Modélisation du jeu Labyrinthe

Daniel Le Berre

Février 2021

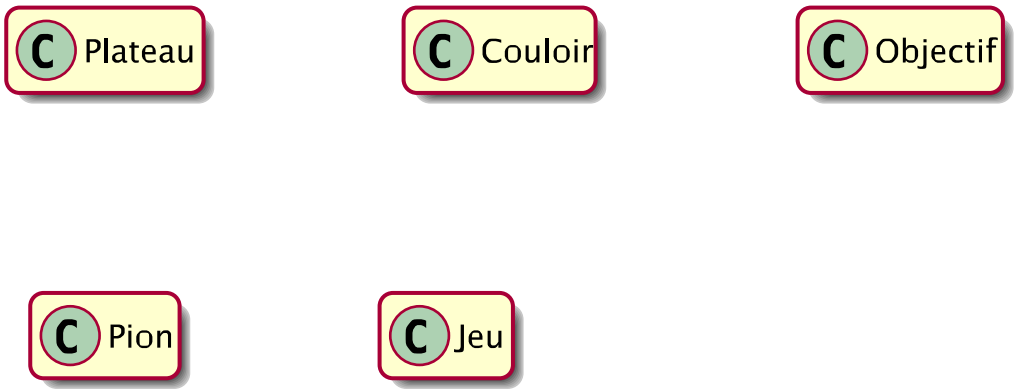
On modélise ici le jeu labyrinthe d'un point de vue objet. Le public visé correspond à des L3 informatique.

But du jeu : Dans un labyrinthe enchanté, les joueurs partent à la chasse aux objets et aux créatures magiques. Chacun cherche à se frayer un chemin jusqu'à eux en faisant coulisser astucieusement les couloirs. Le premier joueur à découvrir tous ses secrets et à revenir à son point de départ remporte cette passionnante chasse aux trésors.

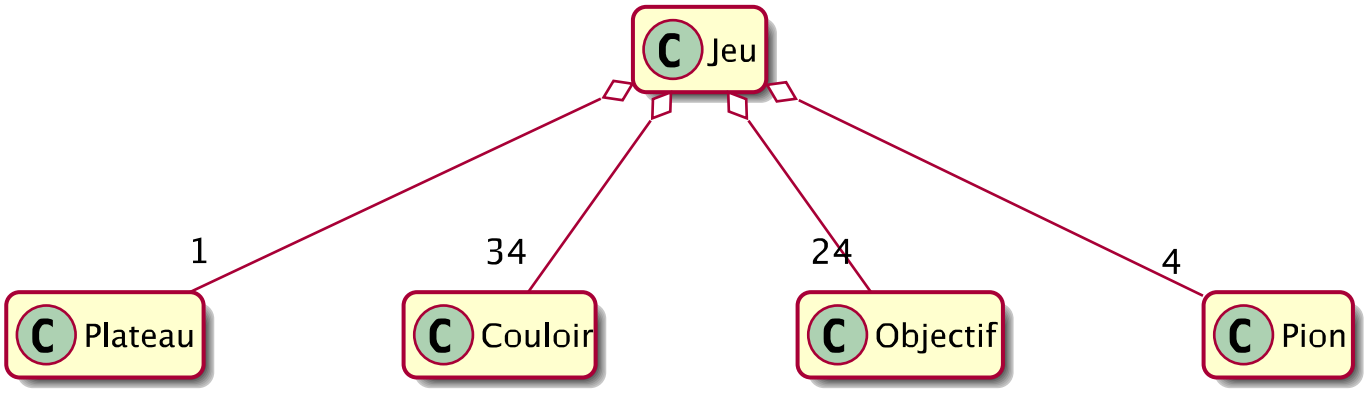
Contenu : 1 plateau de jeu, 34 plaques Couloir, 24 cartes Objectif, 4 pions.

Première étape : retrouver les classes principales

L'une des façons simples de retrouver des classes est d'utiliser une classe pour représenter chaque élément physique du jeu, c'est à dire le `Plateau`, les plaques `Couloir`, les cartes `Objectif`, les `Pions`, et bien sûr le `Jeu` lui même.

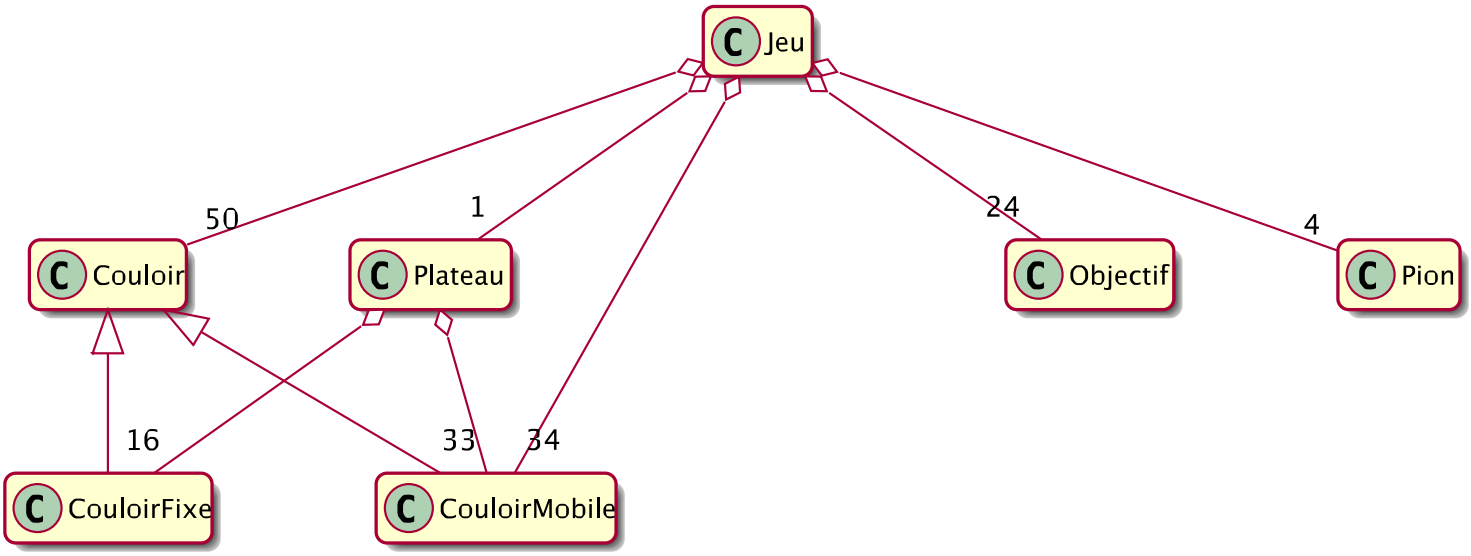


On utilise maintenant les relations entre classes et les multiplicités pour expliciter les dépendances entre classes.

