POLYTECH
CLERMONT-FERRAND

Année Scolaire : 2021/2022
GMM5A

Rapport de projet

« Tchou Tchou »

Projet réalisé par : Lafon Gabin, Sendra Thomas, Limousin Lucas

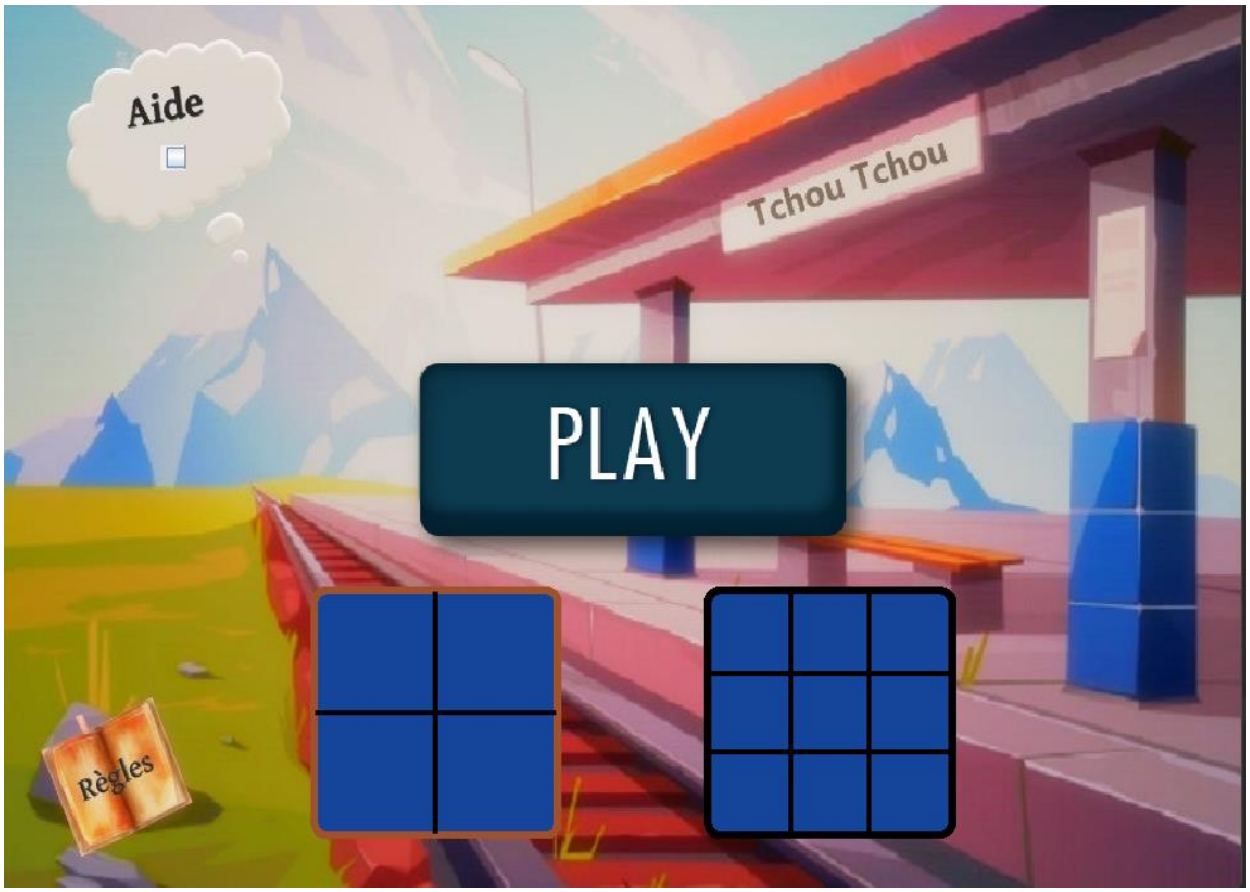
Table des matières

1 / Présentation du jeu :

Tchou Tchou est un jeu de plateau dont l'objectif est de reconstituer un chemin détruit permettant au train de se déplacer de l'entrée à la sortie du plateau modélisées par des flèches.

Ce jeu propose deux niveaux de difficulté correspondant à la taille du plateau qui est composé de 4 rails (2x2) pour le niveau débutant et de 9 rails (3x3) pour le niveau avancé.

Afin de créer le bon chemin, l'utilisateur doit déplacer les rails du plateau. Une fois qu'il estime avoir trouvé une solution possible, il actionne le feu afin de lancer le train. En outre, une option d'aide dévoilant une ou plusieurs cases d'un chemin correct a été mise à disposition de l'utilisateur.



2/ Partie Analyse et Conception du logiciel

Dans cette partie, nous allons présenter les différents diagrammes ayant servi à modéliser notre jeu.

a) Le diagramme de cas d'utilisation

Nous avons choisi de débuter par les cas d'utilisation (Fig.1) afin de cibler les principaux axes à approfondir en accord avec les exigences clients. Ce diagramme permet ainsi de modéliser les actions possibles pour l'enfant lorsqu'il est en train de jouer.

