JEREMY MOOTOOVEEREN
&
SAMI BELMELLAT

RAPPORT DE PROJET

JANVIER 2023

PRÉPARÉ PAR
JEREMY MOOTOOVEEREN ET
SAMI BELMELLAT

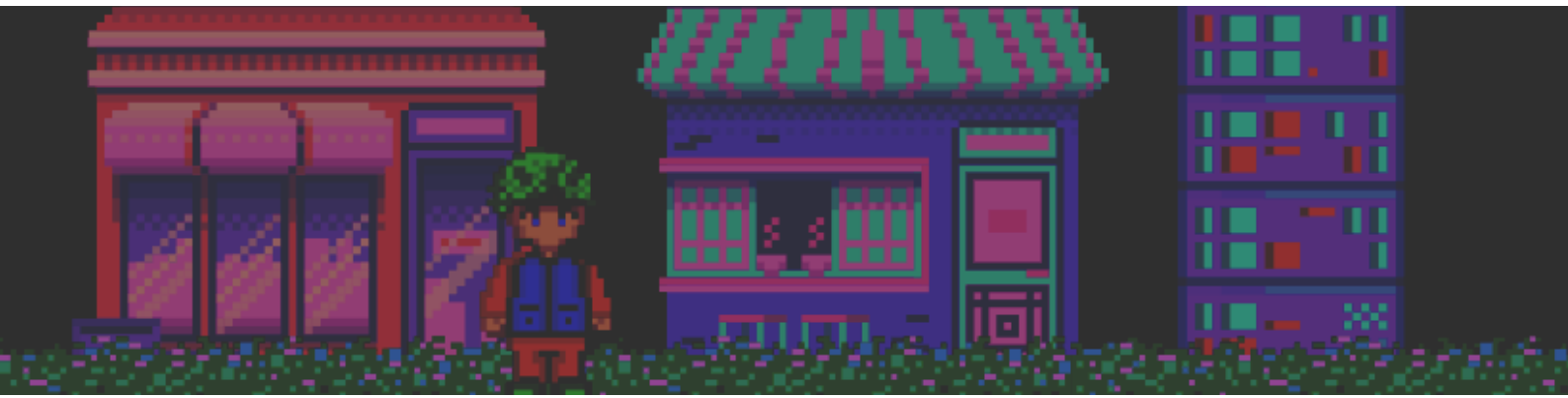
SOMMAIRE

Le déroulé de ce rapport :

