Projet POOIG 2019-2020

Kevin DANG
Julia ARNOUX

1-Modélisation

S’organiser

Dès que notre groupe a été formé nous avons immédiatement créé un dépôt git afin de faciliter notre organisation. Nous nous sommes ensuite immédiatement attelés à la modélisation en réfléchissant chacun de notre côté avant de mettre nos idées en commun. La modélisation (telle que vous pouvez en voir la schématisation dans la dernière page de ce rapport) a rapidement émergé avec deux concepts majeurs à sa base. Premièrement, garder le fonctionnement du jeu le plus indépendant possible de son affichage, deuxièmement, base notre modélisation sur un découpage du plateau en différentes zones.

Un découpage en zones

Après avoir étudié longuement les règles du jeu AZUL, nous avons conclu que le jeu consiste principalement à déplacer des tuiles colorées entre différentes zones selon certaines règles précises. On distingue alors rapidement les zones invisibles au joueur (Sac et Défausse), où les tuiles sont stockées, les zones avec lesquelles chaque joueur peut interagir (Fabriques et Centre du plateau) et les zones privées à chaque joueur (Lignes de Motif, Mur, Plancher). Notre but a donc été de tout d’abord modéliser de simples déplacements de tuiles entre zones puis de rajouter peu à peu des fonctionnalités sous forme de fonctions et héritage, correspondant aux diverses règles régissant le déplacement des tuiles dans Azul.

2-Implémentation

Le jeu

Comme expliqué plus haut, nous avons commencés par implémenter des fonctions et classes très simples avant de complexifier notre code au fur et à mesure. Nous avons fait le choix d’utiliser des ArrayList pour stocker le contenu de chaque Zone. Bien qu’imposant quelques contraintes, le fait de conserver une structure commune à toutes les zones nous a permis d’éviter de nombreux problèmes et erreurs potentielles. Une situation toutefois redondante que nous avons rencontrée est celle des types effectifs et déclarés, due à la nature même de notre modélisation. Il est rapidement apparu, que la fonction principale

permettant de récupérer une Zone du plateau devait renvoyer un objet de type Zone sans quoi elle serait trop spécifique, cependant cela nous empêchait d'accéder directement aux fonctions propres aux classes héritant de la classe mère Zone. Toutefois lorsque ce problème est survenu nous étions déjà bien avancés dans l'implémentation et avons décidé de nous accommoder de cette situation plutôt que de se risquer à des corrections qui auraient pu casser tout notre code, en s'assurant tout simplement de bien convertir la zone récupérée en un objet de son type effectif via le transtypage (ou cast) par exemple.

REMPLIR LES LIGNES DE MOTIF

Après les fonctions de bases comme celles permettant le déplacement de tuiles entre diverses zones sans réelle restriction nous nous sommes attelés aux deux plus grosses fonctions de notre implémentation. Premièrement, celle permettant à un joueur de remplir une ligne de motif avec des tuiles piochées dans une fabrique ou au centre du jeu. La principale complexité de cette fonction était de n'omettre aucun détail. Il y a la vérification de points tels que s'assurer que la ligne n'en contient pas une autre couleur ou n'est pas déjà pleine, que la zone d'origine sélectionnée est bien pleine etc. Ensuite il faut penser à envoyer les tuiles en trop au plancher et celles restantes au centre du plateau. La réelle difficulté cependant viens plus des cas particuliers, gérer les tuiles Joker, gérer l'envoi immédiat au plancher etc.

Cela nous aura nécessités de nombreux ajouts et corrections avant d'arriver à un résultat satisfaisant.

REMPLIR LE MUR

L'autre gros morceau était la fonction permettant de décorer le mur. Cette fois-ci nous avons surtout dû faire preuve d'ingéniosité pour permettre l'implémentation des tuiles joker sans trop allonger le code. Au final, il s'agit encore de faire attention à de nombreuses vérifications, la ligne de motif est-elle bien pleine ? Sommes-nous dans la variante où le joueur peut choisir comment décorer son mur ? etc. Plutôt que d'avoir deux fonctions différentes l'on soit dans la variante ou non, nous avons décidé de donner un entier en argument à la fonction. Si l'on se trouve dans la variante il correspondra à

l'indice où doit être placée la tuile sur la ligne du mur, sinon il vaudra -1. Ainsi le code est beaucoup plus efficace.