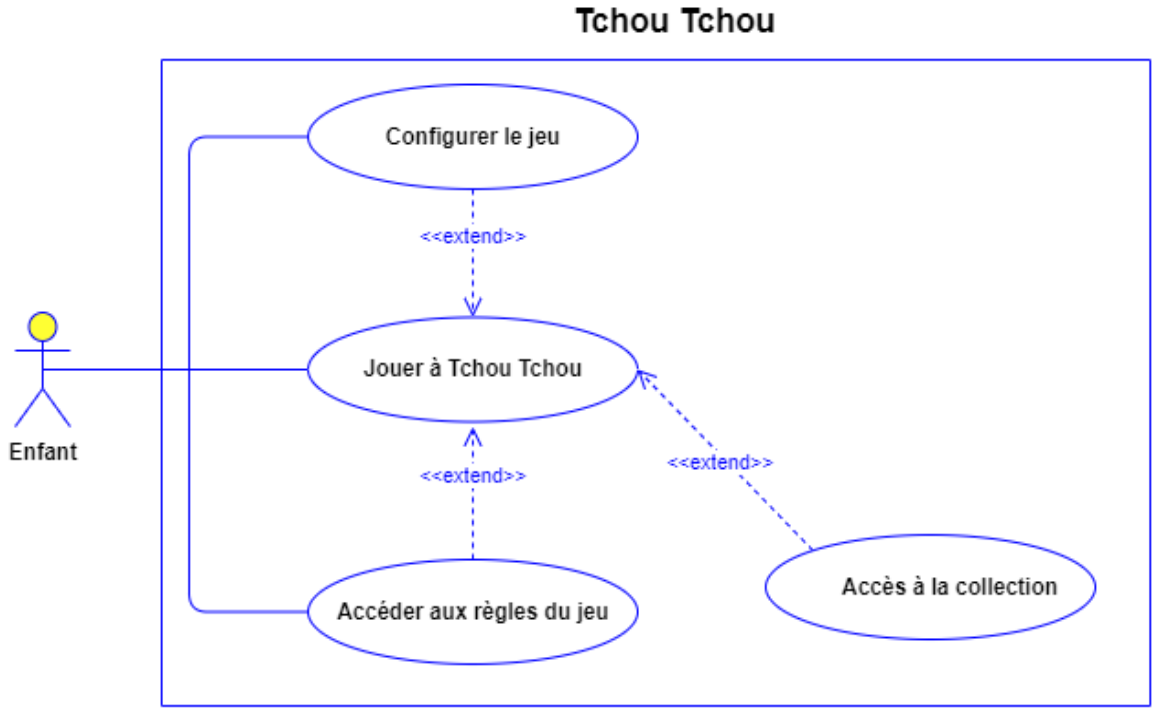


Figure 1 : Cas d'utilisation

Nous avons recensé 4 cas d'utilisation :

- Jouer une partie :
    1. Un menu d'accueil est affiché par le système
    2. L'enfant appuie sur le bouton "Play"
    3. La partie se lance
    4. La partie se déroule
    5. La partie se termine

- Configurer le jeu :
    1. Le joueur a la possibilité de choisir la difficulté du jeu (2x2 ou 3x3)
    2. Le joueur a la possibilité d'activer ou de désactiver l'aide depuis le menu d'accueil

- Accéder aux règles du jeu :
    1. L'utilisateur appuie sur l'icône du livre à partir du menu d'accueil
    2. Le système affiche les règles dans une nouvelle fenêtre

- Accéder à la collection :
    1. Le joueur peut visualiser la collection de trains et choisir un des trains qu'il a débloqués pour jouer avec (1 train débloqué par victoire)

b) Diagramme de séquences

Ce deuxième diagramme (Fig. 2) permet de mieux comprendre l'interaction entre l'enfant et l'interface de jeu représentée par Tchou Tchou. La notion de temporalité offerte par ce diagramme affine la compréhension de l'enchainement des échanges ayant lieu entre ces deux acteurs.

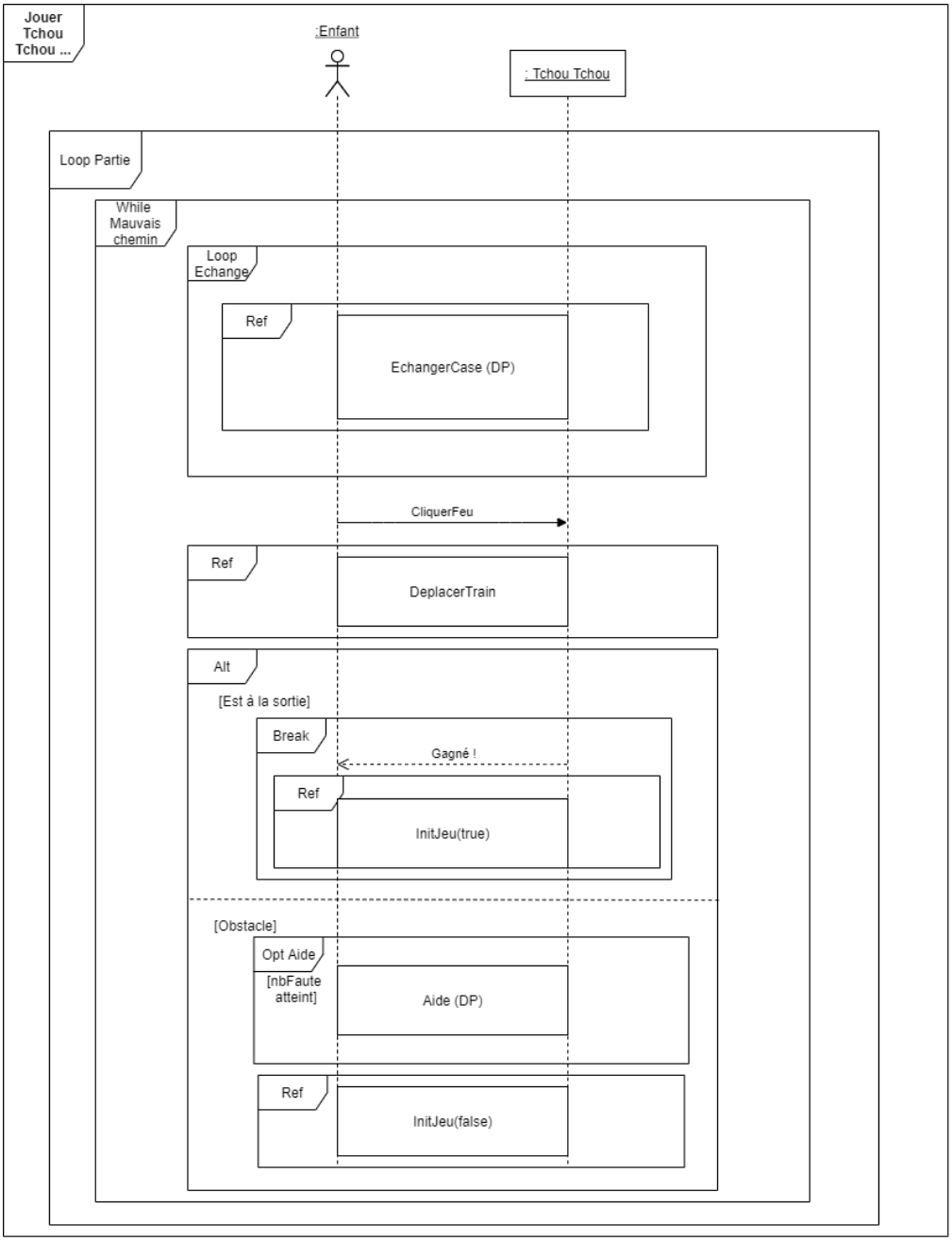


Figure 2 : Diagramme de séquences système

Ce diagramme comporte beaucoup de références vers d'autres diagrammes plus spécifiques afin que celui-ci ne soit pas trop chargé.