1) Cahier Des Charges

2) Problèmes Connus

3) Représentation Graphique du Modèle de Classes

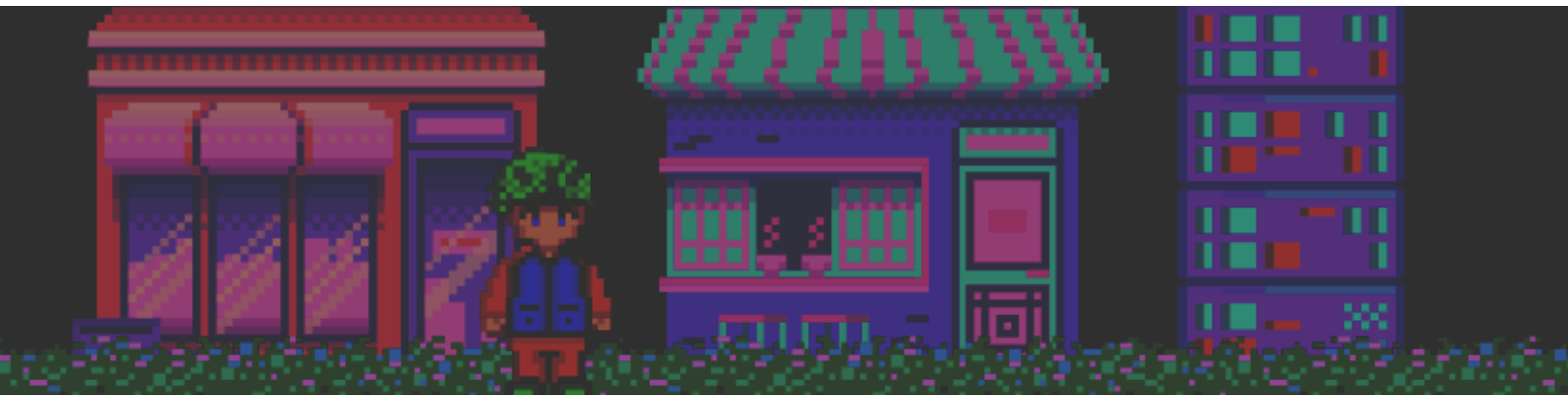


CAHIER DES CHARGES

Environnement de Jeu

Nous avons pu utiliser au maximum les librairies AWT et SWING en utilisant un découpage des éléments de l'interface en JFrame, JPanel et JLabel. Pour ce qui est du menu de départ, nous avons choisi de faire une classe représentant le menu principale, une autre classe pour le menu principale et un GameView lié à l'interface de jeu. Ce découpage a permis de faire des transitions qualitatives entre chacun de ces découpages du jeu et nous permettant de pouvoir agir individuellement sur chacun d'eux. Pour le style, nous utilisons des images mais aussi les librairies de dessin comme Graphic2D. Le GameView sera détaillé ci-dessous. En fin de rapport, vous pourrez voir un schéma qui décrira notre implémentation au complète du jeu avec les classes et les relations entre elles.

