Dominos et Carcassonne

Projet de Programmation Orientée Objet

Nathan GUETTEVILLE et Benjamin DECKER

Double Licence Mathématiques et Informatique

Janvier 2023

Sommaire

Généralités

Traitement du cahier des charges

Notre tache dans ce projet était de réaliser deux jeux fonctionnels, l'un étant un domino carré et l'autre un Carcassonne. Il était nécessaire que l'utilisateur puisse choisir le jeu auquel il souhaitait jouer, le nombre de joueurs, si ces joueurs sont des humains ou des Intelligences Artificielles et enfin, dans le cas du domino, s'il voulais jouer au mode de jeu textuel dans la console ou bien au mode de jeu graphique.
Dans un premier temps, nous avons cherché à identifier les principales similarités entre ces deux jeux afin de regrouper au mieux les concepts proches et donc de gaspiller un minimum de temps et d'efforts dans l'écriture de certaines classes.
De ce fait, nous avons organisé notre code en trois package distincts, common, dominos et carcassonne. Les éléments de dominos et de carcassonne seront décris plus précisément dans les parties du même nom mais ils étendent tous une classe abstraite présente dans common que nous allons décrire ici.

En effet, les jeux de dominos carrés et le Carcassonne possède plusieurs similitudes. Tout d'abord, ces jeux se jouent avec des tuiles carrées. Elles possèdent donc quatre côtés que nous avons intitulés nord, sud, est et ouest pour comprendre rapidement l'orientation des tuiles car il faudra régulièrement les tourner pour pouvoir ensuite les placer sur le plateau, de plus les tuiles possèdent des getters pour accéder à leurs cotés, ce qui servira notamment à vérifier les conditions pour poser une tuile. Les tuiles possèdent donc une méthode qui les fait pivoter vers la droite « tournerDroite() » et une autre qui les fait pivoter vers la gauche « tournerGauche() ». De plus, pour savoir si elles ont été posées, elles possèdent un attribut booléen « posee ». Enfin, chaque tuile créée est unique et possède donc un numéro d'identification « id ».

Nous venons de voir que les tuiles, qu'elles servent dans les dominos ou bien dans Carcassonne, possédaient des côtés. Pour ce qui est des côtes, nous les avons tout simplement modélisés avec attribut String qui décrit le-dit côté ainsi que son getter.

Il est également évident que les deux jeux se jouent sur des plateaux en deux dimensions. Bien que nous ayons choisis d'avoir un nombre arbitraire de tuiles dans chacun de nos jeu, nous avons décidé de ne pas attribuer de taille arbitraire à nos plateau. En effet, nous ne voulions pas empêcher les joueurs, s'ils le souhaitaient et si c'était possible d'aligner la totalité des tuiles sur une seule ligne ou bien sur une seul colonne du plateau. Cependant, dans un soucis d'efficacité nous ne voulions pas créer un plateau de taille maximale pour chaque partie en sachant pertinemment que dans la grande majorité des parties le plateau n'allais pas être utilisé pleinement. Nous avons donc décidé de créer un plateau qui s'agrandit au fur et à mesure de la partie. Les plateaux ont donc deux attributs entiers « hauteur » et « largeur » qui représentent la hauteur et la largeur actuelle du plateaux. De plus, puisque les dimensions du plateau sont modifiées au cours de la partie, nous avons pris comme référentiel de coordonnées la position de la première tuile placée qui est donc placée en 0,0. Ces coordonnées sont donc représentées par les

entiers x0 et y0. De ce fait, on a également deux méthodes, « majX() » et « majY() » qui nous permettent de passer du système de coordonnées relatives utilisé par les joueurs au système de coordonnées absolue utilisé par le plateau. La classe Plateau possède aussi un attribut entier « tuileCentree » qui est l'id de la tuile sur laquelle on centre l'affichage. Enfin, on compte le nombre de tuiles placées sur le plateau avec l'entier « placees ».

Pour pouvoir jouer, il est aussi nécessaire d'avoir un certain nombre de tuiles et de pouvoir les piocher. Pour cela, nous avons implémenté la classe Sac qui contiendra les tuiles de la partie. Le sac doit être mélangé au début de la partie d'où la méthode « mélange() » et on doit pouvoir récupérer le contenu ou juste une tuile du sac située à l'indice i avec les méthodes « getSac() » et « getSac(int i) ».

