Programmation Orientée Objets et Interface Graphique

Rapport de Projet

Licence 2

Laure Hamme & Elisa Ranjalahy
06/01/2023

Sommaire

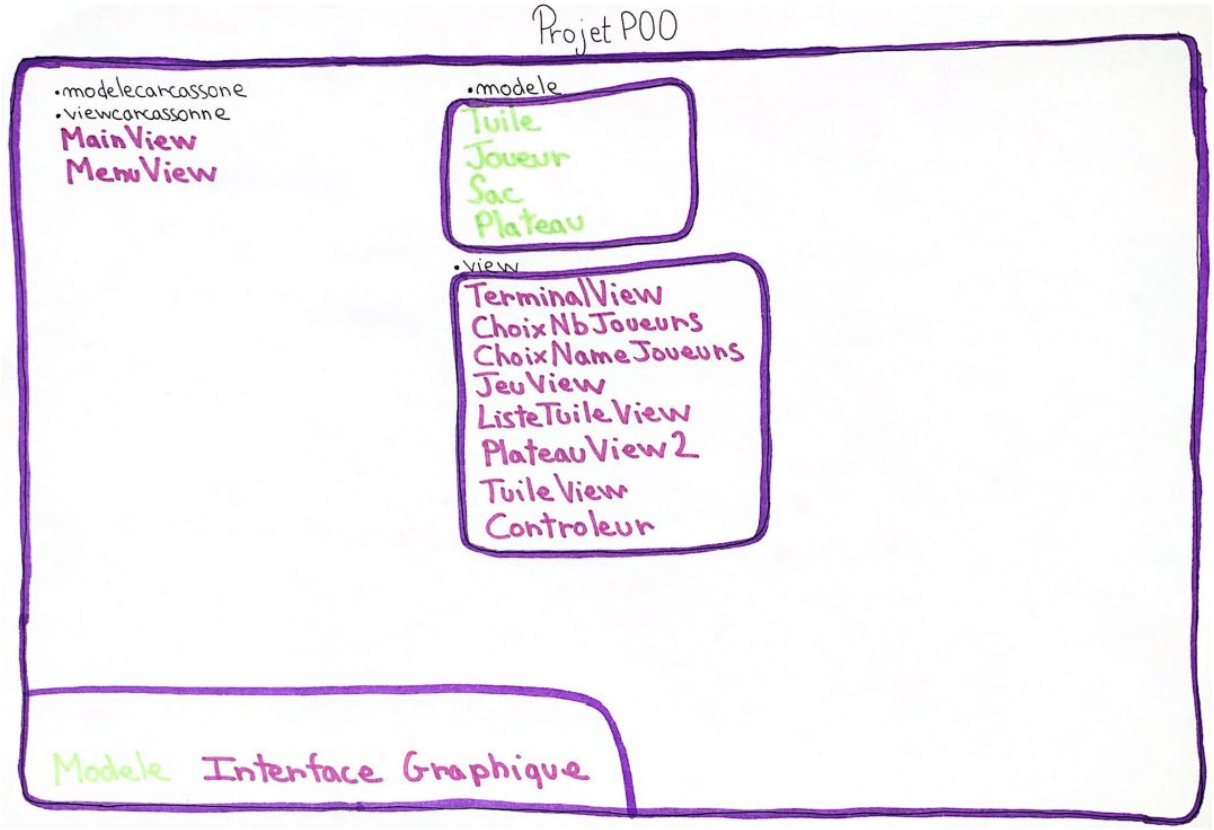


Schéma de l'organisation des classes.

Dans le cadre de nos études universitaires en mathématiques et informatique, nous avons été amenés durant notre seconde année de licence à travailler sur un projet en programmation orientée objets. Notre objectif a donc été d'implémenter deux jeux différents dont la structure était en réalité quasiment la même.

Le premier jeu que nous allons vous présenter est un Domino, un jeu très simple que vous connaissez certainement. Celui-ci ne nécessite pas d'énormément d'éléments de jeu mais seulement d'un plateau ou d'une surface plane de jeu, d'un sac de pièces de domino et tout simplement de joueurs, ou plus précisément de plus d'un joueur car jouer tout seul n'est peut-être pas la chose la plus amusante que l'on puisse faire (bien que cela soit tout à fait faisable).
Vous pouvez remarquer que nous présentons ici les choses en appuyant le fait que ce jeu est « simple » et minimaliste sans voir que ce jeu peut en réalité présenter quelques difficultés quand on décide de le rendre jouable sur un ordinateur. Par notre code, nous avons par ailleurs rendu possible le fait de jouer seul mais cette fois contre une intelligence artificielle (et donc de trouver plus d'amusement que si on s'affrontait soi-même). Mais en quoi consiste ce jeu de Domino ? Laissez-nous vous présenter le sujet qui nous avait été imposé.