Menu Principale

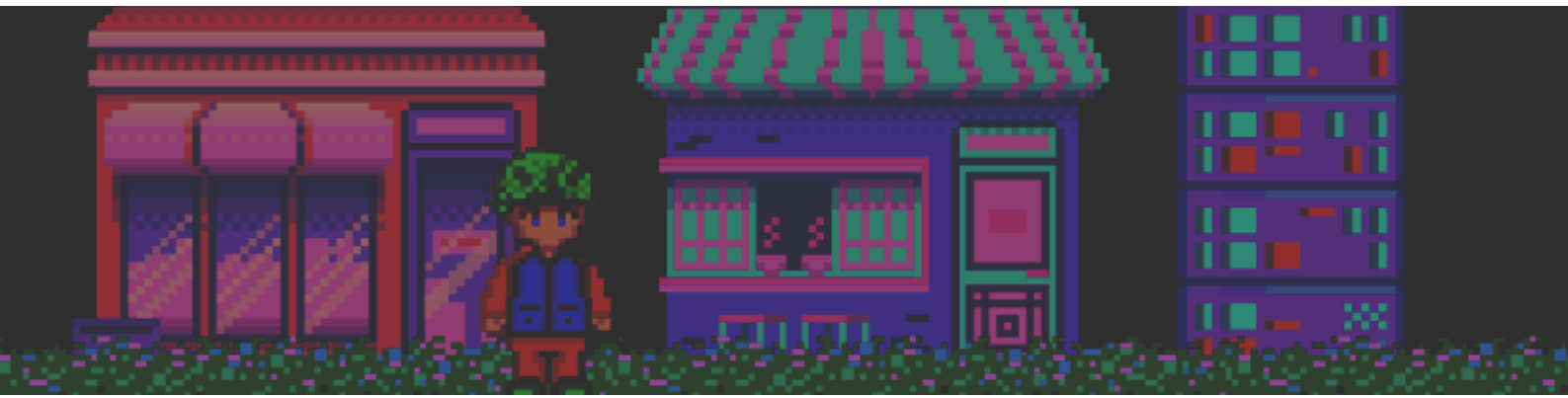


CAHIER DES CHARGES

Environnement de Jeu

Menu Interne

GameView



CAHIER DES CHARGES

Implémentation des règles de Domino

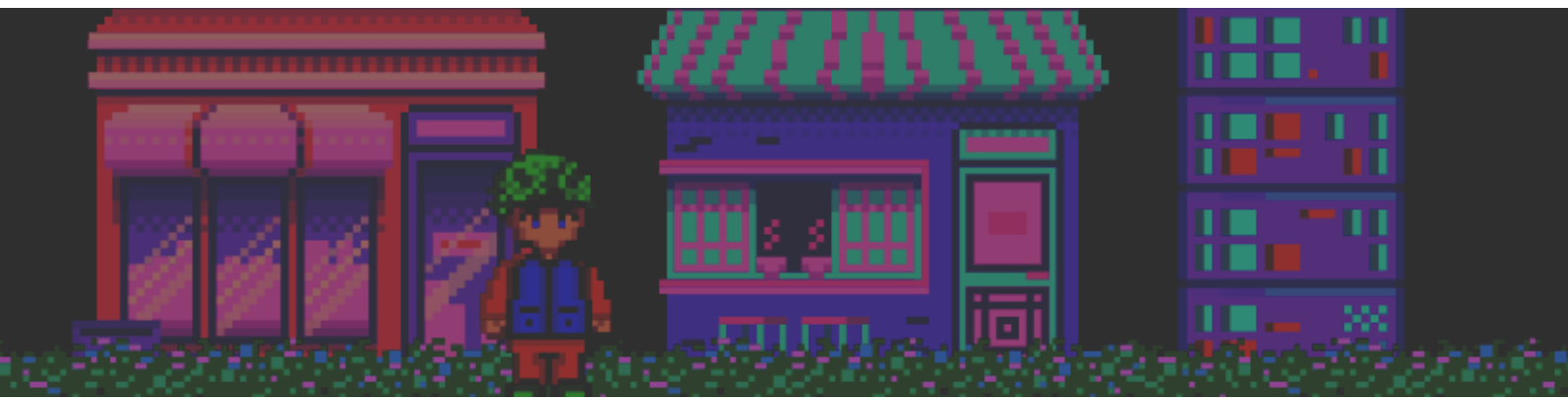
Nous avons pu implémenter toutes les règles du jeu Domino , nous avons Créé une Classe TuileD qui Extend de la Classe Abstraite Tuile, qui a un ensemble de méthodes permettant de :

- ❄ Savoir si le domino actuel peut se placer à coté du domino passé en paramètre
- ❄ Compter le nombre de points que le joueur peut avoir marqué
- ❄ Faire tourner la tuile

TuileD a aussi une méthode statique Permettant de générer toutes les tuiles avec laquelle une partie va être jouée, elle possède comme attribut un tableau bidimensionnel d'entier de taille 4x3 , tel que le tableau d'indice 0 représente le haut , 1 la droite... La