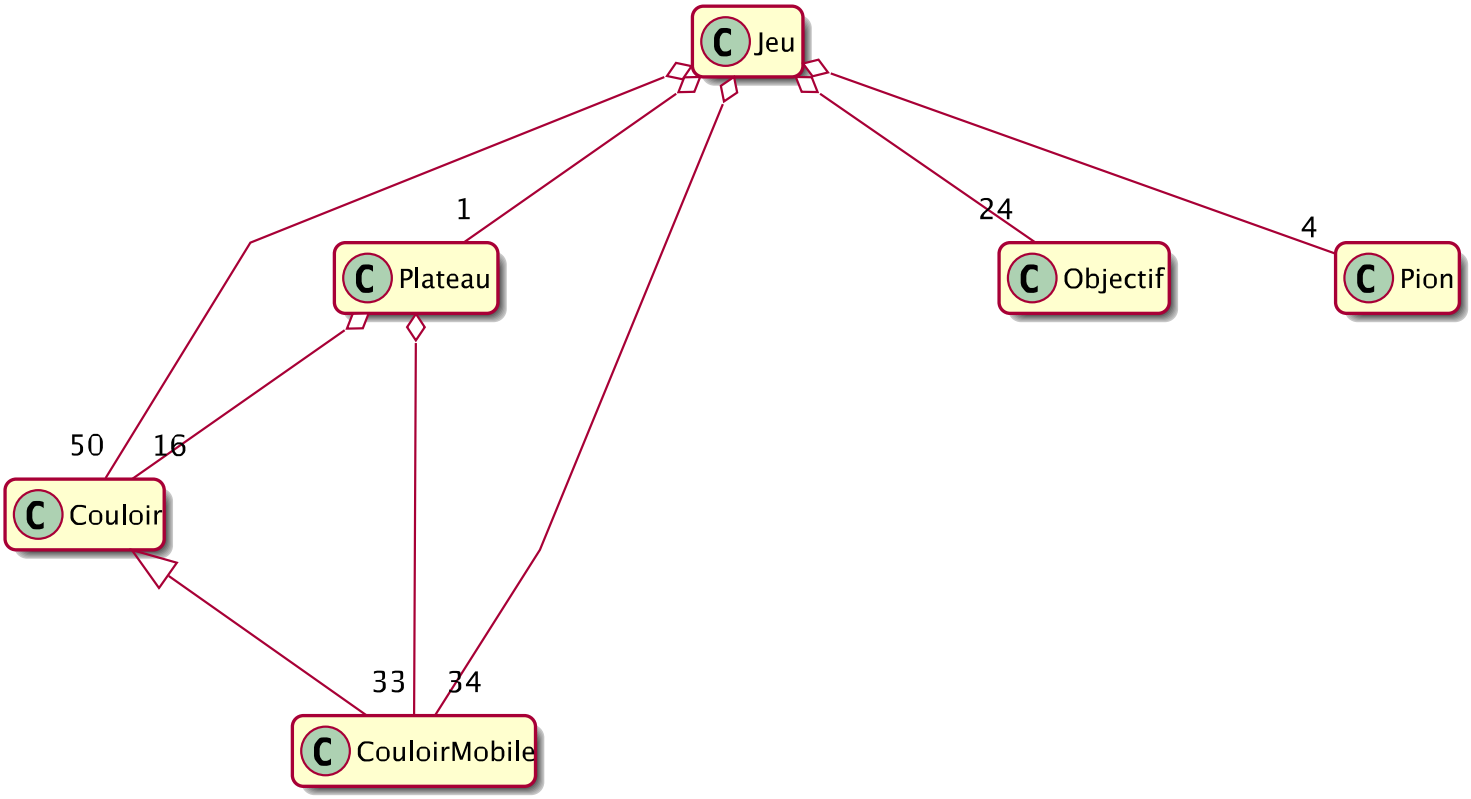


Si on regarde bien le plateau de jeu, on peut noter qu'il contient 16 couloirs fixes. Le plateau contient 49 cases. 16 couloirs fixes + 34 couloirs mobiles correspondent à 50 couloirs pour 49 cases. Il y a toujours un couloir libre durant le jeu.

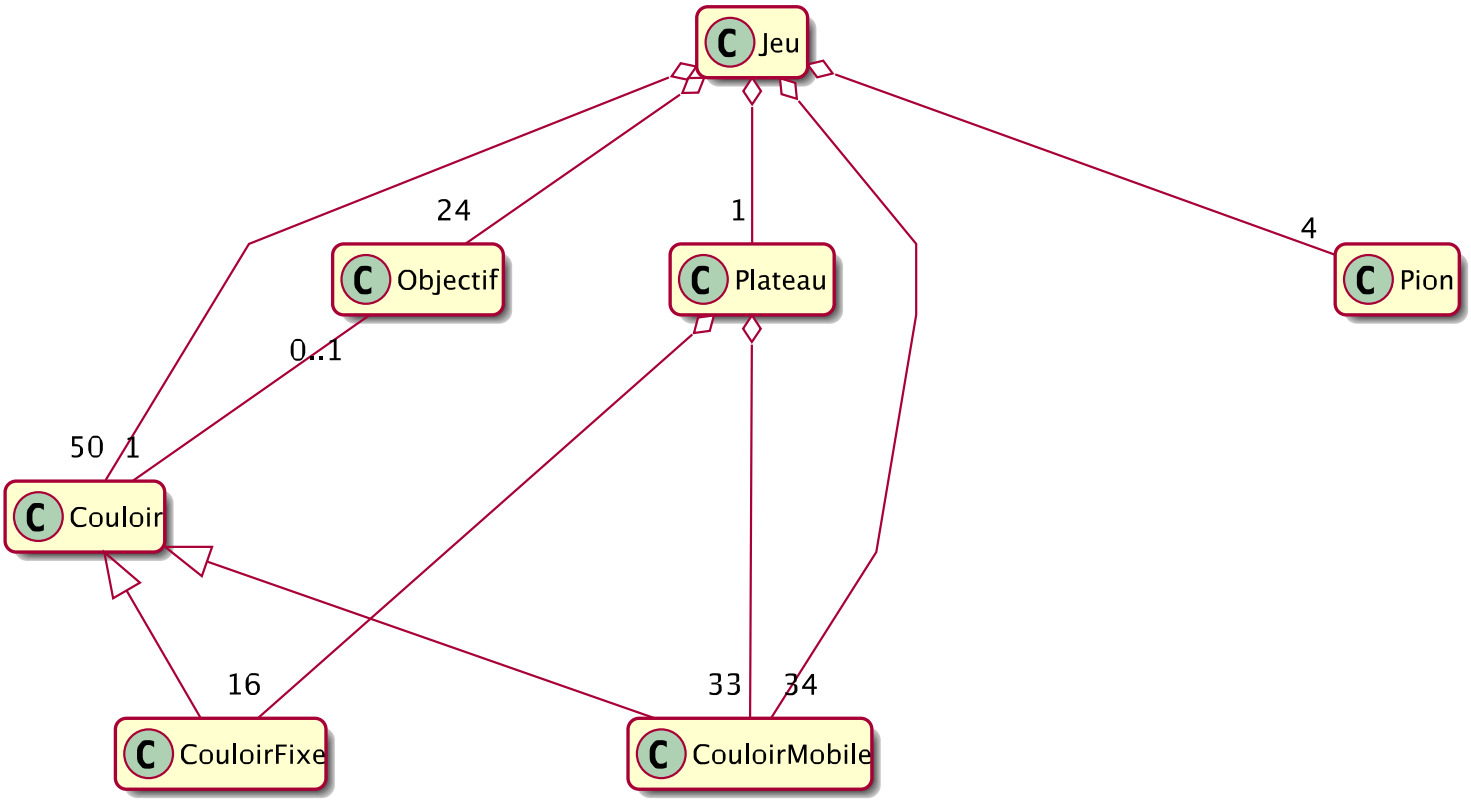
On peut raffiner un petit peu notre conception en rajoutant les couloirs fixes.



Une conception alternative serait d'avoir des couloirs ayant une orientation sans possibilité de la changer, et d'avoir une sous classe ajoutant une nouvelle fonctionnalité, le changement de l'orientation.



On note de plus que les cartes objectif correspondent à des dessins qui se situent sur des couloirs. Il y a donc une relation spécifique entre une carte objectif et un couloir.