LES VARIANTES

Dans notre version d'AZUL, nous avons implémenté deux variantes aux règles suggérées par le sujet du projet. La première, permettant aux joueurs de placer librement les tuiles sur leur mur a été la plus complexe à implémenter car modifiant le plus le fonctionnement du jeu. Nous avons décidé de différencier la variante du jeu normal en créant un tableau répertoriant les valeurs de couleurs attendues pour chaque case du mur lorsque nous ne sommes pas dans la variante. Il a surtout été question par la suite de s'assurer que le joueur ne pouvait pas se retrouver bloqué. Pour cela voilà la solution que nous avons choisie. Un joueur peut décider de vider entièrement une ligne de son mur. Il sacrifie donc ses progrès mais en échange de quoi il peut de nouveau progresser dans la partie. Pour des raisons de praticité il était peu ergonomique de proposer cette option à tout instant dans l'affichage textuel, nous avons donc pris le parti de ne la suggérer que lorsqu'un joueur se retrouve effectivement bloqué quant il s'agit de remplir son mur.

Pour ce qui est des tuiles Joker, l'implémentation a été beaucoup plus simple, voir nous a confortés dans certains de nos choix de modélisation. Faisant partie des fonctionnalités que nous avons ajoutées en dernier, nous n'avons cependant rencontré aucune difficulté particulière. Nous avons simplement rajouté un indice spécial négatif représentant les tuiles joker et dû modifier quelques règles. Le remplissage du sac et des lignes de motif ainsi que le calcule des points.

Affichage Textuel

Après avoir bien avancé dans le jeu, voyant la date de rendu du projet se rapprocher nous avons décidé de nous répartir les tâches de manière plus efficace. En effet, en se concentrant en même temps sur un même aspect de l'implémentation (bien que sur des fonctionnalités différentes), nous devions à chaque fois prendre le temps de comprendre chaque sous fonction ajoutée par l'autre, vérifier si nous créions des conflits, etc. Nous avons donc décidé que l'un d'entre nous s'occuperait de l'affichage textuel et des finalisations sur le jeu tandis que l'autre s'occuperait de l'interface graphique, plus longue à mettre en place, afin de mieux gérer le temps qui nous restait. En ce qui concerne

l'affichage textuel, il a été plus fastidieux que complexe à implémenter. L'idée était surtout de contrôler chaque donnée entrée par le joueur afin d'être certain d'appeler les fonctions dans le bon contexte.

Interface Graphique

Ici, l'une de nos premières décisions a été de favoriser le fonctionnel à l'esthétique. A savoir, d'abord s'efforcer d'avoir une interface graphique fonctionnelle et très basique visuellement avant d'éventuellement en embellir l'aspect. Une de nos premières difficultés a été de savoir comment récupérer les informations sur l'endroit qui a été cliqué. Ayant vu une application similaire en TP nous avons brièvement penser à utiliser les coordonnées x et y mais nous avons rapidement abandonné l'idée ; trop hasardeuse et fastidieuse ; pour une fonctionnalité des mouse event que nous avons découverte à l'occasion, permettant de récupérer les informations sur le JPanel qui a été cliqué. Les principaux problèmes que nous avons rencontrés par la suite ont été les suivants. Premièrement, un problème de rafraichissement de l'affichage sur lequel il nous a fallu nous pencher, certaines zones seulement ne rafraichissant leur affichage que lorsque la fenêtre était bougée. Également un bug que nous avons beaucoup de mal à régler totalement que j'aborderais plus en détail dans la partie 3

3- Difficultés en cours de route

La tuile premier joueur

Une erreur que nous avons commise en amont est de ne pas considérer davantage la particularité de la tuile premier joueur avant de commencer notre implémentation. Quand nous nous y sommes attelés, il est apparu que toutes nos classes et méthodes traitaient toutes les tuiles de la même manière ne les différenciant que pas leur couleur représentée par un entier. A ce stade et vu la structure de notre code, la solution la plus évidente, une classe héritant de tuile, aurait surtout été du rétropédalage. Nous avons donc eu l'idée suivante, différencier la tuile premier joueur également par son entier de couleur, -1. (Nous avons par la suite utilisé le même principe pour les tuiles joker). Ainsi quand il était question de traiter la tuile premier joueur uniquement par une condition lorsqu'il est nécessaire de la déplacer au centre du jeu ou dans un plancher.