DOMINO

« Plusieurs joueurs sont assis autour d'une table, des tuiles sont à leur disposition dans un sac opaque mis en commun. Les tuiles sont des carrés où chaque côté porte trois chiffres.
Au départ, une tuile (prise au hasard) est posée face visible. Chaque joueur à son tour va en piocher une dans le sac, et la déposer sur la table, à côté des autres (avec l'orientation de son choix), pourvu qu'il trouve une correspondance bord à bord avec celles qui y sont déjà. S'il ne le peut pas il la défausse (sans la remettre dans le sac) et passe son tour. Lorsqu'un joueur pose une nouvelle tuile, il marque alors un certain nombre de points : le total des chiffres en contact avec ceux des tuiles voisines. »

Pour commencer, intéressons-nous au modèle (nous nous pencherons sur l'interface graphique par la suite).
Celui-ci se constitue donc de quatre classes :

- Plateau
- Sac
- Tuile
- Joueur

Le Modèle – Les Classes

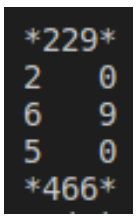
Joueur

La classe Joueur représente tout simplement le joueur qui participe à une partie de Domino. Nous avons décidé de construire une instance de cette classe en lui donnant quatre champs permettant de donner vie à ce joueur virtuel : un entier *score* propre au joueur construit qui est initialisé à 0, une chaine de caractère *nom* prenant tout simplement le nom choisi par le joueur lors d'une création de partie, un tableau de Tuile *mesTuiles* réunissant toutes les tuiles que possède le joueur et enfin un booléen *ia* permettant d'adapter par la suite le code selon si le jouer est un humain ou une intelligence artificielle.

Concernant ce tableau de Tuile, nous avons souhaité, pour rester au plus proche du jeu Domino tel qu'on le connait, permettre au joueur de conserver chacune de ses tuiles piochées au cours de la partie. Il pourra alors établir des stratégies de jeu tout en se rappelant qu'il doit être le premier à finir son « stock » lorsque le sac sera vide pour être le grand vainqueur. Pour rappeler, une partie de Domino termine lorsque le sac est vide et qu'un joueur n'a plus de tuiles en sa possession.

Tuile

Cette classe permet d'instancier la pièce la plus importante du jeu, celle qui permet au jeu d'exister : une tuile, comme son nom l'indique. L'une des difficultés que nous avons eu à rencontrer était la forme de cette pièce. Alors que d'ordinaire, celle-ci est un rectangle divisé en deux avec sur chaque côté un chiffre (ou plutôt un certain nombre de petit rond correspondant à ce chiffre), il nous a été imposé d'implémenter une tuile de quatre faces portant chacune trois entiers.
Nous avons partagé nos points de vue concernant cet objet et avons fini par choisir de le définir comme étant un double tableau d'entiers. Pour parcourir un tel tableau nous savons que l'on peut utiliser deux boucles imbriquées ; la première parcourt donc les faces tandis que la seconde parcours les entiers de chacune d'elles.



Sac

Un sac contient l'ensemble des tuiles, c'est donc tout naturellement que nous l'avons construit comme étant un tableau de tuiles. Celui-ci est au départ un tableau de tuiles vide que nous définissons comme étant des tuiles n'ayant que des zéros, le sac n'est rempli que lorsque chaque tuile est remplie de suites d'entiers non nulles.
Pour cela, nous faisons appel à la méthode remplirSac(), une méthode complexe dont nous sommes fière.

Pour vous la présenter brièvement, cette fonction sépare le contenu du sac en deux, choisit aléatoirement une suite de trois entiers x, y et z entre 0 et 9 (compris) qu'elle ajoute à la face d'une tuile de chacune des deux parties du sac pour garantir une correspondance entre faces. Par ailleurs, le choix de celles-ci lors de cette construction est quant à lui un peu plus travaillé. Le parcours de chaque partie du sac est fait de quatre façon différentes pour éviter l'apparition de tuiles identiques.

Plateau

Enfin, le plateau est le « terrain de jeu », cette fameuse surface plane mentionnée précédemment, nécessaire pour jouer comme il se doit au Domino. Celui-ci est instancié avec pour champs : un double tableau de tuiles formant donc un quadrillage dans lequel se trouvera les tuiles jouées ; un nombre maximum prédéfini de joueurs pouvant jouer lors d'une partie, une liste de ces joueurs représentée par un tableau d'objet Joueur et pour finir un sac.