Lors du début d'une partie, l'enfant se retrouve devant un plateau mélangé. Ainsi, celui-ci peut, tant qu'il le souhaite, effectuer des échanges de cases (Fig.3) jusqu'à ce qu'il arrive à un chemin qui lui convienne.

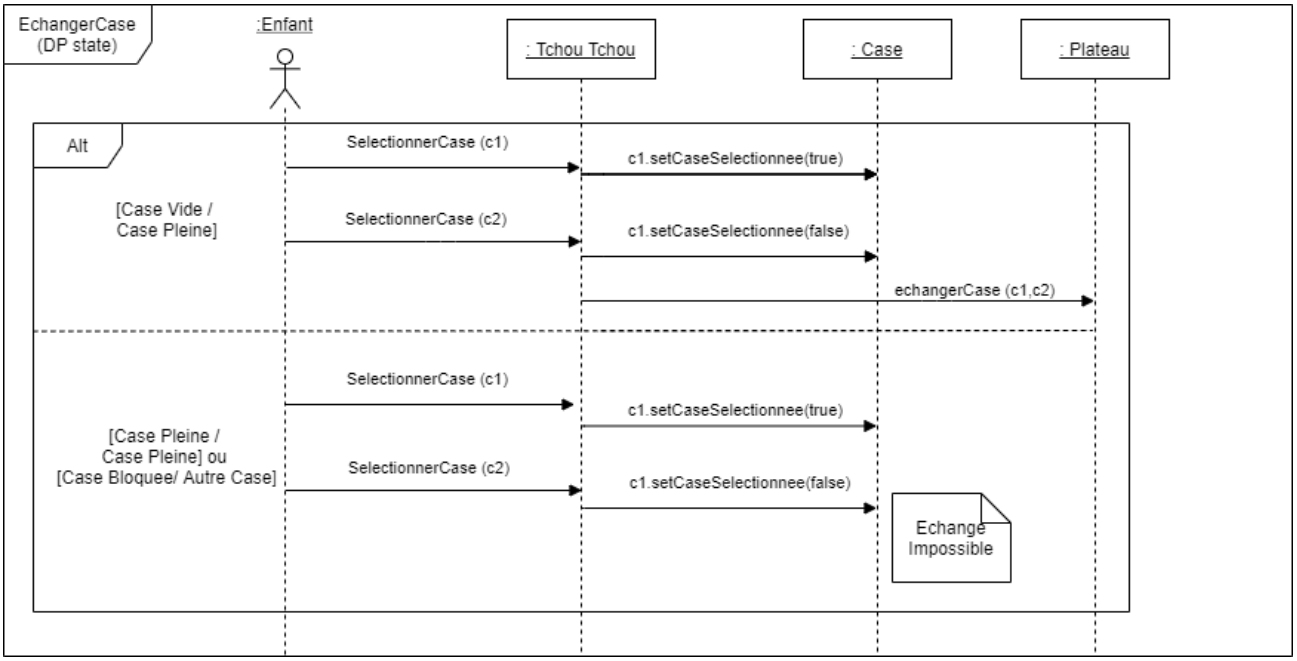


Figure 3 : Diagramme de séquences du design pattern EchangerCase

Lors de l'échange entre deux cases, il y a deux cas possibles :

- si l'une des deux cases est vide alors l'échange est réalisé
- si aucune des cases n'est vide alors l'échange est impossible

Le message setCaseSelectionne permet de mettre en valeur la première des deux cases sélectionnées afin que l'enfant ne se perde pas dans ses échanges. Et ce schéma peut être répété tant que l'enfant souhaite déplacer des cases.

D’après le diagramme de séquences système (Fig.2), l’enfant doit ensuite cliquer sur le feu. Cette action provoquera le déplacement du train qui est illustré dans le diagramme ci-dessous :