Classe Plateau permet de gérer les emplacement des tuiles ainsi que la possibilité de les poser à des coordonnés 'x' et 'y' additionné à la gestion du contenant toutes les tuiles du jeu.

Nous avons aussi crée deux Classes JoueurH ainsi que JoueurB représentant respectivement le joueur Humain et le Bot, toutes les deux sont des Classes filles de la Classe Abstraite Joueur qui possède seulement deux méthodes abstraites :

- ❄ play() : qui fait jouer le joueur
- ❄ playTerminal() : qui fait jouer le joueur selon l'affichage sur le terminal



CAHIER DES CHARGES

Implémentation des règles de Carcasonne

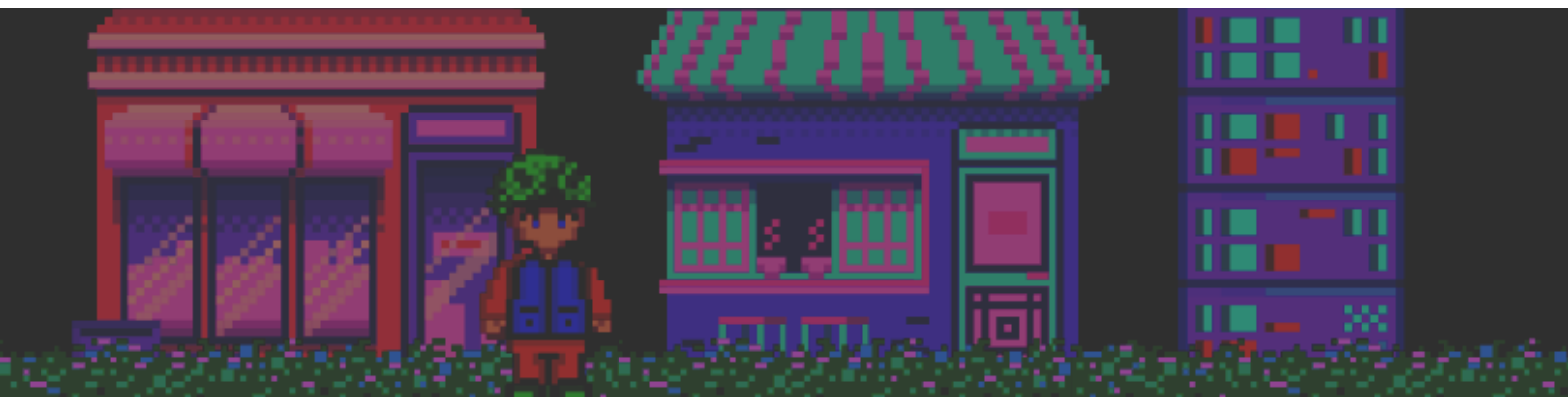
Cette implémentation a été faite presque de la même manière que pour l'implémentation des règles de Domino , avec quelques implémentation supplémentaires pour les nouvelles règles (placement des pion) .