Nous avons à présent toutes les classes nécessaires au bon déroulement du jeu avec ayant chacune ses propres méthodes et son rôle à jouer dans le code ce qui permet de satisfaire toutes les situations potentielles que l'on pourrait rencontrer.
Cela permet d'introduire notre travail sur l'interface graphiques.

Interface graphique

L'interface graphique du Domino est formée de dix classes (*voir schéma page o*). Nous avons fait le choix de faire un système de fenêtres ou chacune d'elles correspondent à une vue d'un élément spécifique comme le plateau ou la liste

des tuiles d'un joueur. Nous avons d'abord essayé d'imaginer une vue avec une unique fenêtre mais il nous paraissait plus simple de faire ce système de fenêtre qui nous a permis d'avancer dans notre interface graphique, en effet cette étape du projet que nous appréhendions déjà au départ nous avait bloqué pendant un temps.

Nous avons donc deux types d'affichage : d'un côté *TerminalView* qui s'occupe comme son nom l'indique de la « vue sur terminal » et dont le déroulé se fait à l'aide d'un scanner (celui-ci pose des questions au joueur et traite ces réponses en conséquence pour faire avancer le jeu) et de l'autre côté toutes les autres classes pour le second type d'affichage.

Dans cette seconde partie, deux classes sont communes aux deux jeux : la classe *MainView* qui contient le *main* permettant de lancer le jeu et la classe *MenuView* qui contient la vue du menu permettant de choisir quel jeu lancer entre Domino et Carcassonne.
Les autres classes sont propres au jeu du Domino. Quand nous appuyons sur le bouton *Domino* du menu cela nous renvoie vers la classe *ChoixNbJoueurs.*

ChoixNbJoueurs

Cette classe, comme la suivante porte le nom de son rôle. Celle-ci sert donc à choisir le nombre de joueurs. Une fois que le bouton du nombre de joueurs a été choisi, il nous renvoie après clique vers la classe *ChoixNameJoueurs* en lui donnant en argument ce nombre de joueurs.

ChoixNameJoueurs

Cette dernière a pour rôle de récupérer le nom des joueurs afin de créer des instances de Joueur (dont le constructeur prend justement une chaine de caractère comme argument) mais aussi de déterminer si le joueur est un humain ou une intelligence artificielle. Une fois ces paramètres déterminés, un bouton nous renvoie vers la classe *JeuView* prenant comme un argument un tableau de joueurs.

JeuView

JeuView sert à initialiser le sac et le plateau (du modèle) pour ensuite ouvrir deux fenêtres : une pour la vue du plateau (PlateauView2) et une pour la vue du joueur *(ListeTuileView*).

ListeTuileView

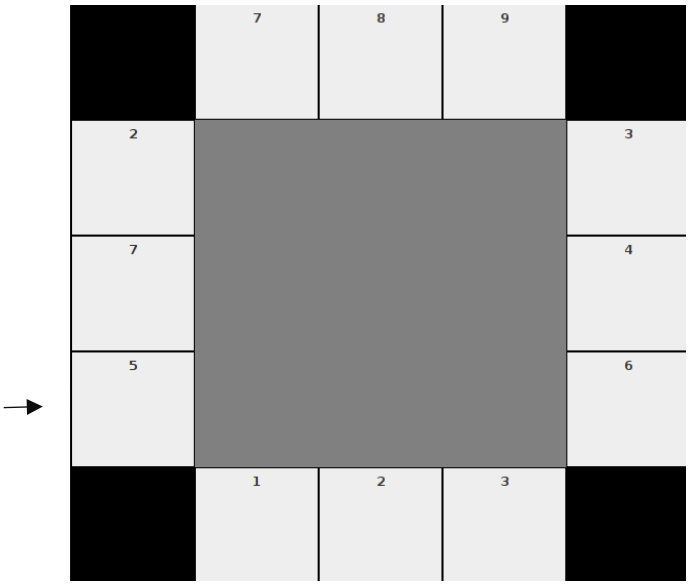
Si le joueur n'est pas une I.A., *ListeTuileView* comporte d'abord des boutons pour piocher, quitter le jeu ou passer au joueur suivant. Nous avons ensuite toutes les tuiles du joueur affichées avec pour chacune un bouton « tourner » et un bouton qui permet de la sélectionner pour la placer sur le plateau.
Ce dernier permet de créer une nouvelle PlateauView2 avec un contrôleur différent. Si le joueur est une I.A. tout cela se fait automatiquement.