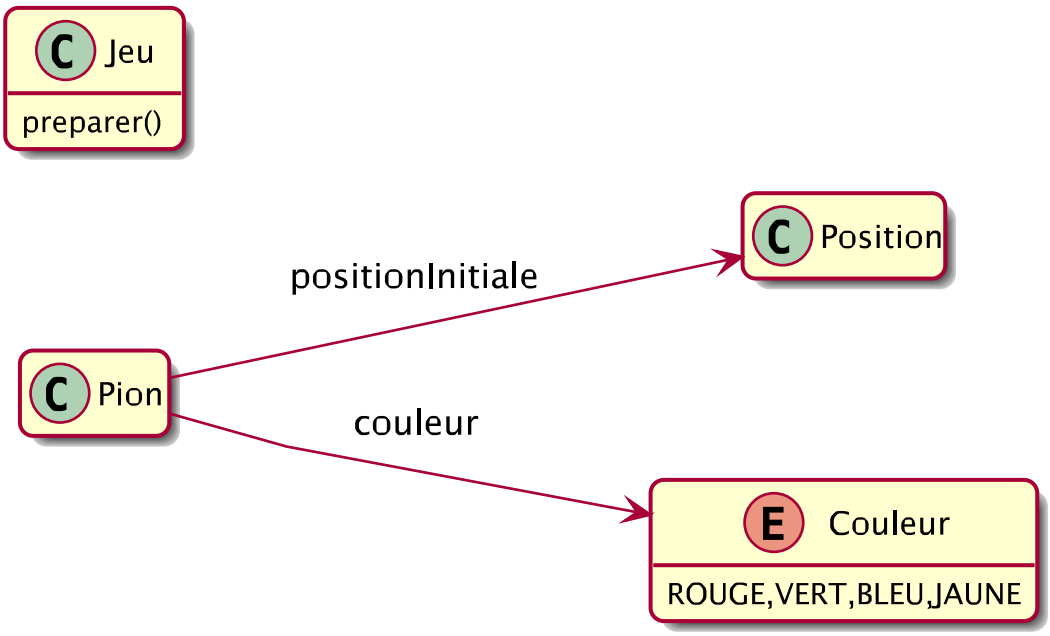


Les règles du jeu

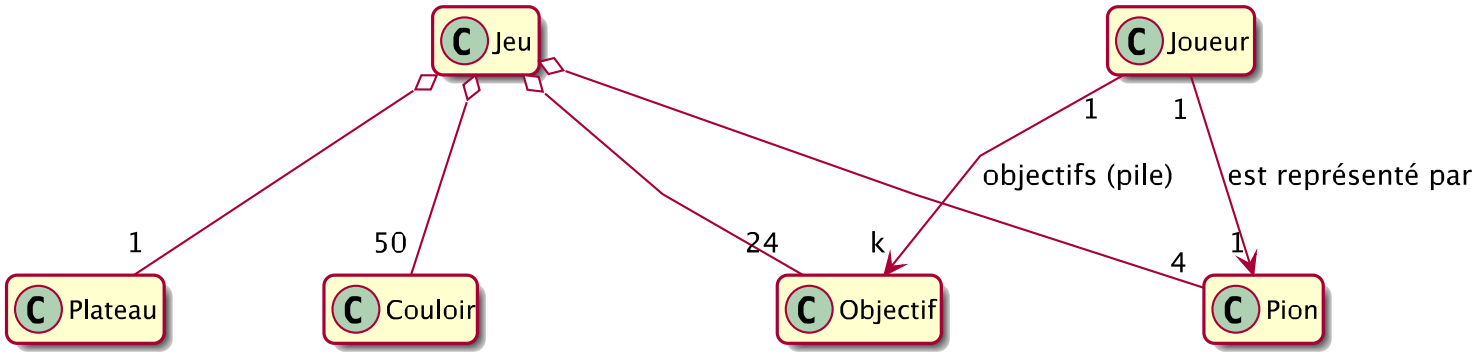
Préparation

Mélanger les plaques face cachée, puis les placer sur les emplacements libres du plateau pour créer un labyrinthe aléatoire. La plaque supplémentaire servira à faire coulisser les couloirs du labyrinthe. Mélanger à leur tour les 24 cartes Objectif face cachée. En distribuer le même nombre à chaque joueur. Chacun les empile devant lui sans les regarder. Chaque joueur choisit ensuite un pion qu'il place sur sa case Départ au coin du plateau de la couleur correspondante.

Une des étapes du jeu est la préparation du jeu. C'est une opération de nombre classe jeu. De plus, on note qu'un pion à une position initiale fixe spécifique à sa couleur. On ne veut pas décider pour l'instant de la façon de gérer la position d'un couloir (numérotation 1 à 49, coordonnées). On crée une abstraction `Position` pour cela. De plus, la couleur permet de désigner un pion particulier dans un monde physique. Ce n'est pas une caractéristique du pion lui même.



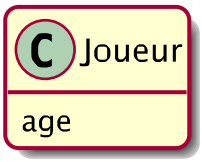
On retrouve maintenant la notion de `Joueur`. On note aussi que les objectifs sont empilés.



Déroulement de la partie

Chaque joueur regarde secrètement la carte supérieure de sa pile. Le plus jeune joueur commence. La partie se poursuit dans le sens des aiguilles d'une montre.

On a besoin de l'âge des joueurs pour déterminer le premier joueur.

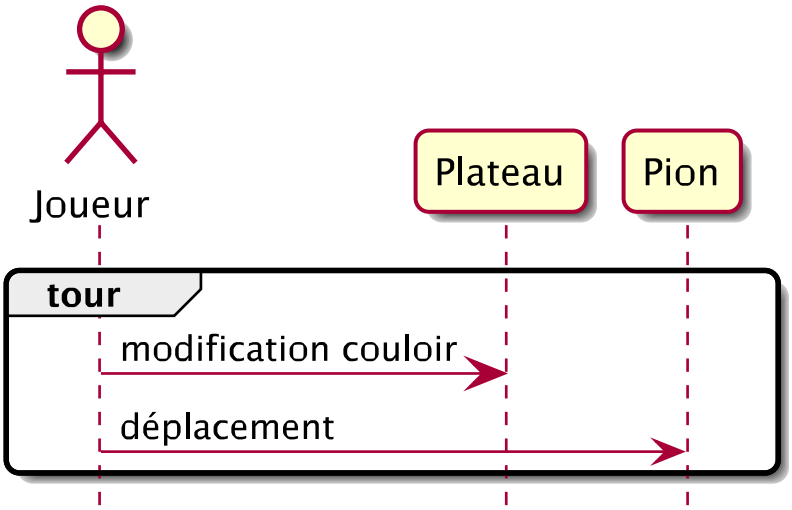


À son tour de jeu, le joueur doit essayer d'atteindre la plaque représentant le même dessin que celui sur la carte supérieure de sa pile. Pour cela il commence toujours par faire coulisser une rangée ou une colonne du labyrinthe en insérant la plaque supplémentaire du bord vers l'intérieur du plateau, puis il déplace son pion.

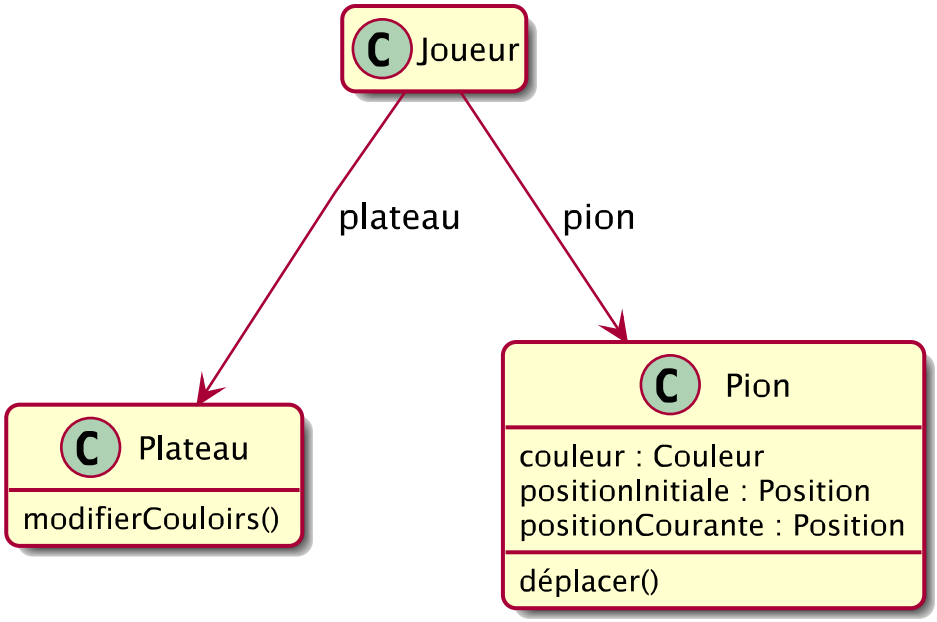
Ainsi, un tour se compose toujours de deux phases :

1. *Modification des couloirs (Introduction de la carte couloir supplémentaire)*
2. *Déplacement du pion*

On peut représenter avec un diagramme de séquence le déroulement d'un tour :



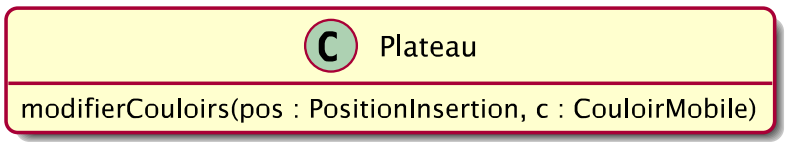
On voit apparaître les premières opérations sur le plateau et sur le pion. On doit modifier les couloirs par insertion d'une carte couloir. On doit pouvoir déplacer un pion. Un pion aura donc une position courante.

