placée, nombre de tuiles dans la main) et un bouton "abandonner" pour permettre à un joueur de se retirer de la partie sans affecter son déroulement.

La partie se termine lorsque la main d'un joueur est vide et sans possibilité de la remplir, ou lorsqu'il ne reste qu'un joueur (abandon de tous les autres), ou si l'ensemble des joueurs passent leur tour successivement sans poser leurs tuiles et sans pouvoir les échanger.

Fonctions Implémentées:

Toutes les fonctions;



Problèmes rencontrés :

- ➔ Lorsque nous déposions une image dans une PictureBox pleine, l'image déjà présente dans cette dernière se trouvait être supprimée et remplacée par la nouvelle image qui venait d'être déposée.
- ➔ Solution : Désactiver l'autorisation (Allowdrop) de dépôt d'image dans la PictureBox lorsque cette dernière contient déjà une image.
- ➔ Lors de la pose ou du retrait d'une tuile, une actualisation des PictureBox situées autour est réalisée afin de déterminer les zones à AllowDrop. Si une tuile est posée en bordure de plateau, cette vérification va chercher une PictureBox qui n'existe pas et donc générer une erreur.
- ➔ Solution : Vérifier que la PictureBox existe avant de changer sa propriété.
- ➔ Il ne restait qu'à déterminer quand nous aurions à effectuer la vérification de la combinaison et comment nous allions la faire.
- ➔ Solution : C'est pourquoi nous l'avons effectuée lors de l'événement DragEnter (quand on entre un objet glissé sur un élément) sur chaque tuile adjacente à une autre. Ainsi, on empêche ou non automatiquement un dépôt de tuiles illicite. De plus, la vérification de la combinaison s'effectue en analysant les tuiles alignées à celle déposée pour éliminer à la fois les doublons et les incohérences de formes et/ou de couleurs.
- ➔ Enfin, le comptage des points posait le même problème, à ceci près qu'il faille analyser et reconstituer chaque combinaison qu'engendreraient les tuiles posées par le joueur lors de son tour. Sans oublier le gain de 6 points dans le cas d'un Qwirkle...
- ➔ Solution : On a donc analysé pour chaque tuile posée par le joueur ayant au moins une tuile adjacente (différente de celles posées par le joueur) les différentes combinaisons formées. Nous n'avions pas pu récupérer l'ensemble de combinaisons formées lors de la vérification car il ne

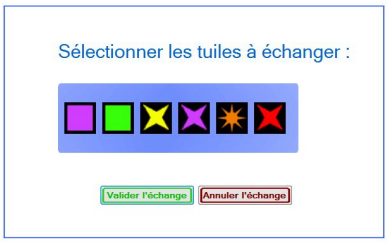
contenait que des morceaux des combinaisons courantes et non des combinaisons entières. On a pu alors incrémenter le compteur. De plus, nous avons rajouté au joueur 6 points s'il a complété une tuile ou une colonne de 6 tuiles ou si lui même a formé une combinaison de 6 tuiles.

- ➔ Récupérer et établir le lien entre la sélection utilisateur (d'une tuile à retirer ou à ajouter) et l'élément instancié réellement.
- ➔ Solution : Pour garantir une bonne récupération des sélections utilisateur, il a fallu nommer correctement et dynamiquement nos tuiles afin d'en extraire plus efficacement les indices de colonnes et lignes de la tuile que l'utilisateur souhaite manipuler. On les a alors nommées comme présent : "pic_8_9" (avec le premier numéro désignant l'indice de ligne et le second l'indice de colonne). Il a fallu effectuer la même démarche pour le nommage des tuiles de la main du joueur.

g. Echange tuiles

Rôles de l'interface :

Permet au joueur de sélectionner les tuiles qu'il souhaite échanger. Les tuiles sélectionnées s'agrandissent puis "Valider l'échange" afin d'obtenir les nouvelles tuiles de la pioche de façon aléatoire. Enfin l'interface affiche le nouveau jeu.



Fonctions Implémentées :

- ➔ getPioche()
- ➔ Joueur.getMainTuile()
- ➔ Joueur.getCurrentPlayer()
- ➔ Tuile.getTuiles()
- ➔ Pioche.echangeTuiles(tuiles sélectionné)
- ➔ Tuile.getCouleur()
- ➔ Tuile.getForme()
- ➔ Combinaison.addTuile(tuile)
- ➔ Combinaison.removeTuile(tuile)

Problèmes rencontrés :

- ➔ Joueur voulant échanger un nombre de tuiles supérieur au nombre de tuiles dans la pioche.
- ➔ Solution : Apparition d'une fenêtre popup interrompant l'échange et passant le tour du joueur.

➔ Main du joueur vide, et pioche vide, erreur car on essaye de remplir la main alors que la pioche est vide.
➔ Solution : Fin de la partie, ouverture du formulaire “Classement”

h. Classement

Rôles de l’interface :

Le résultat de la partie s’affiche par ordre de score décroissant, si il y a des ex-aequo alors ils seront classés au même rang. Afin de distinguer le/les vainqueur(s), la couleurs de son/leur nom(s) est/sont différente.
A partir de là, il est possible de relancer une partie. Il est également possible de retourner à la page d’accueil.

Fonction implémentées :

- ➔ Show()
- ➔ Joueur.setCurrentPlayer()
- ➔ Joueur.setNbJoueurs()
- ➔ Joueur.setPlayersList()
- ➔ Joueur.chooseRanking()
- ➔ Joueur.getNbJoueurs()
- ➔ .getScoreTot()
- ➔ .getPseudo()

Problème rencontré :

- ➔ Un joueur ex-aequo était classé 1er.
- ➔ Solution : récupérer le rang du joueur précédent dans la liste du classement et l'attribuer au joueur concerné s'il y a un ex-aequo.
- ➔ La partie relancée ne démarre pas car les anciens scores et noms sont toujours en mémoire.
- ➔ Solution : remettre tous les paramètres de jeu à 0 avant de relancer une partie.

Remerciements

Nous remercions Madame Sandrine LELIEVRE et Monsieur Samy DELAHAYE , pour l'aide qu'ils nous ont apporté durant leurs heures TPs afin de nous permettre de réaliser au mieux ce projet.