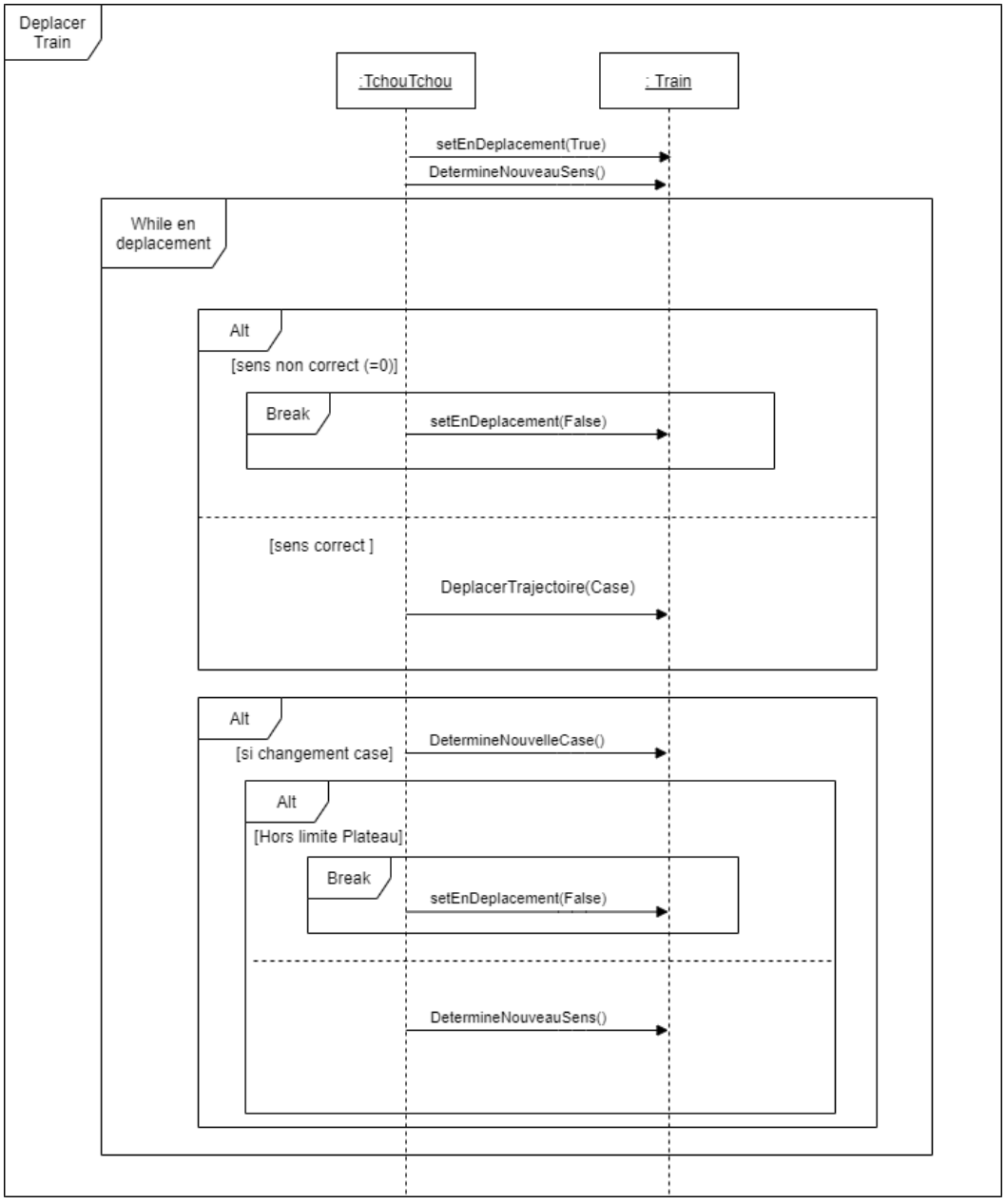


Figure 4 : Diagramme de séquences du design pattern DeplacerTrain

Pour commencer, un message est envoyé au train afin que celui-ci entame son déplacement. Pour la 1ère case, son sens est calculé en fonction de la position initiale du train dans celle-ci (si sens = 1 alors le déplacement se fait à droite, si sens = 2 alors le déplacement se fait à gauche et si sens = 0 alors le déplacement est impossible).

Tant qu'il est en déplacement dans la même case alors il y a deux cas possibles :

- Si le sens est correct (sens = 1 ou sens = 2), le train se déplace d'un pas suivant une trajectoire calculée à l'aide des formules trigonométriques adéquates.
- Si le sens vaut 0, le train s'arrête puisque le rail précedent ne coincide pas avec le rail suivant.

Lorsque le train change de case, on regarde si le train ne se trouve pas hors du plateau :

- Si c'est le cas, alors comme auparavant le train s'arrête.
- Dans le cas contraire, on va calculer le sens de la nouvelle case.

Une fois que le train a fini son déplacement, on va de nouveau se trouver devant deux cas possibles :

- L'enfant a gagné : un message de victoire lui sera envoyé suivi de l'initialisation du jeu à l'état true (Fig.5).
- L'enfant a perdu : si l'option d'aide avait été cochée et que ses conditions sont respectées, celle-ci est appliquée (Fig.6). Suite à cela, le jeu se réinitialisera à l'état false (Fig.5).

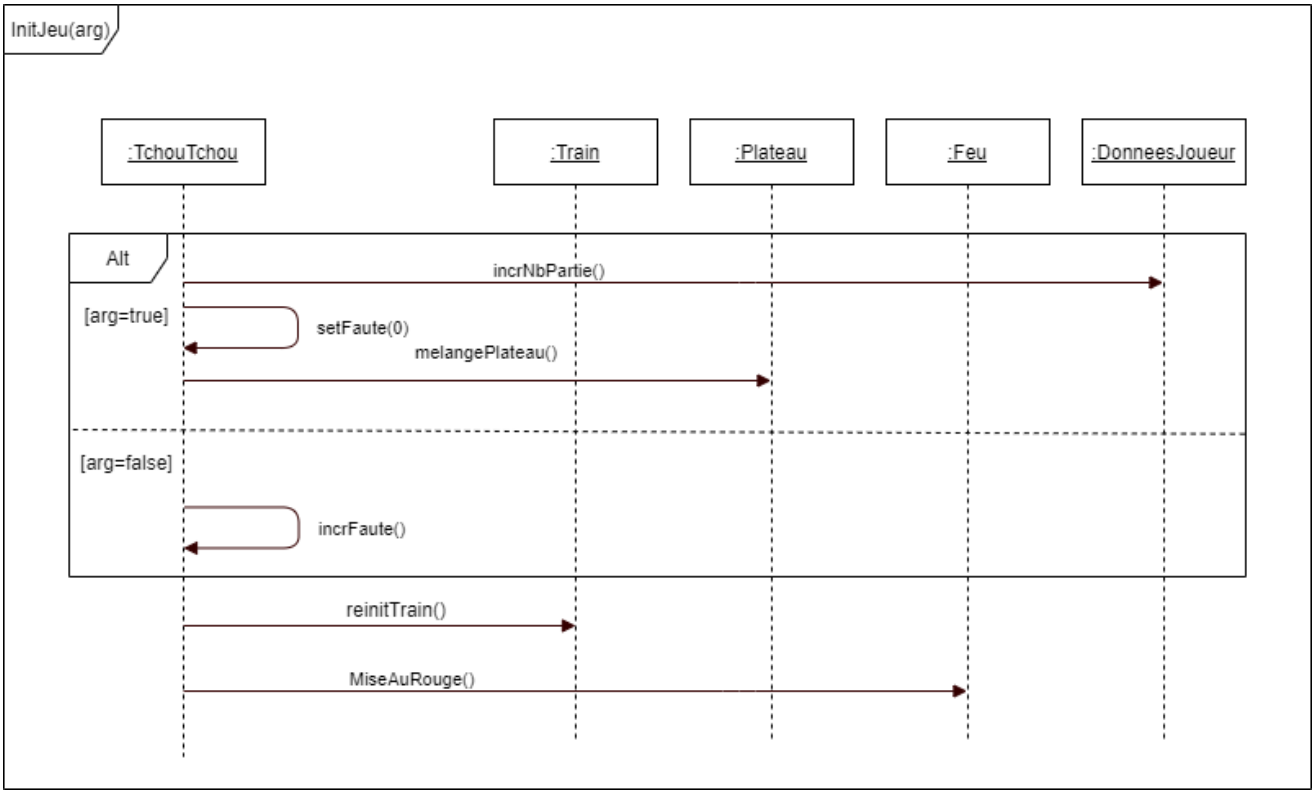


Figure 5 : Diagramme de séquences du design pattern InitJeu

Si l'initialisation du jeu est à l'état true (victoire de l'enfant), alors on incrémente le nombre de parties gagnées stocké dans les données du joueur en cours. Ensuite, on remet le compteur de fautes à zéro pour préparer la prochaine partie et enfin on va mélanger le plateau.