PlateauView2 :

Cette classe contient un *grid layout* dont chaque case correspond à une *TuileView* si la case est remplie ou à un *panel* vide sinon. Dès que l'on clique sur un panel vide, si la dernière tuile sélectionnée dans *ListeTuileView* que nous connaissons grâce au contrôleur peut être placée, cela va générer un nouveau PlateauView2 (avec comme argument le nouveau plateau contenant cette nouvelle tuile) ainsi que la *ListeTuileView* du joueur n'ayant plus cette tuile étant donné qu'il vient de la jouer.

TuileView :

Nous avons créé la vue de notre tuile en nous inspirant du modèle du sujet. Elle prend en argument une Tuile. Cette dernière contient un *grid layout* dont chaque case correspond soit à un entier contenu dans une des faces de la tuile, soit à un carré coloré. Cela permet d'obtenir le rendu suivant.



Int [] tuile = {{1,2,3}, {3,4,6}, {7,8,9}, {2,7,5}}

Vue d'une tuile.

Contrôleur

Le contrôleur permet de transférer certaines informations entre les classes *PlateauView2* et *ListeTuileView* notamment l'indice de la tuile qui est jouée, le joueur ou encore l'indice de ce joueur.

CARCASSONNE

Modèle du Carcassonne

Le code du Carcassonne est très similaire au code du Domino. Par manque de temps nous n'avons pas fait d'héritage entre le code des deux jeux malgré l'indication contraire dans le sujet. Le modèle est donc constitué du même nombre de classe avec des noms qui se ressemblent. Il y a tout de même quelques subtilités notamment dans la classe *TuileCarcassonne*.

Nous avons gardé les quatre faces mais cette fois chaque face contient une chaîne de caractère qui représente : une ville, une rivière, une route, une abbaye ou une campagne. Pour faciliter la comparaison de ces chaînes de caractères nous avons donné à chacune une longueur de huit complété par des espaces si besoin («ville» devient « ville »).

Ces nouvelles tuiles ont donc nécessité la modification de plusieurs méthodes comme estVide(..) ou encore construireFaceTuile(...).
La classe TuileCarcassonne a également deux attributs en plus : un booléen *pion* qui est égal à « false » quand la tuile ne contient pas de tuile et à « true » quand elle en contient une et un entier *numJoueur* qui représente l'indice du joueur qui possède la tuile dans le tableau de joueurs de la classe Plateau. Ces derniers seront utiles pour la vue du Carcassonne par la suite.

Interface graphique

L'interface graphique du Carcassonne a pour modèle une base de code issu de la vue du domino avec des modifications.
Nous avons décidé d'associer chaque chaîne de caractères à une couleur :

- Les villes en jaune
- Les rivières en bleu
- Les routes en gris
- Les abbayes en blanc
- Les campagnes en vert

Malheureusement, les tuiles de notre Carcassonne ne forment pas de dessins en les mettant bout à bout, cela est la conséquence du manque de temps que nous avons ressenti et vécu.

Pour les classes, le fonctionnement reste le même que l'interface graphique de Domino a l'exception de deux choses :

- Dans *ListeTuileViewCarcassonne*, à côté de chaque tuile du joueur se trouve un bouton supplémentaire ayant par exemple pour appellation «0/pion » qui permet de sélectionner la tuile d'indice 0 (dans le tableau de tuile du joueur) tout en y rajoutant un pion.
- Une classe *TuileViewPion* prend un argument de type *TuileCarcassonne* (comme *TuileViewCarcassonne*) et un entier qui représente l'indice du joueur dans la liste des joueurs associés au plateau. Ce dernier nous permet de définir la couleur du pion au milieu de la tuile. Comme nous avons au maximum 8 joueurs, il y a donc 8 couleurs possibles : orange, cyan, rouge, rose, vert, blanc, noir et jaune.

Pour conclure, nous tenons à vous faire part de ce que ce projet nous a apporté. Malgré les heures de travail qui parfois nous ont paru un peu longue et difficile, coder ce projet nous a permis de voir quelles étaient les réelles bases en java que nous avions acquises. Par exemple, lors des Travaux Pratiques que nous avons à faire durant l'année sont formés de questions qui orientent notre code et notre réflexion tandis que ce projet par une simple consigne nous a mené à devoir trouver une logique et une organisation qui nous étaient propres. Nous avons pu voir que nous étions capables de coder un jeu simple mais surtout de faire des interfaces graphiques qui nous paraissaient très compliqué à faire au premier abord.
Nous sommes, pour être tout à fait honnête avec vous, fière de ce travail que nous avons accomplie et fière malgré tout de ce que nous vous présentons, fière de pouvoir dire autour de nous que nous avons codé un jeu du début à la fin, que nous avons permis à une chose imaginaire de devenir aujourd'hui concrète.

Nous vous remercions pour votre temps, merci de nous avoir lu.

Elisa Ranjalahy et Laure Hamme