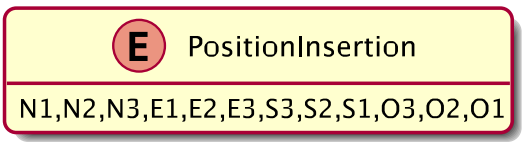


Phase 1 : Modification des couloirs

- *12 flèches sont dessinées en bordure de plateau. Elles indiquent les rangées et colonnes où peut être insérée la plaque supplémentaire pour modifier les couloirs du labyrinthe.*

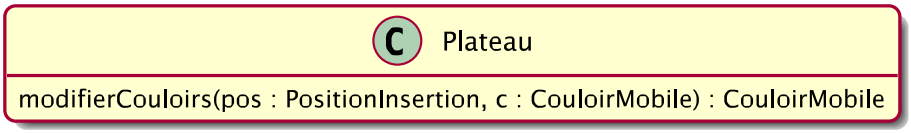
On a besoin d'identifier l'un des 12 endroits ou il est possible d'insérer le couloir libre. On pourrait utiliser un entier de 1 à 12. Mais on ne pourrait pas facilement vérifier que c'est bien une des 12 positions qu'il s'agit. Ici, on décide d'utiliser une énumération en préfixant chaque position de la première lettre de son orientation et un chiffre de 1 à 3. On peut raffiner notre méthode `modifierCouloir()`.

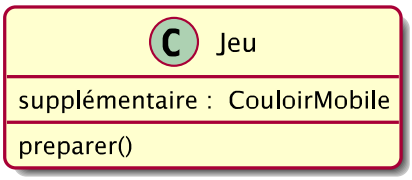




- *Quand vient son tour, le joueur choisit l'une de ces rangées ou colonnes et pousse la plaque supplémentaire vers l'intérieur du plateau jusqu'à ce qu'une nouvelle plaque soit expulsée à l'opposé. La plaque expulsée reste au bord du plateau jusqu'à ce qu'elle soit réintroduite à un autre endroit par le joueur suivant.*

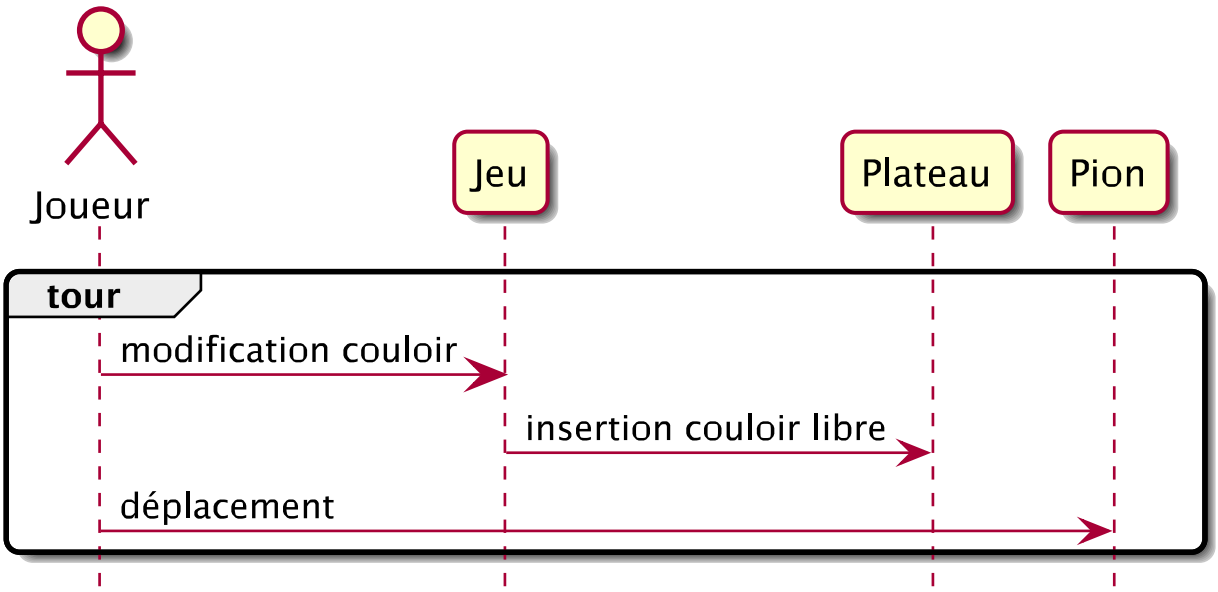
L'introduction du couloir libre dans la rangée permet de libérer un nouveau couloir. On doit récupérer ce couloir, et le stocker. Il s'agit d'un attribut de notre jeu.

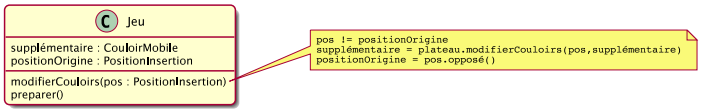


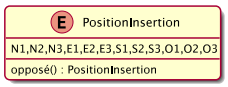


- *Ce dernier n'a cependant pas le droit de réintroduire la plaque Couloir à l'endroit d'où elle vient d'être expulsée !*

Il s'agit d'une contrainte métier. Il faudra garder en mémoire de quelle position vient le couloir libre. On peut créer un nouvel attribut. Du coup, le joueur ne doit pas agir directement sur le plateau mais via le jeu, pour que celui-ci puisse vérifier cette règle du jeu. On note que l'on a aussi besoin de connaître la position opposée à la position d'insertion.

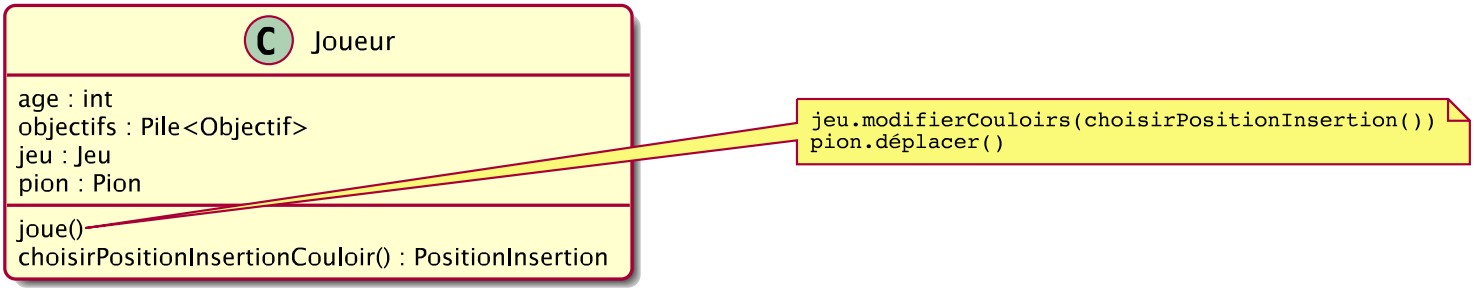






- *Un joueur est toujours obligé de modifier le labyrinthe avant de déplacer son pion, même s'il aurait pu atteindre le dessin recherché sans déplacer les couloirs.*