Envoi de tuiles au plancher

Lors de la phase de test, nous nous sommes aperçus qu'avec les règles actuelles, un joueur pouvait se retrouver bloqué en ayant plus aucune ligne où il ait le droit de placer ses tuiles. En réexaminant la version disponible en ligne d'AZUL, nous nous sommes aperçus que dans de telles situations il est en effet possible d'envoyer des tuiles directement au plancher. Une première solution aurait alors été d'envoyer les tuiles au plancher dès qu'un joueur tente d'envoyer des tuiles vers une ligne de motif déjà pleine ou portant des tuiles d'une autre couleur. Cependant, un tel choix aurait alors interdit toute erreur de saisie plutôt que de d'abord avertir le joueur de l'impossibilité de son action. Nous avons donc opté pour la solution suivante. Ajouter un cas à part, représenté par l'indice de ligne -1, dans la fonction gérant le remplissage des lignes de motif, envoyant directement les tuiles sélectionnées au plancher.

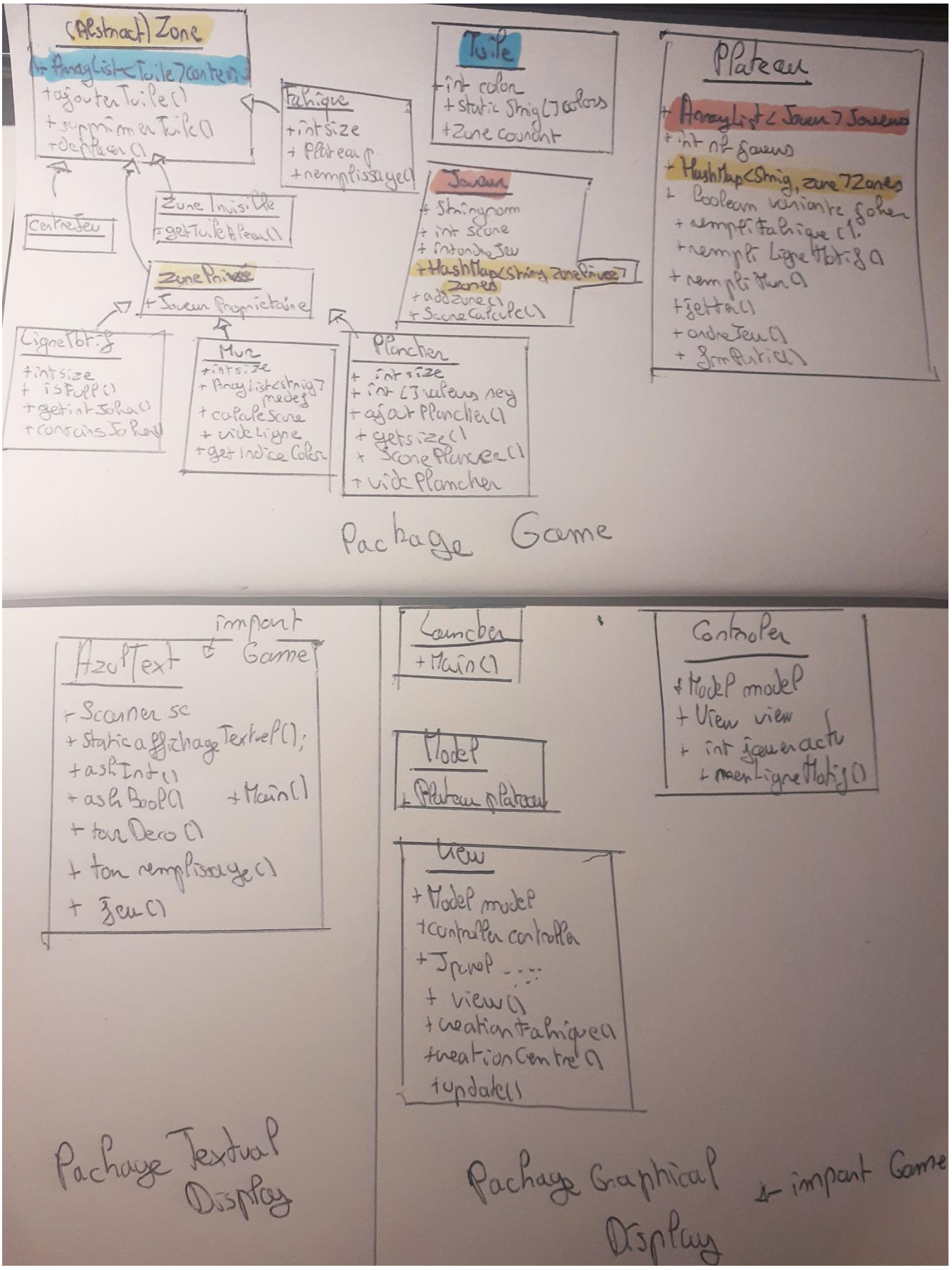
Le bug propre à l'Interface Graphique

Lors de la mise en place de l'IG, nous avons rencontré un bug que nous avons beaucoup de mal à régler entièrement. Parfois, et sans réelles conditions identifiables, lorsque l'on tente de prendre des tuiles au centre du plateau, une exception ArrayOutOfBound est soulevée et l'action n'a pas lieu. Après plusieurs tests il est apparu que le problème était le suivant. Dans certaines conditions très floues, après avoir portant effectué correctement le début de la fonction, la fonction remplissant les lignes de motif considère que le centre est vide, ce qui ne pourrait jamais être le cas si on arrive à ce stade de la fonction. Si en tâtonnant nous avons réussi à supprimer certains cas ou le bug se présentait nous n'en avons pas vraiment compris la nature et avons eu beaucoup de mal à identifier les conditions où il se produisait, car toujours très différentes les unes des autres. D'autant plus que ce bug n'a jamais lieu dans l'interface textuelle.

NB : Malgré des essais de correction jusqu'au dernier jour du délais imparti, il est possible que ce bug soit en partie toujours présent et nous en sommes conscient.