Nous avons créé comme pour l'implémentation précédente une Classe TuileC qui Extend de Tuile , la Classe Plateau ainsi que les deux Classes Joueurs seront les mêmes pour Carcassonne , car elles manipulent toutes des Tuile référence .

Pour TuileC , elle a cette fois-ci un tableau de char de taille 4, qui aura dedans soit : 'R' => Route , 'C' => Champs , 'V' => Ville .

Comme pour TuileD l'élément présent à l'indice 0 représente le face du haut , 1 la droite...

Ainsi qu'un champs int pion qui sera l'identifiant propre a chaque joueur , qui est initialisé par défaut a -1 , et pour finir un boolean abbaye .



CAHIER DES CHARGES

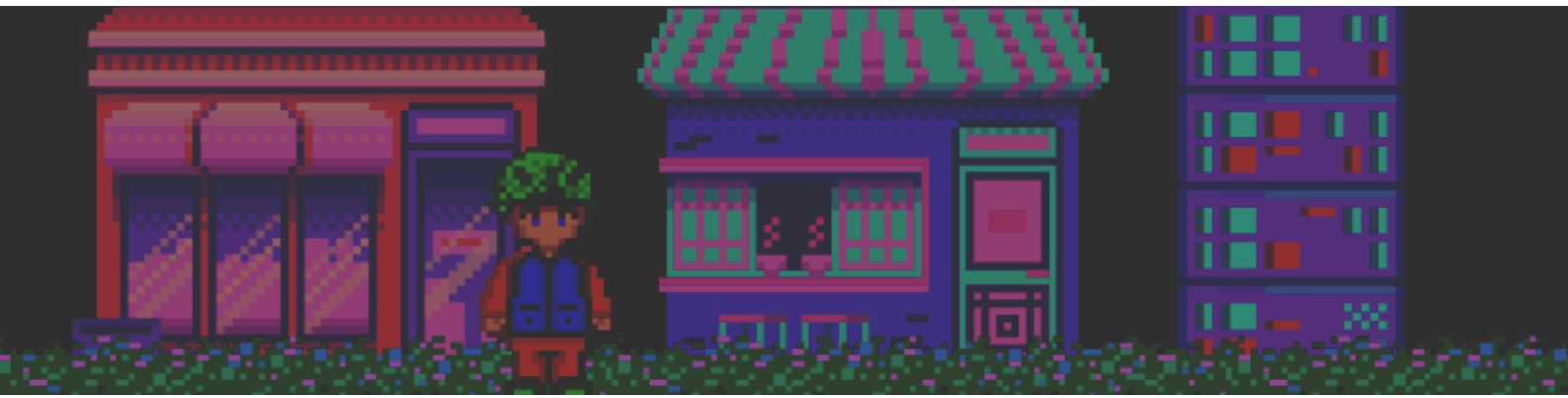
Mode de Jeu Texte pour Domino

Nous avons créé une Classe Spéciale pour cela . Le fichier StartTerminal.java sert a faire jouer l'utilisateur directement sur le terminal , un texte apparaît pour lui présenter le jeu , et des questions lui seront posé pour faire entrer depuis le clavier le nombre de joueurs en premier lieu , puis le nombre de Bot parmi eux , (Nous avons pris soin de vérifier que les informations que l'utilisateur donne sont toujours cohérentes).

Par la suite on crée un plateau , et un tableau de joueurs qu'on va initialiser , en suite tant que la méthode boolean finDeGame() de Plateau ne renvoie pas vrai , le jeu continue .