Au contraire, dans le cas de l'état false (échec de l'enfant), seule l'incrémentation du nombre de fautes est opérée.

Dans tous les cas, InitJeu se termine par la réinitialisation du train au point de départ ainsi que la mise au rouge du feu.

Pour terminer l'explication détaillée du diagramme de séquences système (Fig.2), nous allons aborder le système d'aide (Fig.6) que nous avons rajouté. Cette aide ne s'applique que lorsque le nombre de fautes est atteint : une aide est proposée au bout de 3 erreurs dans la limite de 2 aides pour le 2x2 et de 3 aides pour le 3x3.

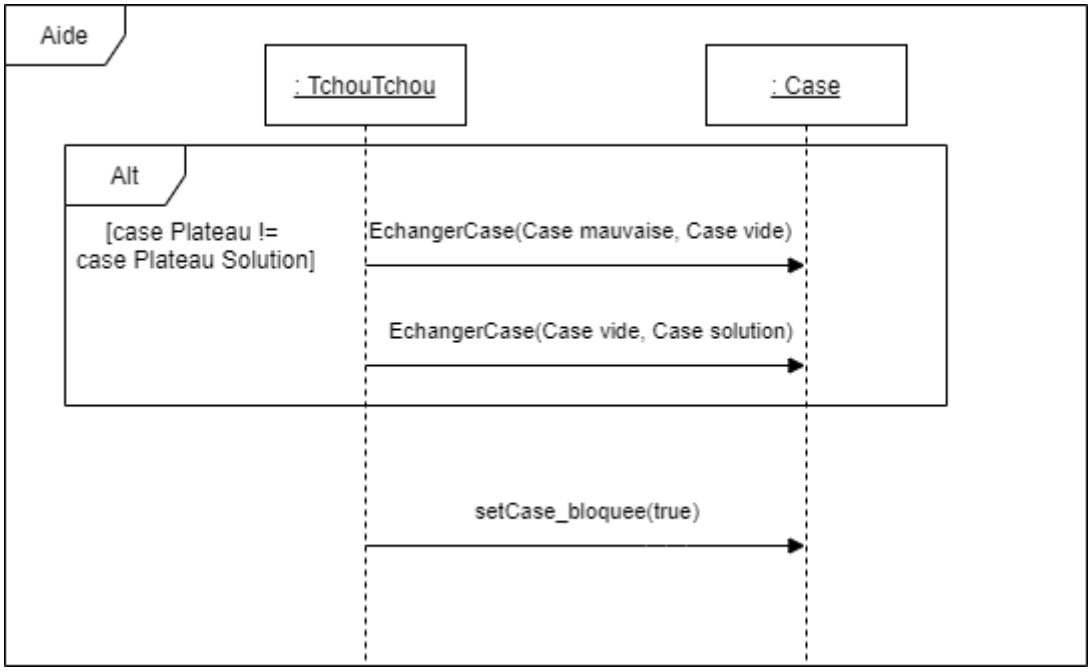


Figure 6 : Diagramme de séquences du design pattern Aide

Suivant le plateau de solution proposé, l'aide affichera la première case qui complète le chemin d'aide actuel (pour la 1ère aide, cette case sera forcément la 1ère du plateau). Si la case solution n'est pas à sa place, nous l'échangeons pour la mettre au bon endroit.

Cependant, comme nous avons imposé un échange entre deux cases uniquement si l'une des deux est vide, nous devons décomposer cet échange en 2 étapes.

Une fois les échanges terminés, cette case est bloquée c'est-à-dire qu'on ne pourra plus l'échanger avec une autre case. Cette opération est matérialisée par le changement de couleur de la case solution.

Tout le processus présenté ci-dessus est reproduit tant que l'enfant n'a pas trouvé la solution. Une fois trouvée, comme le présente le diagramme InitJeu (Fig.5), une nouvelle partie s'enchaine et cela se répétera tant que l'enfant souhaitera jouer.

c) Diagramme de classes

i) Analyse

Après vous avoir présenté les différents diagrammes de séquences, nous allons aborder le diagramme de classes en commençant par celui d'analyse. Nous avons choisi ce diagramme car celui-ci reflète au mieux notre modèle dans sa globalité tout en posant un regard plus précis sur les interactions entre les différentes classes que nous avons créées.

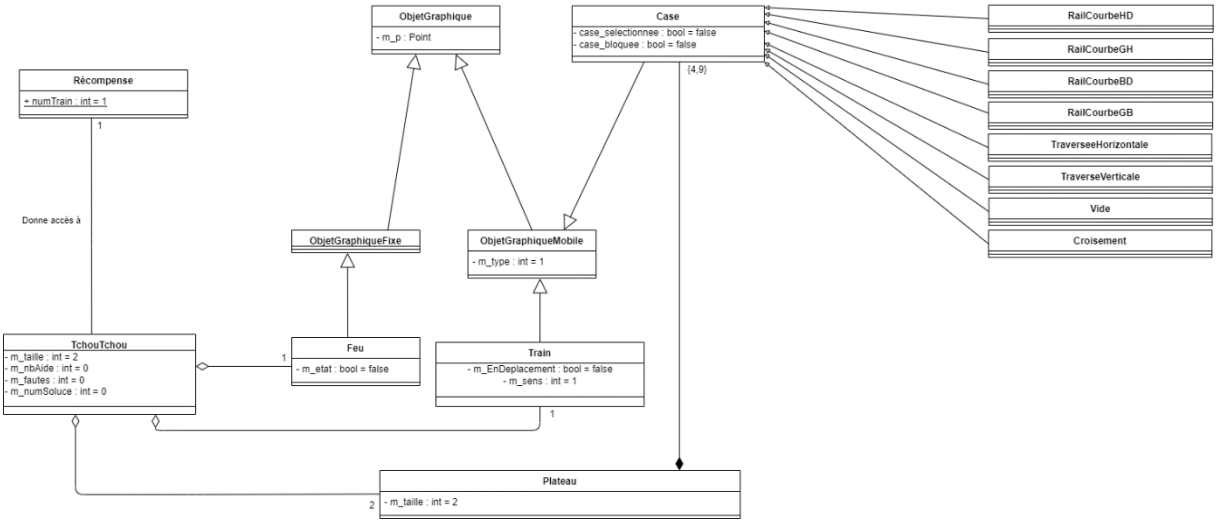


Figure 7 : Diagramme de classe d'analyse

Pour notre diagramme de classes, nous sommes partis de la classe ObjetGraphique. Cette classe est la classe mère de nombreux concepts de notre jeu. Un ObjetGraphique ne sera défini que par un point en 2D.