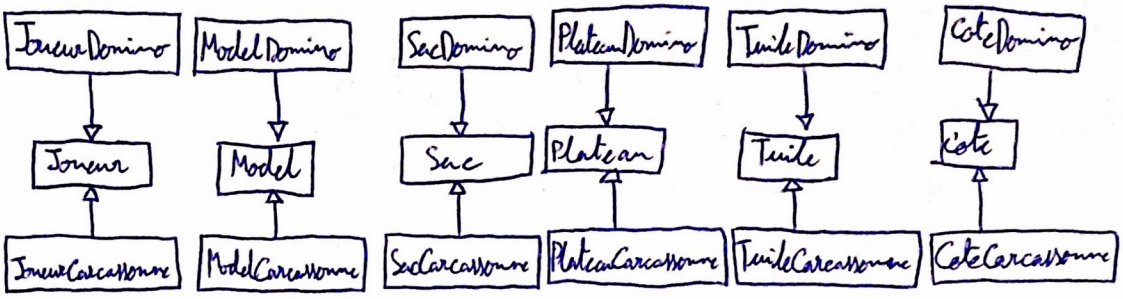
Ensuite, que serait un jeu sans joueurs ? Nous avons décidé de ne pas séparer en plusieurs classes les joueurs humains des IA et nous les différencions donc avec un booléen « humain » qui est vrai lorsque le joueur est un humain et est faux sinon. Les joueurs ont aussi un autre attribut booléen « abandon » qui permet de déterminer si le joueur est toujours en jeu ou pas, il est donc par défaut initialisé à faux et son setter ne permet que de le mettre à vrai. Les joueurs ont différentes méthodes qui implémentent les actions qu'ils peuvent faire. Il peuvent donc utiliser la méthode « placerTuile(int x, int y) » pour placer leur tuile sur le plateau en utilisant le système de coordonnées relatives à la tuile de départ. Il peuvent également se défausser avec la méthode « deffausser() » ou encore abandonner avec la méthode « abandonner ».

Afin d'utiliser ces différents éléments pour faire fonctionner les jeux, nous avons définit une classe Model qui possède deux attributs entiers, « nbJoueurs » qui est le nombre de joueurs au début de la partie et « tourDeJeu » qui permet de connaître quelle tuile du sac on doit piocher mais aussi de savoir quel joueur est en train de jouer.

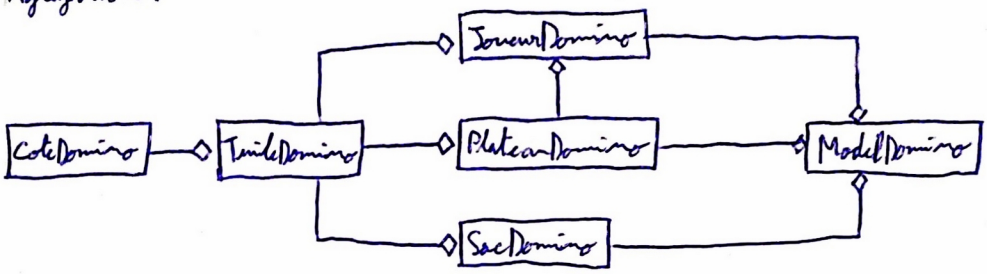
Enfin, la classe Launcher permet au joueur de décider à quel jeu il souhaite jouer, avec combien d'autres joueurs (humains ou IA) et lance le jeu. Il possède comme attributs un ModelDomino, un ModelCarcassonne, une VueCarcassonne ainsi qu'un ControleurCarcassonne. Ces attributs sont instanciés ou non en fonction des choix de l'utilisateur.