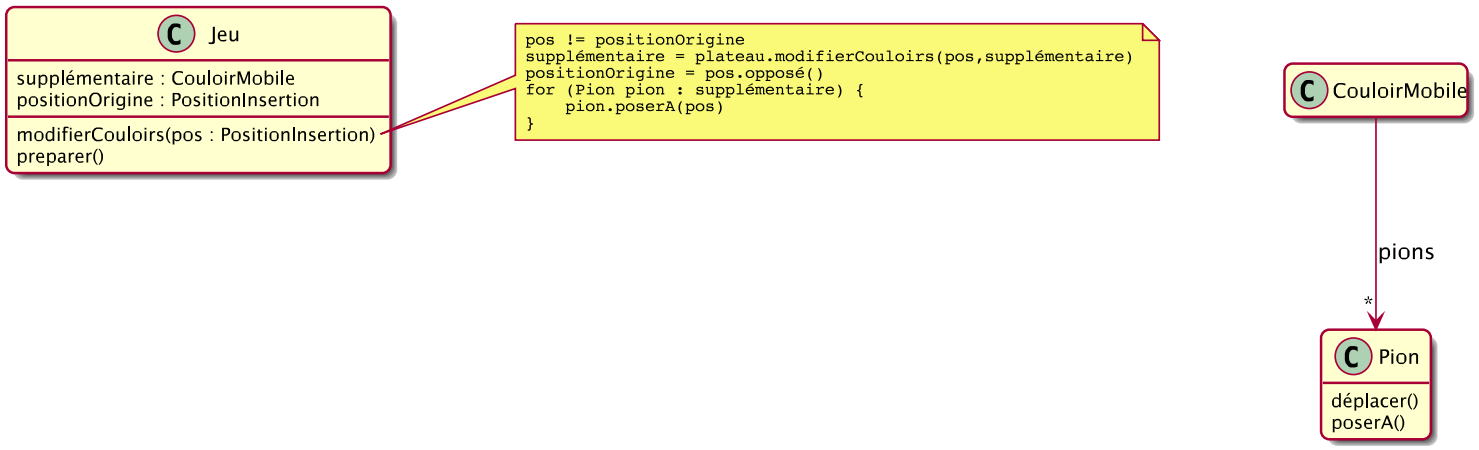
On voit que le joueur doit modifier le plateau quand il joue. Il faut qu'il choisisse à chaque tour à quelle position insérer le couloir. On peut modéliser cela par une opération particulière.



Si, en faisant coulisser les couloirs du labyrinthe, un joueur expulse son pion ou un pion adverse du plateau, il est alors replacé à l'opposé, sur la plaque qui vient d'être introduite. Mais ceci n'est pas considéré comme un déplacement du pion !

Cela signifie que le "jeu" peut modifier l'emplacement d'un pion.

On rajoute cette fonctionnalité à notre pion.

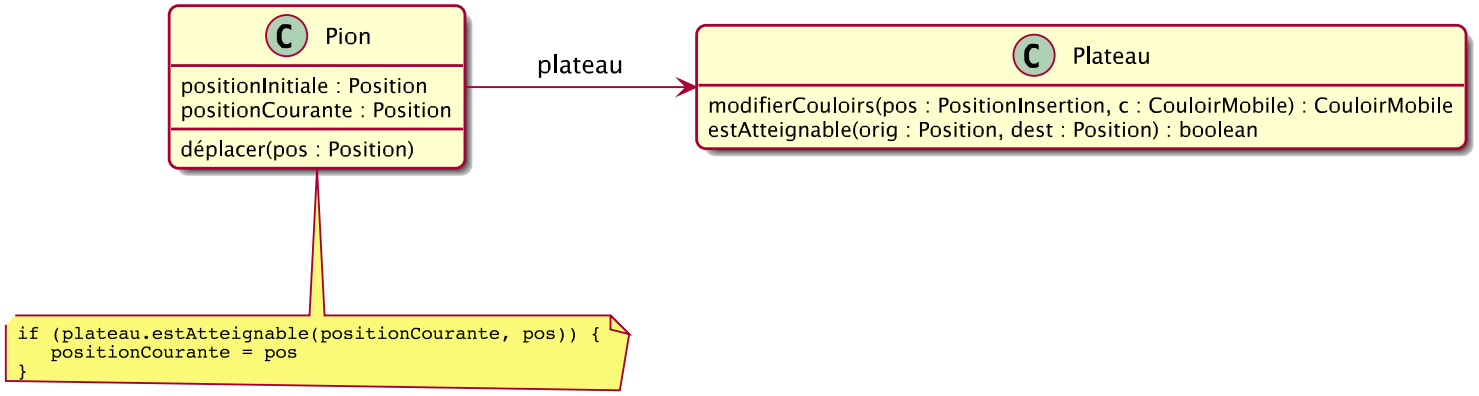


Phase 2 : Déplacement du pion

- *Dès qu'il a modifié le labyrinthe, le joueur peut déplacer son pion. Il peut le déplacer aussi loin qu'il veut jusqu'à n'importe quelle plaque en suivant un couloir ininterrompu.*

Ici, il faudra déterminer qui vérifie que le couloir est ininterrompu. Est-ce le jeu ? Est-ce le plateau ? Est-ce le pion ? Est-ce le joueur ?

A priori, ce n'est pas le rôle du joueur. Pour les autres classes, la question peut se poser. C'est cependant le plateau qui dispose de toutes les informations. On peut rajouter une opération permettant de vérifier qu'une position destination est atteignable d'une position d'origine.



On peut notifier le joueur que le déplacement n'est pas possible : par un message, pas une impossibilité de déposer le pion, etc.

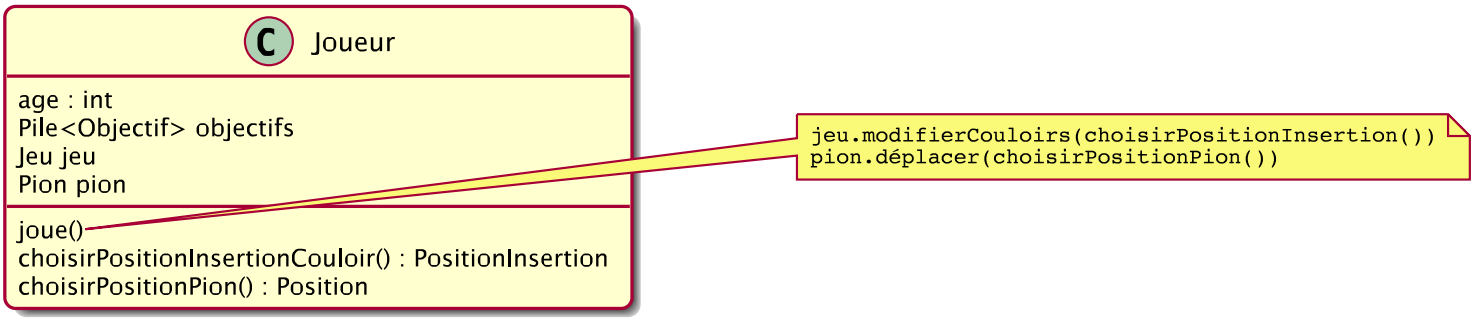
- *Un joueur peut même s'arrêter sur une case déjà occupée. S'il veut, il peut aussi choisir de rester sur place ; il n'est pas obligé de se déplacer.*

Il s'agit de règles métier qu'il faudra vérifier. On note qu'un couloir peu accepter plusieurs pions. On peut penser à permettre l'accès aux pions d'un couloir. Dans la suite des règles, on verra que c'est vraiment important pour les couloirs mobiles.

Question : comment ne pas se déplacer ? Se déplacer à sa position actuelle ou ne pas appeler la méthode déplacer ?

- *Si le joueur n'atteint pas le dessin recherché (= celui figurant sur la carte supérieure de sa pile), il peut déplacer son pion aussi loin qu'il veut de manière à être en bonne position pour le prochain tour.*

On peut laisser au joueur le choix de la position de déplacement du pion. Ce sera au jeu de vérifier que le déplacement est valide.

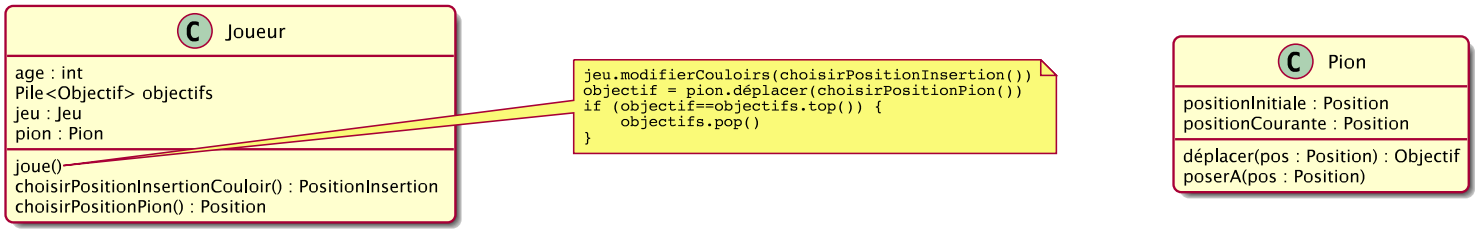


S'il atteint le dessin recherché, il retourne sa carte à côté de sa pile. Il peut immédiatement regarder secrètement la carte suivante de sa pile pour connaître son prochain objectif.