De cette classe, nous avons choisi de créer deux sous-classes distinctes : ObjetGraphiqueMobile et ObjetGraphiqueFixe. Etant donné que dans notre jeu, certains objets seront amenés à être déplacer par l'enfant tandis que d'autres ne bougeront jamais, nous avons jugé pertinent de distinguer ces deux catégories d'objet.

Un objet graphique fixe n'aura pas d'autre attribut supplémentaire et nous permet uniquement de conserver une certaine cohérence.

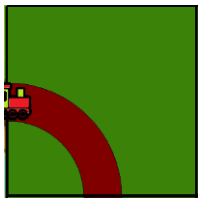
Un objet graphique mobile est, quant à lui, caractérisé par un type (en plus du point hérité de la classe mère) nous servant à distinguer nos objets graphiques mobiles.

Le seul objet graphique fixe que nous avons est le feu d'où l'absence de l'attribut type dans ObjetGraphiqueFixe. Ce feu possède seulement un attribut état correspondant au fait qu'il soit rouge (false) ou vert (true).

En ce qui concerne les objets graphiques mobiles, nous avons deux sous catégories :

- Il y a le train qui sera amené à se déplacer dans le plateau et dont le type sera 1. Il possède deux attributs supplémentaires qui sont le sens (cf Fig.4 DeplacerTrain) et un booléen EnDeplacement permettant de récupérer l'information relative au déplacement du train : true s'il est en train de se déplacer, false sinon.
- De l'autre côté, nous avons les cases qui composent notre plateau. L'attribut case_selectionnee permet la mise en valeur de la case lors de l'échange (cf Fig.3 EchangerCase) et case_bloquee permet d'empêcher des futurs échanges avec une autre case (cf Fig.6 Aide).

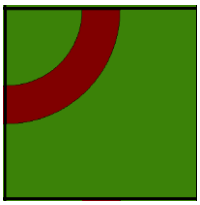
<u>Case va être la classe mère de l'ensemble des rails que l'enfant a à sa disposition :</u>



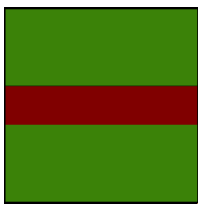
Rail GaucheBas

Rail BasDroite

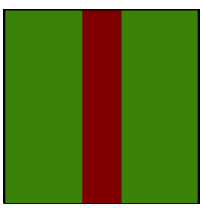
Rail HautDroite



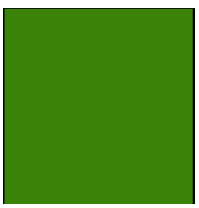
Rail GaucheHaut



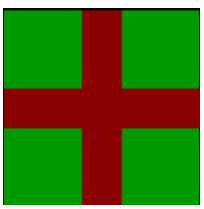
Traversée Horizontale



Traversée Verticale



Rail Vide



Croisement

Nous avons la classe Plateau qui va être composée des cases précédentes dont le seul attribut est la taille et qui indiquera le nombre de cases dans le plateau (taille 2 : 4 cases, taille 3 : 9 cases).

Pour rassembler tous les concepts vus précédemment, nous avons créé la classe Tchou Tchou. Celle-ci représente l'instance d'une partie pour l'enfant.

Cette classe possède plusieurs attributs :

- m_taille : la taille du plateau voulue pour la partie
- m_fautes : le compteur du nombre de fautes commises par l'enfant
- m_nbAide : le compteur du nombre d'aides déjà fournies qui sert à voir si le nombre d'aide maximal a été atteint pour l'enfant
- m_num_Soluce : le numéro d'un des plateaux solutions proposés au joueur
- m_feu : le feu associé à la partie
- m_plateau : le plateau associé à la partie
- m_plateau_soluce : le plateau solution servant pour le système d'aide
- m_train : le train associé à la partie

Il ne reste maintenant que la classe Récompense qui sera liée à la fois à Tchou Tchou et à la fois aux données du joueur. L'attribut numTrain représente le nombre de trains débloqués par l'enfant lors de l'ensemble de ses parties précédentes et sera incrémenté de 1 lors d'une nouvelle victoire.

ii) Conception

Après avoir présenté le diagramme de classes d'analyse et déjà abordé la structure du projet de façon globale, nous allons détailler un peu plus précisément ces classes par le biais du diagramme de classes de conception.