Conclusion

Nous sommes plutôt satisfaits du résultat de notre travail, s'il n'a pas été exempt d'erreurs et de fautes de jugements lors de la conception qui ont pu nous poser problèmes par la suite, et que le rendu final n'est pas aussi complet que nous le voulions, nous estimons être arrivés à quelque chose de fonctionnelle, implémentant deux variantes et avec un jeu entièrement fonctionnel en affichage textuel ainsi qu'une interface graphique lisible bien que non entièrement complétée faute de temps.
Nous en avons également tiré des enseignements sur la gestion et l'organisation d'un travail en groupe et dans un temps limité, la répartition des tâches et l'important d'une modélisation pensée bien à l'avance.


(Abstract) Zone
+ ArrayList<Tuile> contenu
+ ajouterTuile()
+ supprimerTuile()
+ getPlateau()
Fabrique
+ int size
+ Plateau p
+ remplissage()
Tuile
+ int color
+ static String[] colors
+ Zone courant
Plateau
+ ArrayList<Joueur> Joueurs
+ int nb_joueurs
+ HashMap<String, zone> Zones
+ Boolean variante Solen
+ remplirFabrique()
+ remplirLigneMotif()
+ remplirMur()
+ getter()
+ ordreJeu()
+ getPlateau()
Centre Jeu
Zone Invisible
+ getTuileAPiocher()
Joueur
+ String nom
+ int score
+ int ordreJeu
+ HashMap<String, zonePrivee> zones
+ addZone()
+ scoreCalcule()
Zone Privee
+ Joueur proprietaire
LigneMotif
+ int size
+ isFull()
+ getInfoJoueur()
+ contientJoueur()
Mur
+ int size
+ ArrayList<String> modeles
+ calculeScore
+ videLigne
+ getIndiceColor
Plancher
+ int size
+ int[] valeurs neg
+ ajoutPlancher()
+ getSize()
+ scorePlancher()
+ videPlancher
Package Game
import Game
AzulText
+ Scanner sc
+ Static affichageTextuel();
+ askInt()
+ askBool()
+ tourDeco()
+ tourRemplissage()
+ jeu()
+ Main()
Launcher
+ Main()
Model
+ Plateau plateau
View
+ Model model
+ controller controller
+ JPanel ...
+ view()
+ creationFabrique()
+ creationCentre()
+ update()
Controler
+ Model model
+ View view
+ int joueurActu
+ menLigneMotif()
Package Textual Display
Package Graphical Display
import Game