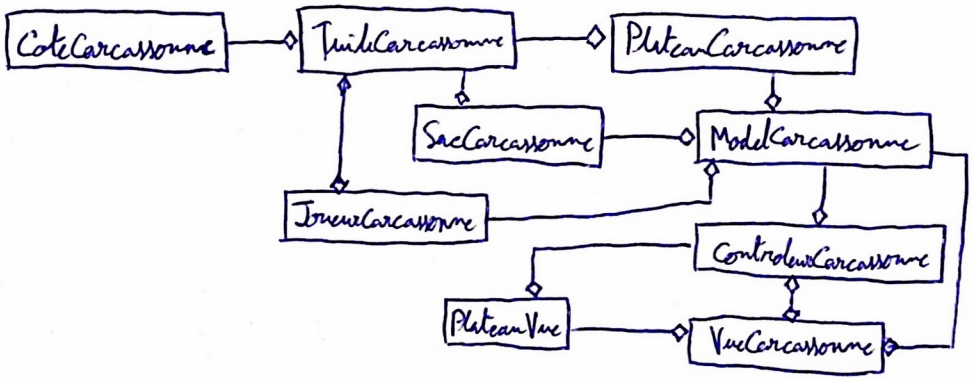
Modèle des classes

Hiérarchie :

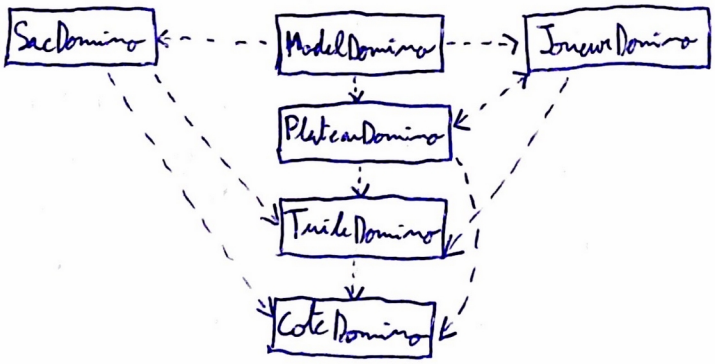


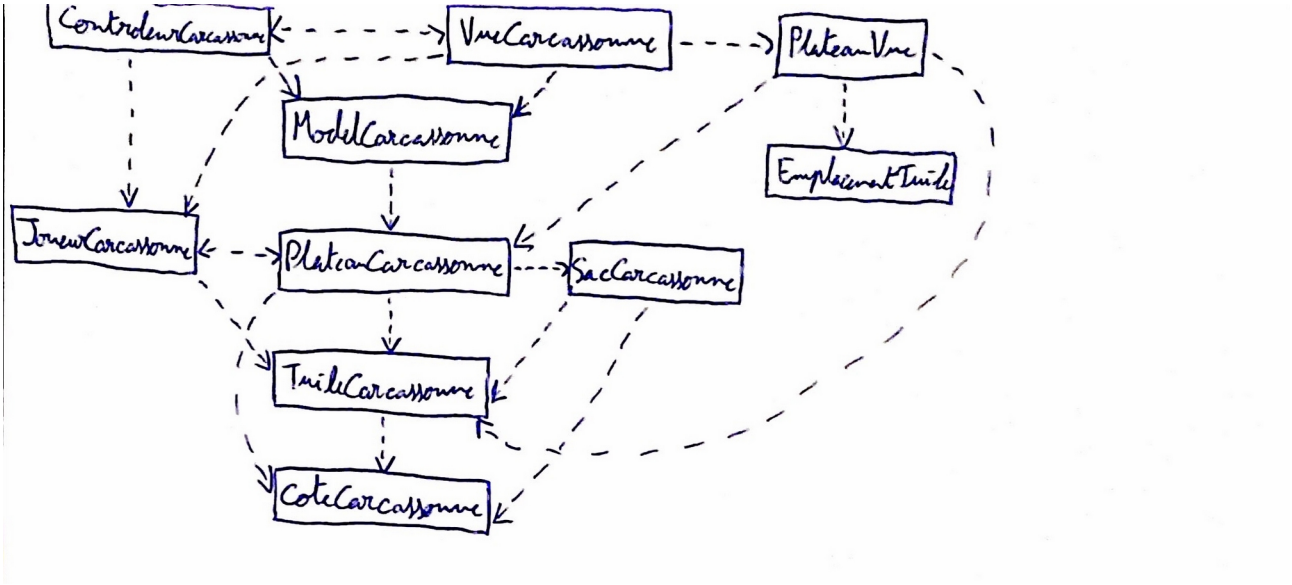
Agrégation :