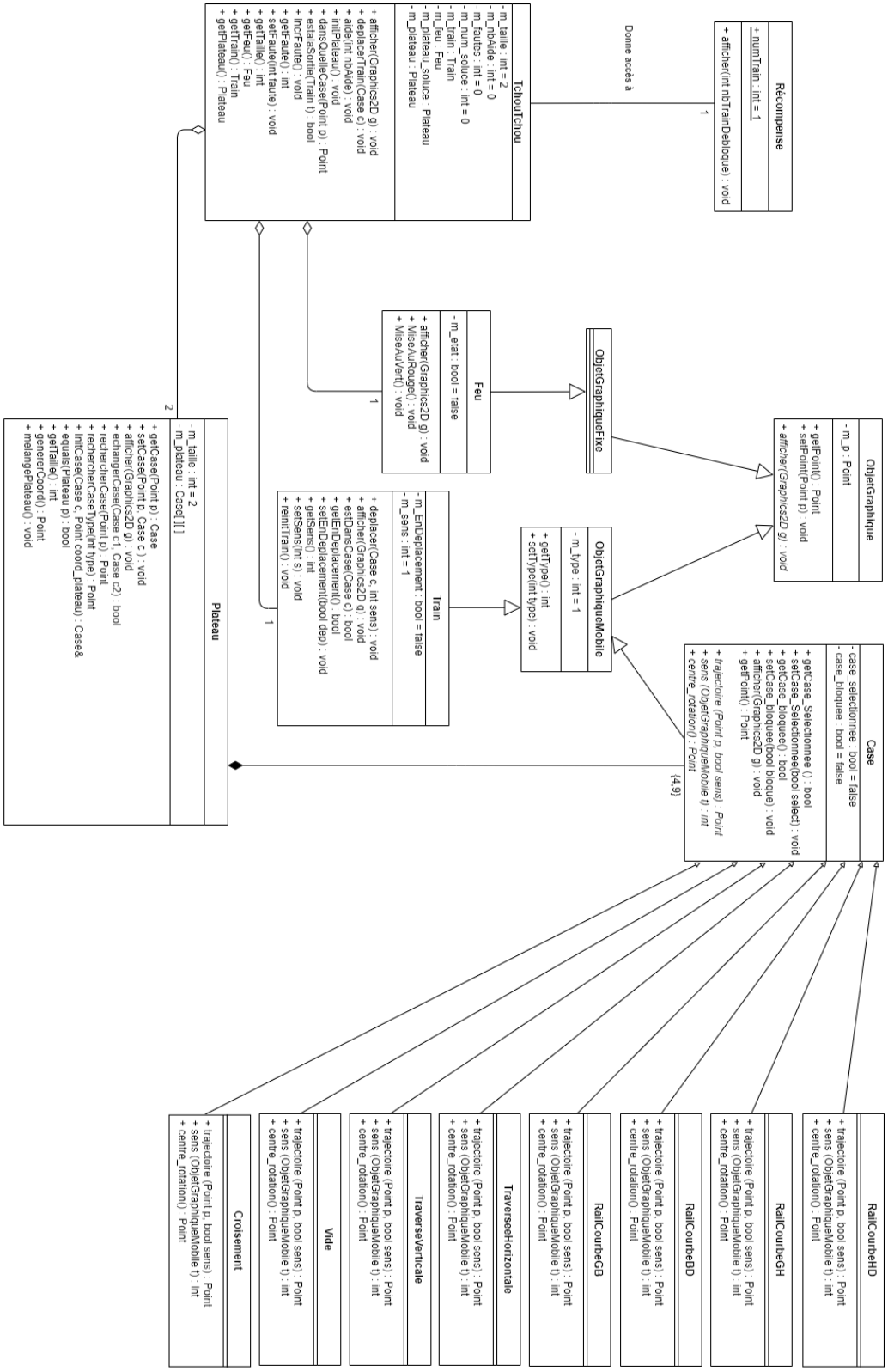


Figure 8 : Diagramme de classes de conception

Lors des explications des différentes méthodes de ce diagramme de classes, nous ne nous pencherons pas sur l'explication des accesseurs et mutateurs ainsi que sur certaines méthodes que nous jugeons suffisamment explicites.

Dans la classe ObjetGraphique, ce qui était important était de pouvoir afficher ces objets, que ce soit le train, une case ou le feu. C'est pourquoi la méthode **afficher** est abstraite rendant ainsi la classe ObjetGraphique abstraite.

Dans la classe Case, nous redéfinissons la méthode **afficher** issue d'ObjetGraphique. Apparaissent également trois autres méthodes abstraites :

- **Trajectoire** : méthode prenant un point en 2D avec un sens (comme nous l'avons vu précédemment) afin de calculer la trajectoire suivie par le point à la prochaine itération. Cette méthode est abstraite puisque la trajectoire n'est pas la même suivant le rail auquel elle est appliquée.
- **Sens** : méthode qui à partir d'un ObjetGraphiqueMobile (le train en l'occurrence) permet de savoir s'il peut se déplacer et si oui dans quel sens il doit aller pour ne pas revenir sur ses pas.
- **CentreRotation** : méthode servant exclusivement aux rails courbes permettant de connaitre l'emplacement du centre de rotation et servant notamment pour le calcul de la trajectoire.

Toutes ces méthodes sont redéfinies dans les différentes cases suivant leur spécificité.

Concernant le plateau, nous avons implémenté la méthode **EchangerCase** qui, comme son nom l'indique, permet d'échanger les deux cases en argument (cf Fig. 3 EchangerCase). La méthode **RechercherCase** permet de localiser les coordonnées dans le plateau du point passé en argument tandis que **RechercherCaseType** permet de localiser l'emplacement dans le plateau de la case du même type que celui passé en argument.

L'initialisation d'une case dans le plateau se fait par l'intermédiaire de la méthode **InitCase**.

Lors de l'implémentation de la classe Train, l'enjeu principal était de le déplacer d'où la méthode du même nom permettant, suivant la case où il se situe et le sens, d'avancer d'une itération. La méthode **estDansCase** permet de savoir si le train est dans la case passée en argument ou non. Enfin, nous avons **ReinitTrain** qui va se charger de remettre le train dans sa position de départ. Cette méthode est utilisée lors de l'appel de la méthode InitJeu (cf Fig.5 InitJeu).

Quant à la classe Tchou Tchou, l'une des méthodes les plus importantes est **aide** qui permet la mise en place de l'aide lorsque celle-ci est activée (cf Fig. 6 Aide).

De surcroît, la méthode **dansQuelleCase** renvoie les coordonnées du plateau où le point passé en argument est situé.

3/ Partie programmation du logiciel

a) L'architecture du logiciel

Figure 9 : Diagramme de classes des fenêtres graphiques

La fenêtre graphique MainWindow nous permet de gérer l'ensemble de notre menu d'accueil.

Le déroulement d'une partie est assurée grâce à la fenêtre graphique Jeu.

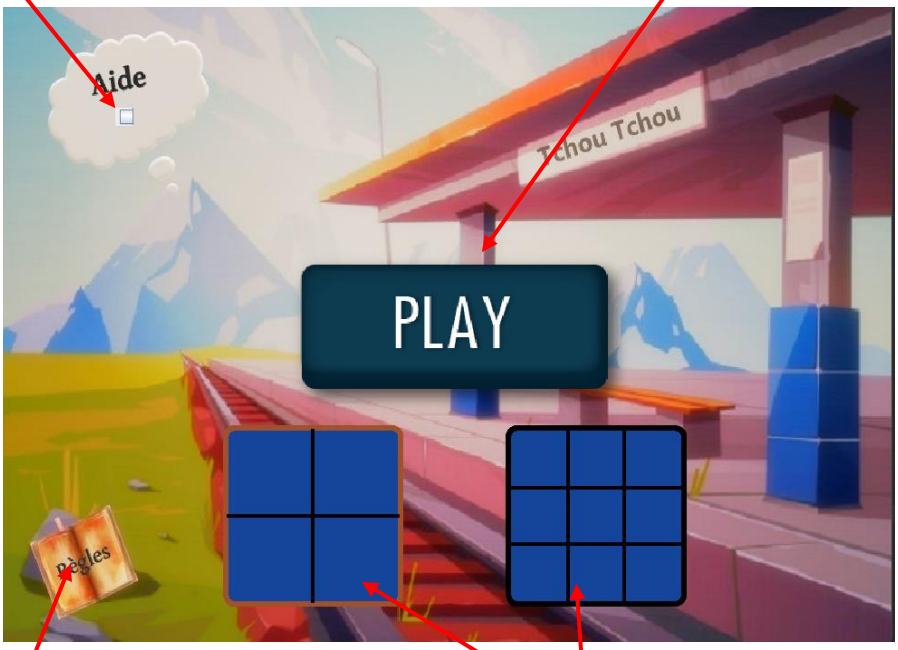
L'affichage des règles de notre jeu se fait par l'intermédiaire de la fenêtre graphique Regle.