On voit que le déplacement du pion est lié à un objectif donné. Si le pion est sur le dessin mentionné par l'objectif, alors on passe à l'objectif suivant.

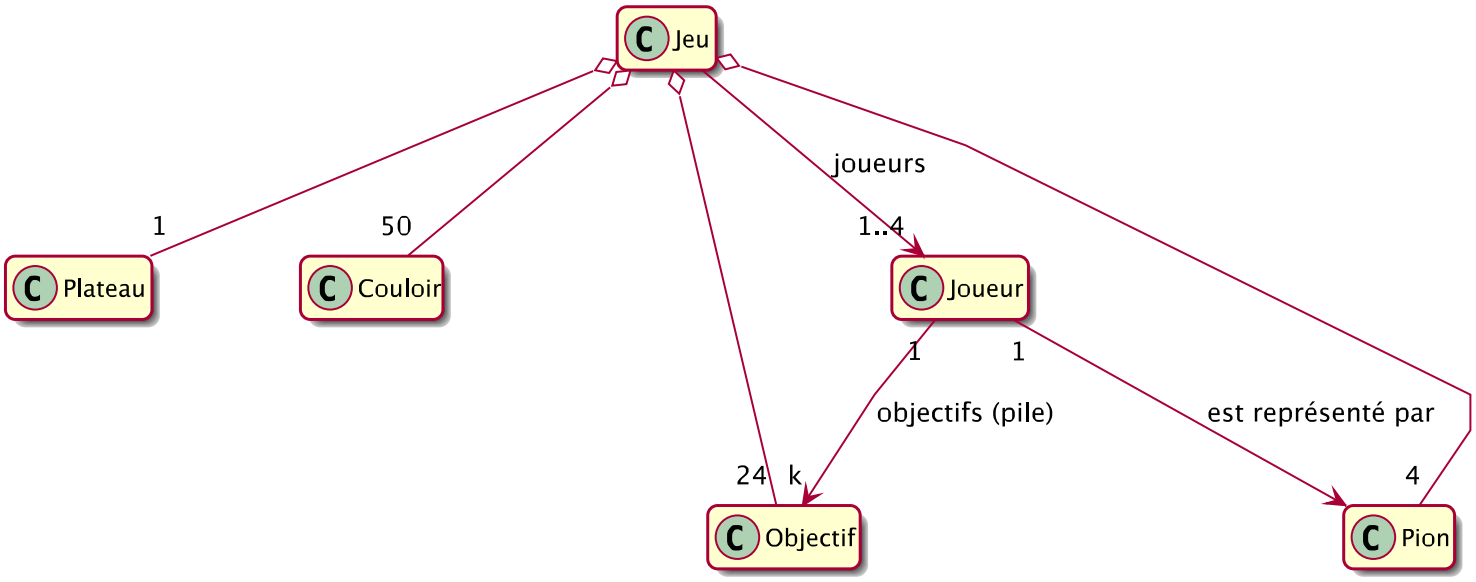
La question qui se pose est : qui vérifie que l'objectif est trouvé ? Est ce le joueur ? Est ce le Jeu ? Est ce le plateau ?

Il faut en tout cas modifier le déplacement du pion pour faire apparaître l'objectif : soit en paramètre si c'est le pion, le plateau ou le jeu qui vérifie l'accès à l'objectif. Soit le dessin situé à la position du pion est retourné.

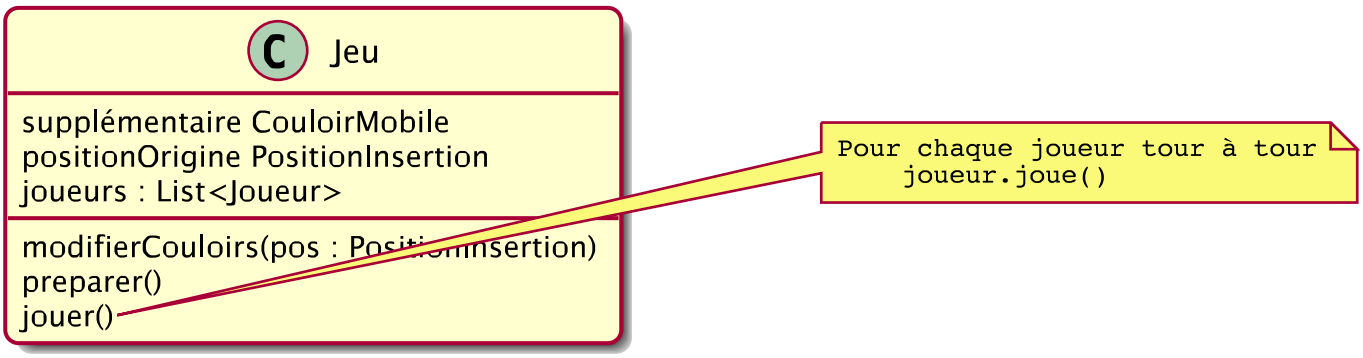


C'est maintenant au tour du joueur suivant de jouer. Lui aussi procède de la même façon : il introduit la carte Couloir supplémentaire, puis déplace son pion en essayant d'atteindre son objectif.

Le jeu doit disposer de la liste des joueurs. Ceux-ci jouent tour à tour. On peut modifier notre diagramme de classe global.



Et mettre à jour la classe `Jeu`.



III – Fin de la partie

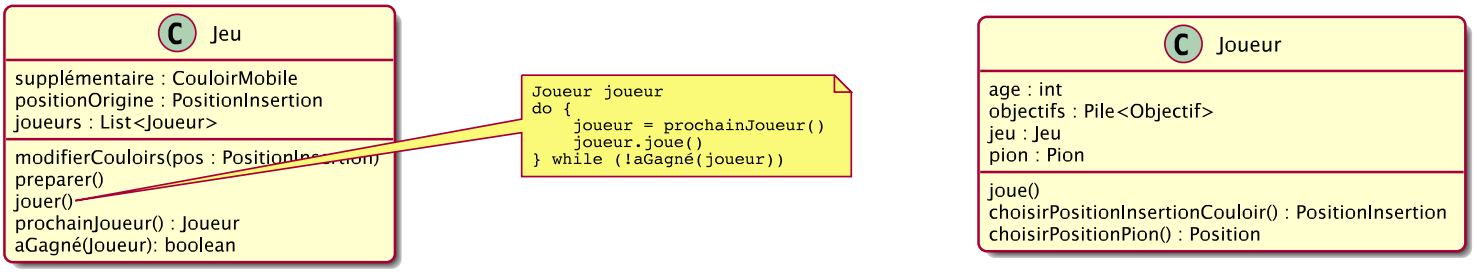
La partie s'arrête dès qu'un joueur a atteint tous ses objectifs et qu'il est revenu à son point de départ. C'est lui qui a su se déplacer le mieux dans le labyrinthe et il remporte la partie !

Il faut détecter la condition d'arrêt du jeu : avoir atteint tous ses objectifs et être revenu au point de départ.

Le joueur pourrait lui-même détecter cette situation. Néanmoins, on préférera laisser cette responsabilité au jeu.

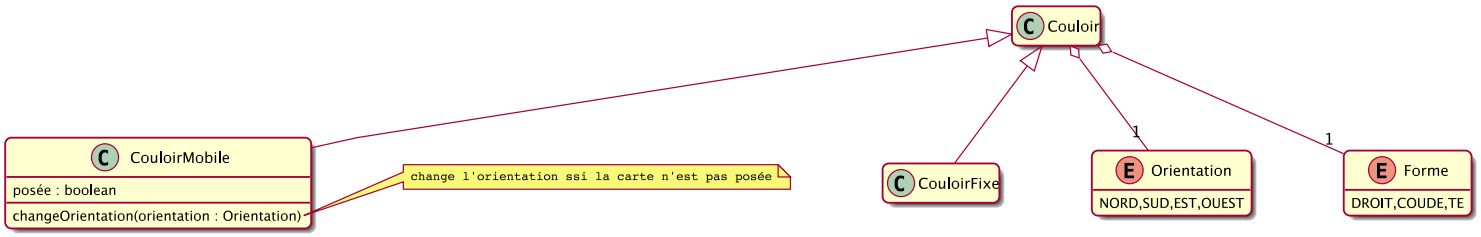
On ne décide pas pour l'instant comment le jeu peut décider si un joueur a atteint ses objectifs : on peut compter le nombre de cartes objectif visibles par exemple, ou regarder la taille de la pile.