Dépendance :





Dominos

Les tuiles des dominos possèdent chacune quatre CoteDomino. Il est possible d'en instancier de différentes façons. Tout d'abord, elles peuvent être instanciées avec des cotés aléatoires avec le constructeur sans argument. On peut aussi créer des tuiles avec des cotés connus en les passant en arguments d'un autre constructeur. Enfin, on peut créer des copies de TuilesDomino en passant en arguments au constructeur une tuile et son id. Ces tuiles possèdent effectivement un numéro d'identification unique qui sers notamment à savoir sur quelle tuile on souhaite centrer sa vue dans le mode textuel.

Les CoteDomino qui composent les TuileDomino sont représentés par un String de longueur trois composé de chiffre aléatoires compris entre 0 et 4. Nous avons choisis d'avoir des cotés générés aléatoirement afin que les parties ne se ressemblent pas. Il est également possible de créer la copie d'un coté existant et même son inverse avec le constructeur qui prend un CoteDomino « c » et un booléen « inverse ». Il est en effet nécessaire de pouvoir créer des cotés inversés car lorsqu'on pose une tuile, pour que deux cotés correspondent ils doivent être l'inverse l'un de l'autre. Les CoteDomino possèdent donc deux getter, un qui renvoie le String les représentant et l'autre renvoyant le caractère à l'indice i (compris entre 0 et 2) du String qui les représente. Enfin, ils possèdent une méthode « getInverse() » qui renvoie le String inversé de celui qui les représente et une méthode « sommeChiffre » qui fait la somme des chiffres du côté.

Le plateau est composé d'une grille qui est une ArrayList<ArrayList<TuileDomino>>. Il peut être initialisé vide ou bien avec une tuile de

départ. Il possède une méthode « placer » qui prend une tuile et des coordonnées relatives à la tuile de départ. Cette méthode renvoie vrai si la tuile a bien été posée et faux sinon. La méthode « placableGeneral » vérifie si la tuile est plaçable en agrandissant le tableau au plus une fois dans les directions nécessaires. Elle vérifie également que la case où placer la tuile n'est pas déjà occupée. La méthode « placableTuile » vérifie les conditions de placement de la tuile. C'est à dire si elle a bien au moins un voisin non vide et que leurs cotés correspondent. La méthode « placableIA » fait la même chose que la méthode précédente à l'exception qu'elle n'affiche rien dans la console et n'est appelée que par des IA. La méthode « ListVoisins » crée un tableau de tuiles contenant les voisins de la tuile dont on a donné les coordonnées absolue. Cette méthode est utilisées dans les deux précédentes. La méthode « sommeCotesAdja » effectue le calcul des points pour une tuile qui vient d'être posée. La fonction « allNull » vérifie si toutes les tuiles d'un tableau de TuileDomino sont null. La méthode « afficherLigne » permet d'afficher dans la console une ligne du plateau et la méthode « afficher() » affiche le plateau dans sa totalité. Cependant, nous craignions que le plateau prenne parfois trop de place pour être afficher convenablement donc nous avons décidé qu'il n'afficheraient que les tuiles autour de celle sur laquelle on se centrerait. Ces méthodes sont « afficher(int id) », « afficherLigneVide » et « afficherPetiteLigne ». La fonction « getXY(int id) » permet d'obtenir les coordonnées absolues de la tuile dont on a donné l'id en paramètre. La méthode « deplacer(int direction) » met à jour l'id de la tuile sur laquelle on est centré s'il existe une tuile dans la direction indiquée. Et enfin, nous avons des méthodes qui permettent d'agrandir le plateau dans toutes les directions.

Le SacDomino possède un tableau de TuileDomino de taille 50, nous avons choisis d'imposer un nombre de tuile précis à l'image des jeux de plateau classiques. Nous avons vu plus haut que les dominos étaient générés aléatoirement. Lors de l'initialisation du sac, on génère tout d'abord 15 cotés aléatoires ainsi que leurs inverses et à partir des ces 30 cotés, on génère 50 dominos et enfin, on mélange le sac pour qu'il soit prêt à être utilisé.