Enfin, la fenêtre graphique FenetreRecompense est utilisée pour afficher la collection de trains du joueur et peut être ouverte depuis la fenêtre Jeu.

b) Présentation des écrans

En cliquant ici : on active ou désactive l'aide et c'est le slot **actionPerformed()** qui est appelé.

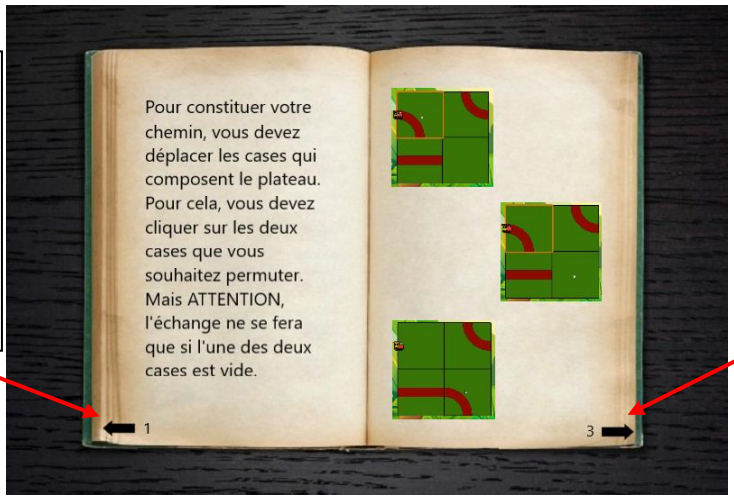
En cliquant ici : la partie se lance selon le niveau choisi.



En cliquant ici : la méthode **formMouseClicked()**, qui récupère le clic, ouvre le livre de règles et la fenêtre règle s'ouvre.

En cliquant ici : on choisit le niveau de notre jeu. Le clic est interprété par la méthode **formMouseClicked()** et la valeur de l'attribut niveau est modifiée pour correspondre à la sélection.

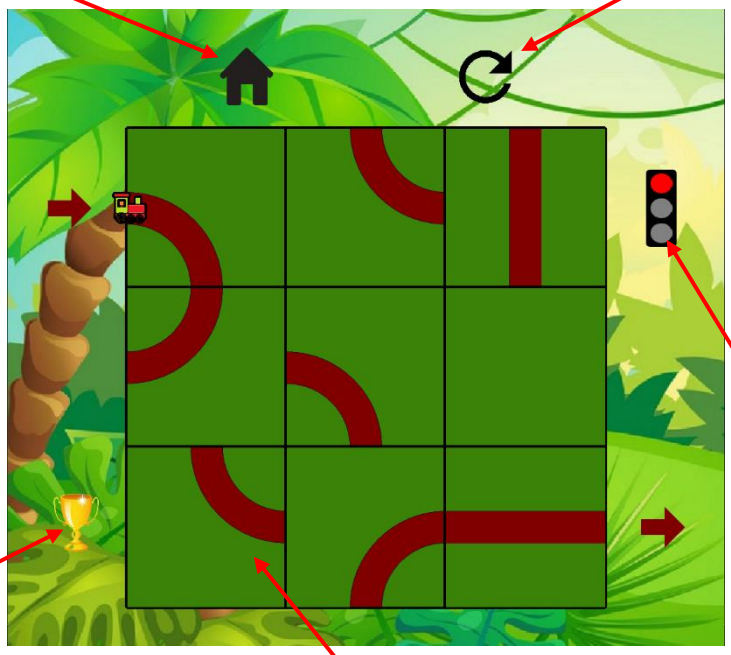
En cliquant ici : le clic est interprété par la méthode **formMouseClicked()** et le numéro de page diminue en utilisant **decrPage()**.



En cliquant ici : le clic est interprété par la méthode **formMouseClicked()** et le numéro de page augmente en utilisant **incrPage()**.

En cliquant ici : la méthode **formMouseClicked()** récupère le clic, provoque la fermeture de cette fenêtre.

En cliquant ici : la méthode **formMouseClicked()** récupère le clic et provoque la génération d'un nouveau plateau.



En cliquant ici : la méthode **formMouseClicked()** récupère le clic et ouvre la fenêtre FenêtreRecompense.

En cliquant ici : la méthode **formMouseClicked()** récupère le clic et appelle la méthode **cliquerFeu**().

En cliquant ici : la méthode **formMouseClicked()** récupère le clic et sélectionne une case en vue de l'échanger.

En cliquant ici : le **formMouseClicked()** permet d'interpréter le choix de l'enfant concernant le train qu'il souhaite voir apparaitre à l'écran.

4/ Conclusion

Le changement de langage avec le passage de Qt à Java nous a permis de mettre en place une nouvelle fonctionnalité avec l'ajout du son qui était quelque chose auquel nous avions pensé l'an dernier mais que nous n'avions pas pu mettre en place sous Qt.

Le seul petit bémol que nous pouvons soulever est le fait que le changement du train et du fond qui lui est associé ne se fait pas automatiquement lors de la fermeture de la fenêtre Récompense (contrairement à Qt). En effet, ce problème est du au fait que le code continue à être exécuté lorsque la fenêtre Récompense est ouverte. C'est pourquoi les changements de train et de fond ne se font qu'après un nouveau clic sur la fenêtre principale.

Nous estimons que notre projet est abouti. L'intégralité des exigences du client nous paraissent respectées. De plus, toutes nos idées ont pu être implémentées comme nous l'imaginions à l'origine.

Le côté ludique du projet, qui s'adresse aux enfants en bas âge, laissait libre cours à notre imagination et permettait de concevoir un jeu complet et divertissant.