On note que chaque joueur doit connaître son point de départ, et sa position courante. Cette information est gérée par son pion. Cette information est disponible à l'objet jeu.

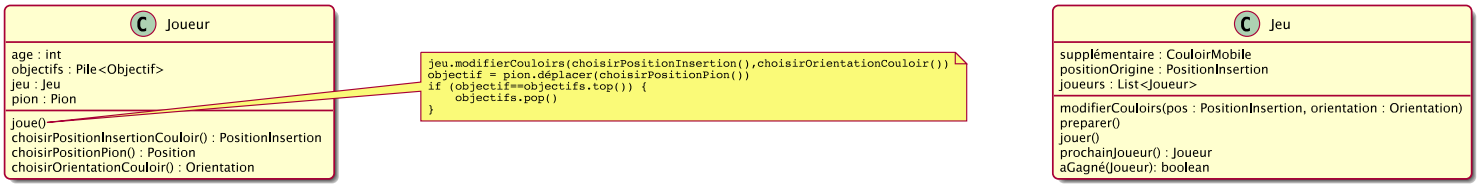


Au delà des règles du jeu

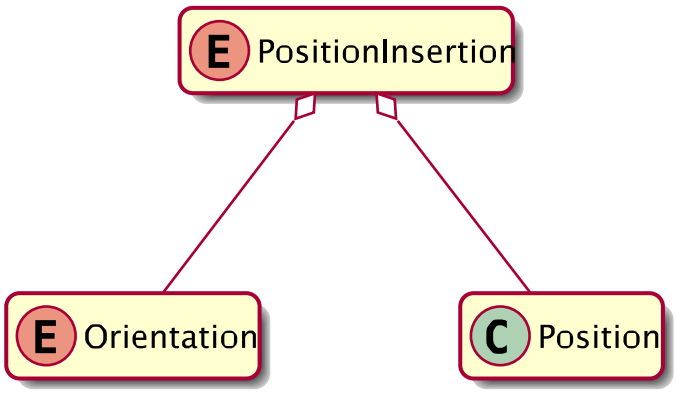
Si on regarde de plus près les couloirs, on note qu'ils sont de trois formes : droit, coudé ou en t. Ces couloirs sont orientés dans les directions cardinales. Les couloirs fixes ont une orientation qui ne peut être changée. Les couloirs mobiles ont une orientation qui peut être choisie lors de l'insertion sur le plateau. Une fois posée, l'orientation ne peut plus être modifiée.



Du coup, cela signifie que le joueur doit décider comment orienter le couloir avant de l'insérer.

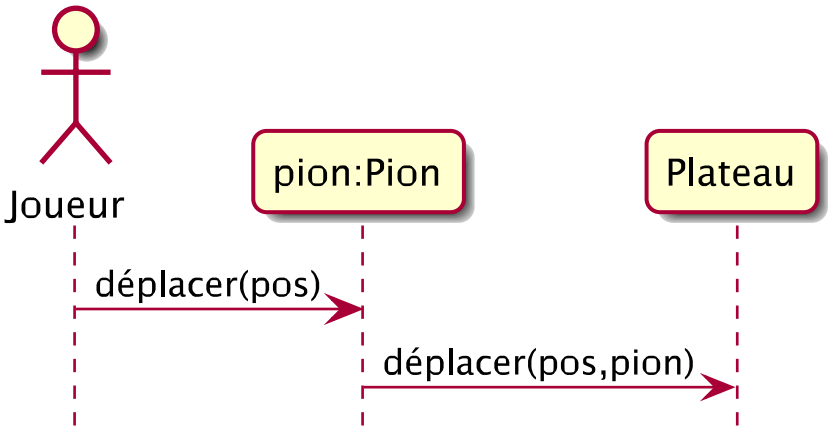


De plus, on note que notre PositionInsertion correspond en fait à une position et l'orientation du décalage.



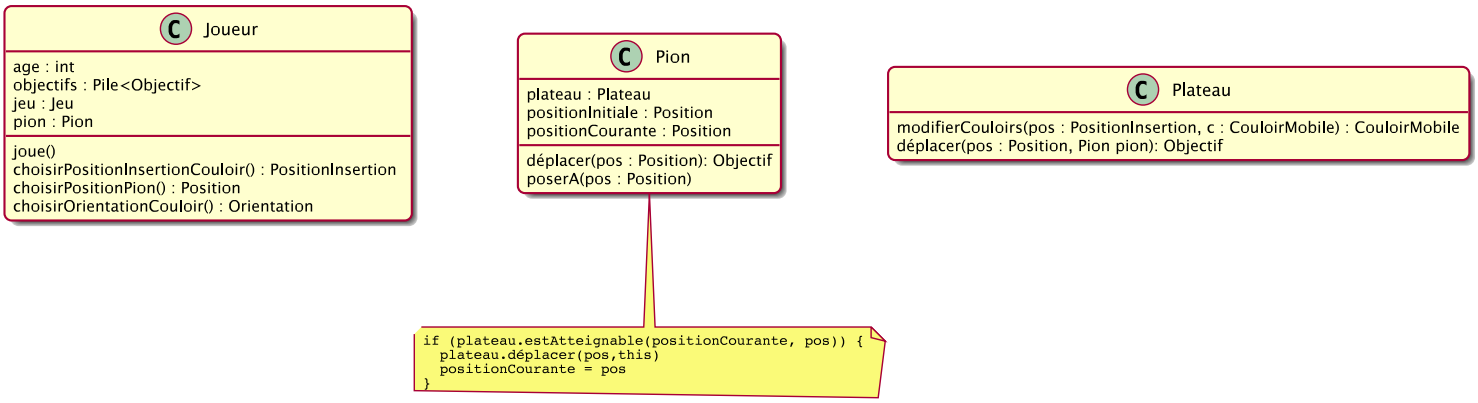
Par exemple, N1 correspondrait à l'orientation SUD et à la position (1,2) si on utilise des coordonnées.

La relation entre le pion et le reste du jeu n'est pas évidente. Un diagramme de séquence doit nous permettre d'y voir plus clair.



Le pion délègue son déplacement au plateau, qui a une vue globale de la situation : il est le seul objet à pouvoir vérifier si un chemin existe entre la position du pion et la position destination.

Que se passe t'il si la destination n'est pas atteignable ?



Remarques :

Il y a pas mal de possibilités concernant les structures de données et les algorithmes à utiliser pour déterminer si il existe un chemin entre deux points.

Application des principes GRASP

Créateur

Qui va créer nos différents objets ?