Les joueurs ayant abandonne ne joueront pas , et pour les jouers humain quand leur tour vient , le plateau de jeu leur est affiché ainsi que la Tuile qu'ils ont en leur possession , puis on leur laisse le choix de choisir la prochaine action qu'ils feront :

- ❄ Poser la Tuile => ils devront entrer des coordonnes valides
- ❄ Passer leur tour
- ❄ Abandonner
- ❄ Tourner à gauche ou à droite la leur Tuile



CAHIER DES CHARGES

Mode De Jeu Graphique Pour Domino Et Carcassonne

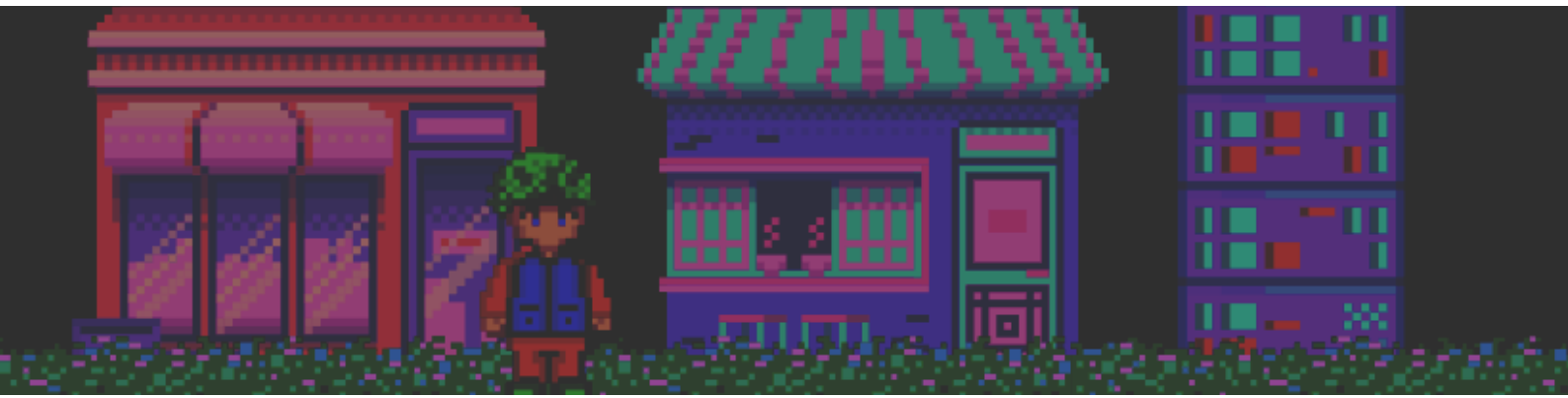
Nous avons suivi le design pattern de MVC , le modèle pour Carcassonne et Domino est le même , seule les Tuiles utilisés déterminent le jeu auquel on jouer , les Classes utilisés pour l'affichage sont aussi commune aux deux jeux . Afin de respecter le MVC nous avons crée trois Classes :

- GameModel
- GameView
- Controller
-

(*)GameModel : Contient 3 champs , une instance de Plateau , un tableau de Joueur ainsi qu'un champs qui encapsule un Joueur qui est le joueur actuel , elle possède des fonctions qui permettent de renvoyer des informations sur l'état interne du jeu par exemple : la fin du jeu , le prochain joueur...

(*)GameView : Qui fera en sorte d'afficher l'état interne du jeu sur l'écran de manière lisible et clair pour l'utilisateur , chaque JoueurH a sa propre interface de jeu avec son score affiche et des boutons permettant d'effectuer les différentes actions du jeu , il y aura aussi le plateau qui sera affiché et qui se met a jour a chaque fois qu'un joueur pose une Tuile

(*) Controller : Comme son nom l'indique fait office d'intermédiaire entre la Vue et le Modèle ici entre GameView et GameModel , afin de séparer au plus ces deux dernier , il s'occupe de faire transmettre l'état interne du jeu pour faire modifier la Vue et se mettre à jour , le Vue peut aussi passer par Controller pour effectuer des vérification dans le modèle et selon le résultat le Controller s'occupe de soit notifier la Vue pour qu'elle se mette à jour en fonction du Modèle .