Les joueurs de dominos possèdent une TuileDomino ainsi qu'un PlateauDomino sur lequel il vont jouer et un score pour savoir qui a gagné. En plus des méthodes héritées JoueurDomino possède une méthode « placerIA » qui place la tuile d'une IA à la première position possible et qui met à jour le score. Il y a également les getters de la tuile et du score et le setter de la tuile.

La classe ModelDomino possède comme attributs, un PlateauDomino sur lequel la partie va se dérouler, une ArrayList<JoueurDomino> qui contient tous les joueurs participants, un JoueurDomino qui est le joueur à qui c'est le tour de jouer, un SacDomino où l'on piochera les tuiles et un entier qui contient l'id de la dernière tuile sur laquelle on s'est déplacé (par défaut, la tuile de départ). Puisque nous n'avons pas eu le temps de

réaliser l'interface graphique du jeu, le model ne possède pas d'attribut de vue et le controller est directement intégré à la méthode « play() » qui lance la partie et gère toutes les actions des joueurs ainsi que la fin de la parite. Le tableau de joueurs est initialisé vide et est rempli dans la classe Launcher. Le ModelDomino possède donc des getters pour son tableau de joueurs et son plateau. Il possède également une méthode « victoireParAbandon() » qui vérifie qu'il y a plus d'un joueur qui n'a pas encore abandonné.

Carcassonne

Les TuilesCarcassonne possèdent un tableau de quatre CoteCarcassonne qui représentent les côtés nord, est, sud et ouest. Elles ont également un attribut entier « id » unique, un booléen « abbaye » qui renseigne sur la présence d'une abbaye sur la tuile, un autre booléen « pion » qui indique si un pion a été posé sur cette tuile, un JoueurCarcassonne qui est l'éventuel propriétaire du pion sur la tuile et enfin, un attribut BufferedImage « image » dans lequel on stock l'image représentant la tuile. Nous avons décidé d'implémenter les tuiles de bases du jeu Carcassonne à l'exception de celles possédant un bouclier et de quelques tuiles qui auraient eu la même représentation dans notre jeu que d'autres. Cela nous à donc donné un total de 17 types de tuiles différentes. Les tuiles sont donc instanciées avec un entier représentant un de ces 17 types de tuile. Cette classe possède donc les méthodes héritées de sa classe mère ainsi que les getters et les setters de ces attributs. Elle implémente également la méthode « placerPion(JoueurCarcassonne j) » qui place un pion appartenant au joueur j sur la tuile.

Nous avons choisis d'implémenter trois différents types de terrain pour les CoteCarcassonne, à savoir les routes, les villes et les champs. Chaque coté peut donc prendre une de ces trois valeurs.

La classe PlateauCarcassonne a, à l'exception des méthodes d'affichage, de la méthode « sousTableau » qui renvoie un tableau de TuileCarcassonne de taille 5 par 5 centré sur la tuile actuelle et de la méthode « placerPion » qui permet de placer un pion sur une tuile du plateau, sensiblement les mêmes méthodes que la classe PlateauDomino mais appliquées à une grille de TuileCarcassonne. Cela nous a donc fait prendre conscience qu'il aurait sûrement été plus judicieux d'avoir une seule classe Plateau utilisant la généricité pour avoir une grille d'objets étendant la classe Tuile. Cependant, par manque de temps, nous n'avons pas changé notre implémentation initiale.

Pour le jeu Carcassonne nous avons voulu nous rapprocher le plus possible du jeu de base, le SacCarcassonne possède donc un tableau de TuileCarcassonne de taille 72 et générant les mêmes tuiles que dans le jeu de plateau à l'exception des tuiles avec bouclier que nous avons simplement remplacées par leur équivalent sans bouclier. Le sac est ensuite mélangé à l'exception de la première tuile qui sera la première posée.

À l'image des JoueurDomino, les JoueurCarcassonne ont un attribut TuileCarcassonne et un PlateauCarcassonne mais également un attribut entier « pions » qui représente le nombre de pions partisans qu'il leur reste ainsi qu'une Color qui représente la couleur de leurs pions et un entier « num » qui correspond à leur numéro d'identification. Les JoueurCarcassonne ont donc en plus des méthodes qu'ils héritent de Joueur, les getters de leurs attributs, le setter de la tuile et de la couleur, la fonction « placerIA » qui a le même rôle que dans JoueurDomino et la méthode « placerPion » qui diminue de un le nombre de pions restant au joueur.