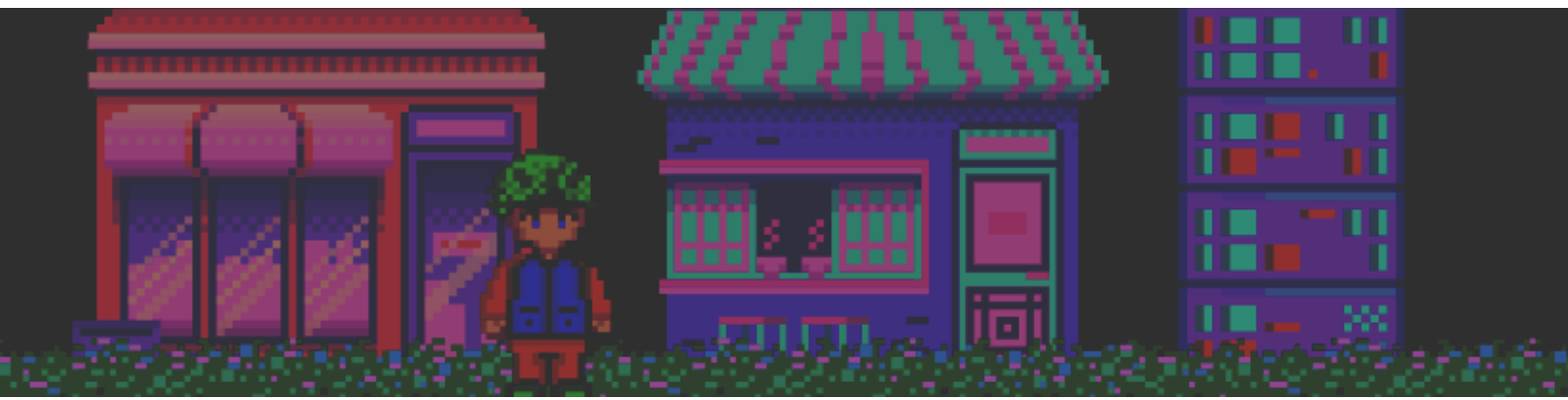
PROBLÈME CONNUS

Classe Plateau

En premier lieu , on est parti sur un tableau bidimensionnel de Tuiles de 40x40 , en plaçant le premier domino au centre ce qui fait une marge de 20 Tuiles vers la gauche droite haut et bas , ce qui peut poser un problème si jamais on a pu poser 19 dominos a partir du centre vers une seule direction et qu'on arrive a la limite du plateau , on ne peut plus poser même si c'est possible , comme solution on a choisi de doubler les dimensions du plateau en partant plus sur un tableau de 80x80 , ce qui laisse une marge de 40 dominos vers tous les cotés .

L'affichage du plateau était aussi un problème , comment faire pour afficher seulement les parties dont on a besoin sans afficher toutes les cases ? Et comment savoir à partir de quel lignes et colonnes on doit afficher sans passer par une fonction avec une complexité exponentielle ?

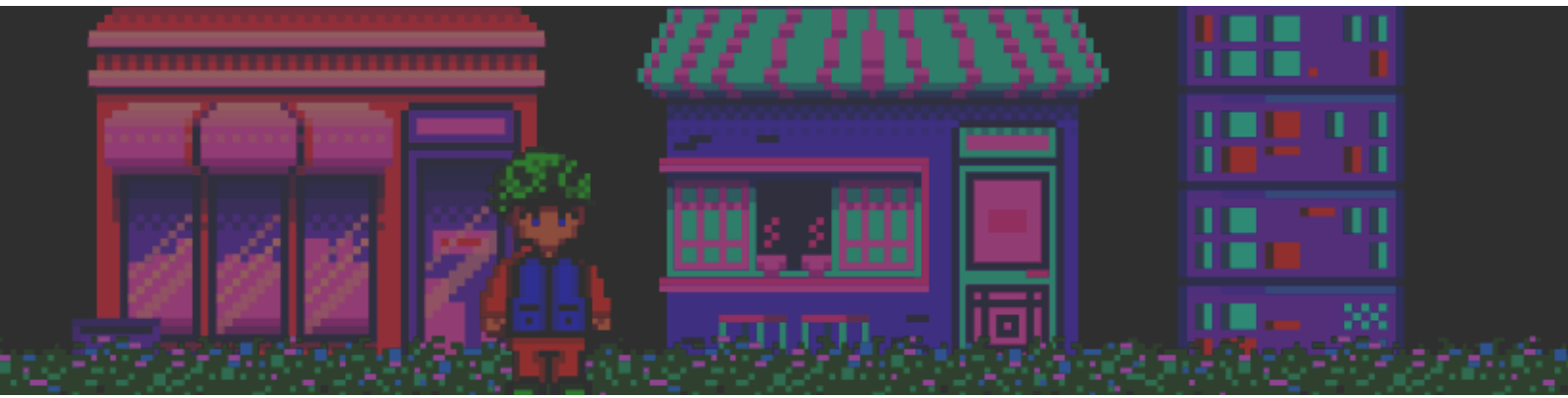
On a pu résoudre cela grâce à un tableau de taille 4 (x0 ,y0, x1, y1) qui stocke l'indice des première lignes et colonnes vides en partant de la position (40;40) .
Exemple : si on a une tuile en (40;40) et (39;40) alors : (x0,y0,x1,y1) aura : (38,39,41,41)



PROBLÈME CONNUS

Classe TuileDView/TuileCView

Ces classes, héritant de JLabel, était faite pour représenter la tuile d'un domino ou d'une image de carcassonne. Le problème était de savoir comment adapter les tuiles en fonction des informations donné en argument aux classes (les faces des dominos ou les caractéristiques des images pour Carcasonne).
Nous avons opté, dans le cas du domino, sur un dessin des différentes faces du domino sur une image vierge pour fournir une nouvelle image avec les faces en question. Pour le cas de Carcasonne, le fonctionnement de la classe est plus simple mais notre travail s'est effectué en amont car nous avons dû renommer chaques images en fonction de ses caractéristiques.

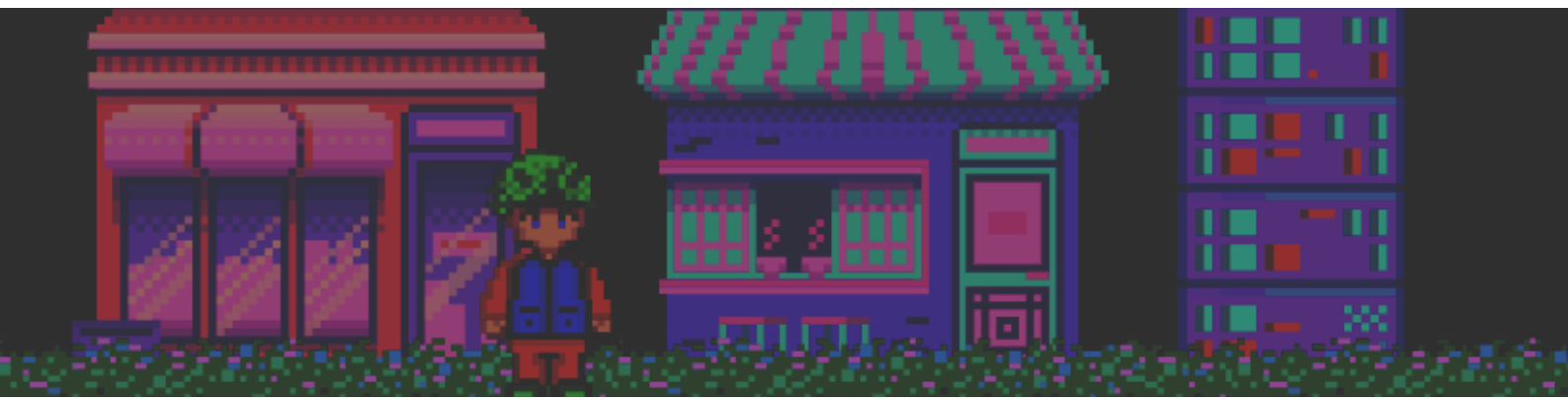


PROBLÈME CONNUS

PROBLÈMES D'IMPLÉMENTATION

L'interface Graphique

L’interface graphique était un des points qui était mis en exergue dans notre réflexion. En effet, il était essentiel pour nous de savoir faire le pont entre l’interface terminal et l’interface graphique afin d’y retrouver les mêmes fonctionnalités mais surtout pour qu’elles soient bien implémentées dans les deux cas. Nous avons eu une grosse réflexion et plusieurs difficultés sur la réalisation du plateau. En outre, fallait-il faire un plateau dont la taille est adaptative en fonction des tuiles qui y sont placées ou alors fallait-il faire un plateau dont la taille est initialement finale et complète où il est possible de mettre les dominos partout. Nous avons opté pour la première option mais cela n’était pas une partie de plaisir car nous avons eu beaucoup de difficulté à pouvoir adapter la taille des tuiles en fonction de la taille des cases du plateau. La solution a été de réaliser une fonction reSize qui faisait une adaptation des tuiles à chaque ajout de tuile. Notre interface graphique se base sur le principe du MVC (qui sera détaillé plus bas) et c’est exactement grâce à cela que l’interface graphique est utilisable dans les deux parties du projet (pour Domino et Carcasonne)



PROBLÈME CONNUS

PROBLÈMES D'IMPLÉMENTATION

Les Ressources (Images)

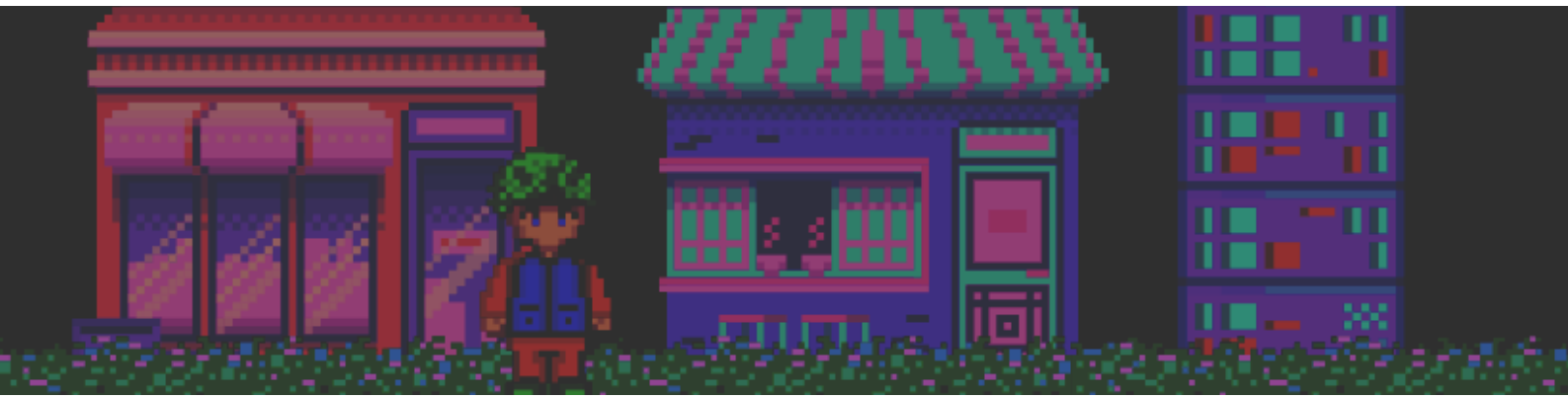
Un autre problème notamment lié à l'interface graphique a été l'utilisation d'images en guise de tuile. Le raisonnement expliqué ci-dessus concernant TuileDView et TuileCView avait un point important : la modification des images. Dans le cas de Carcasonne, la réflexion s'est basé autour de l'identification de chaque image en fonction de son nom et de ses caractéristiques.

Nous avons décidé de résoudre cela en donnant des noms avec des caractères liés aux routes, champs abbaye, etc, qui permettent d'identifier au mieux l'image. Le cas du domino été plus simple mais fastidieux car nous devions écrire sur l'image "neutre" du domino, les faces de celui-ci et créer une nouvelle image.

L'implémentation de ces classes devait être en communication avec le contrôleur afin de toujours adapter les tuiles en fonction des données fournies. C'est là qu'intervient le MVC.

Implémentation du MVC

C'était le point central du projet car une fois implémenté , on pouvait dire que c'était fini , les difficulté venaient plus du fait de Comment tout relier la Vue et le Modèle? Est ce que c'est mieux de mettre le modèle en champs de la Vue ?
Comme solution on s'est mis d'accord pour ajouter le Controller , qui nous servira de passerelle entre la Vue et le Modèle .



REPRÉSENTATION GRAPHIQUE DU MODÈLE DE CLASSES