Le ModelCarcassonne possède comme attributs un PlateauCarcassonne, un tableau de JoueurCarcassonne, un JoueurCarcassonne actuel à qui c'est le tour de jouer ainsi qu'un sacCarcassonne de tuiles. Il possède également des getters et des setters associés à ses attributs ainsi qu'une méthode « start() » qui initialise le premier joueur et donne une couleur et un numéro à chacun d'entre eux, « piocher() » qui fait piocher le joueur actuel, « incrementeTour() » qui passe au tour suivant en sautant les joueurs qui ont abandonné, « setCouleurEtNum() » qui attribue une couleur et un numéro à chaque joueur et des méthodes « tuilesRestantes() » et « joueursRestants() » qui comptent respectivement les tuiles restantes dans le sac et les joueurs qui n'ont pas abandonné.

La VueCarcassonne est divisée en quatre partie, en haut, c'est là qu'est affiché le plateau et où l'on clique pour poser sa tuile. En bas à gauche c'est là que l'on peut voir sa tuile et son orientation. En bas au centre, se trouvent les boutons qui permettent de faire pivoter la tuile, poser un pion sur la tuile qu'on a en main et se déplacer sur le tableau. Enfin, en bas à droite, il s'agit de la pioche qui lorsqu'on clique sur la tuile face cachée, place la tuile dans notre main. En plus des différents attributs JPanel, JLabel et JButton que la vue possède, elle a également un attribut ModelCarcassonne ainsi qu'un ControleurCarcassonne. Elle implémente aussi plusieurs méthode d'update qui permettent de mettre à jour différents éléments de la vue comme le plateau affiché, l'orientation des tuiles, la pioche ou encore la main du joueur… Enfin, elle implémente une méthode « fin() » qui est appelée à la fin de la partie et qui affiche un message de fin.

Le ControleurCarcassonne possède des attributs ModelCarcassonne et VueCarcassonne et permet au joueur d'effectuer des actions comme piocher une tuile, la

tourner, la placer ou bien se défausser tout en mettant à jour les données du model et l'affichage de la vue. Il a également un attribut « piochee » qui détermine si l'action de pioche à déjà été effectuée dans ce tour auquel cas, il passe au tour suivant. Pour les méthodes, il y a tout d'abord « piocher() » qui permet de tirer une tuile et de gérer le tour des joueurs non-humains. La méthode « defausser() » permet de se défausser. « placerPion() » place un pion sur la tuile que l'on a en main. « Pivot() » fait pivoter la tuile que l'on possède. La méthode « placerTuile(int x, int y) » place si possible la tuile dans notre main sur le plateau avec des coordonnées relatives à la tuile de départ. La méthode « abandonner() » fait abandonner le joueur courant et la méthode « finDePartie() » affiche un message de fin.

La classe PlateauVue nous sert à afficher partiellement le plateau dans la vue du jeu. Elle possède un GridLayout de 5 par 5 dans lequel on affiche les cases d'un sous tableau du plateau centré sur la tuile actuelle et de taille 5 par 5 et un ControleurCarcassonne. La méthode « updatePlateau(TuileCarcassonne[] tab, PlateauCarcassonne pCarcassonne) » met à jour la vue en fonction du sous tableau tab et initialise des EmplacementTuile qui implémentent MouseInputListener afin de pouvoir placer nos tuile en cliquant dessus.

Les EmplacementTuile représente les tuiles du sous tableau affiché dans la vue du jeu. Ils ont comme attributs une hauteur et une largeur, des coordonnées sur le sous tableau : « xgrille » et « ygrille », des coordonnées relatives à la tuile d'origine : « xorigin » et « yorigin » ainsi qu'un booléen « occupe » qui détermine si la tuile représentée est vide ou bien posée. En plus des getters associés à ses attributs, EmplacementTuile possède une méthode « fill(BufferedImage img) » qui permet d'afficher l'image de la tuile que l'on pose à cet emplacement.