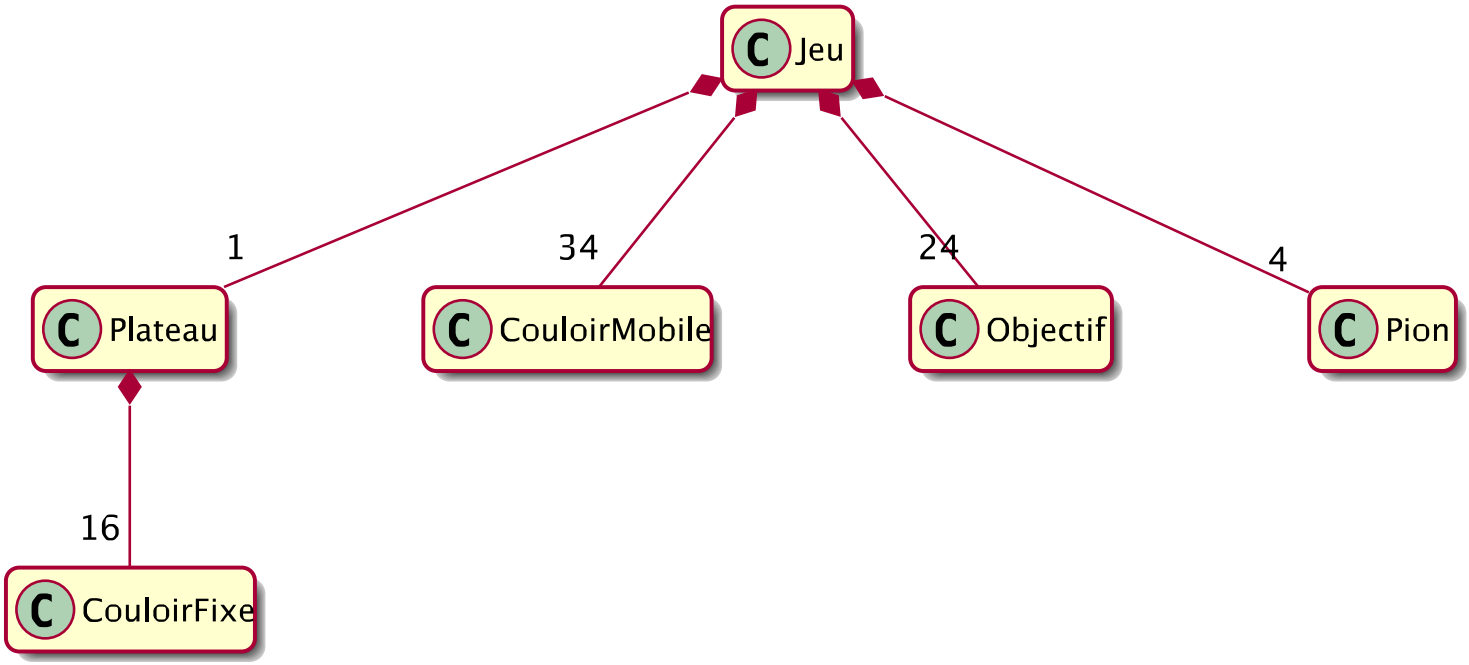
Le plateau est le seul à pouvoir manipuler les couloirs fixes. Ceux-ci sont donc créés par le plateau. Pour la même raison, le jeu créé les couloirs mobiles et le plateau.

Les cartes objectifs et les pions sont manipulées par le joueur. Cependant, ce sont vraiment des objets qui appartiennent au jeu. C'est je jeu qui va donner en début de partie les objectifs aux joueurs. C'est aussi le jeu qui va donner son pion aux joueurs.

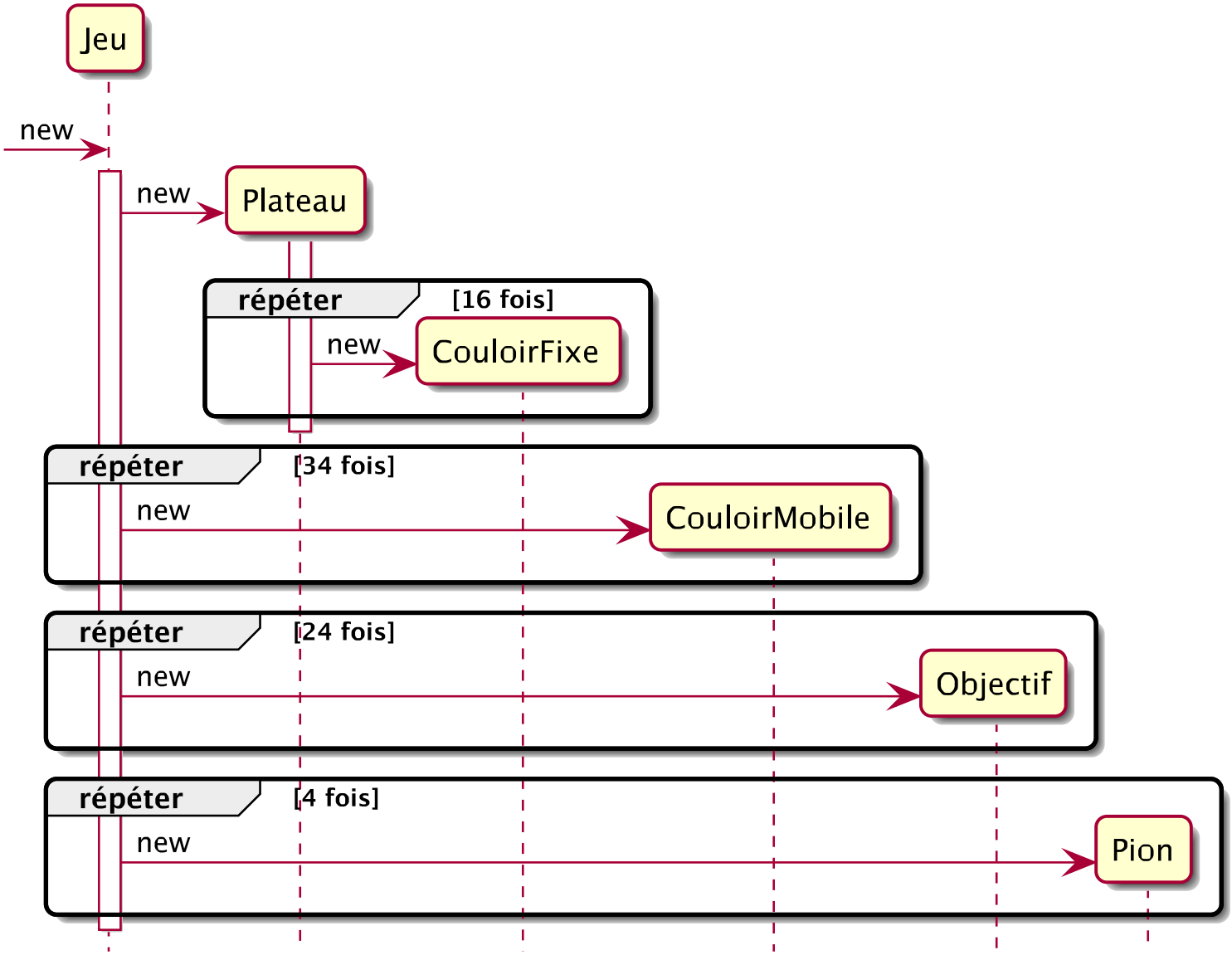
La question peut se poser concernant la classe `Joueur`. Soit on permet à l'objet joueur de s'enregistrer auprès du jeu, soit le jeu crée un objet joueur quand quelqu'un souhaite jouer.

Dans notre cas, on souhaite de la liberté sur le type de joueur : on peut imaginer que les joueurs soient des humains et puissent jouer via une interface graphique, mais aussi que l'on dispose de joueurs artificiels. Les joueurs sont donc créés en dehors du jeu.

On peut donc remplacer dans notre diagramme les agrégations par des compositions pour marquer cette responsabilité de création.

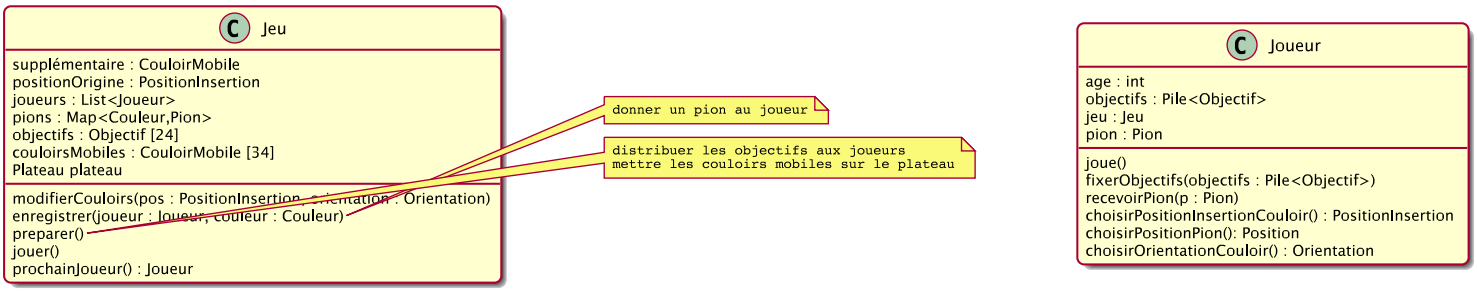


On peut visualiser l'ordre de création des divers objets à l'aide d'un diagramme de séquence.



On peut rajouter de nouvelles opérations sur notre jeu :

1. l'enregistrement d'un joueur auprès du `Jeu` pour récupérer le pion associé à une couleur donnée,
2. et une méthode dans `Joueur`pour recevoir les cartes objectifs et un pion pour jouer.



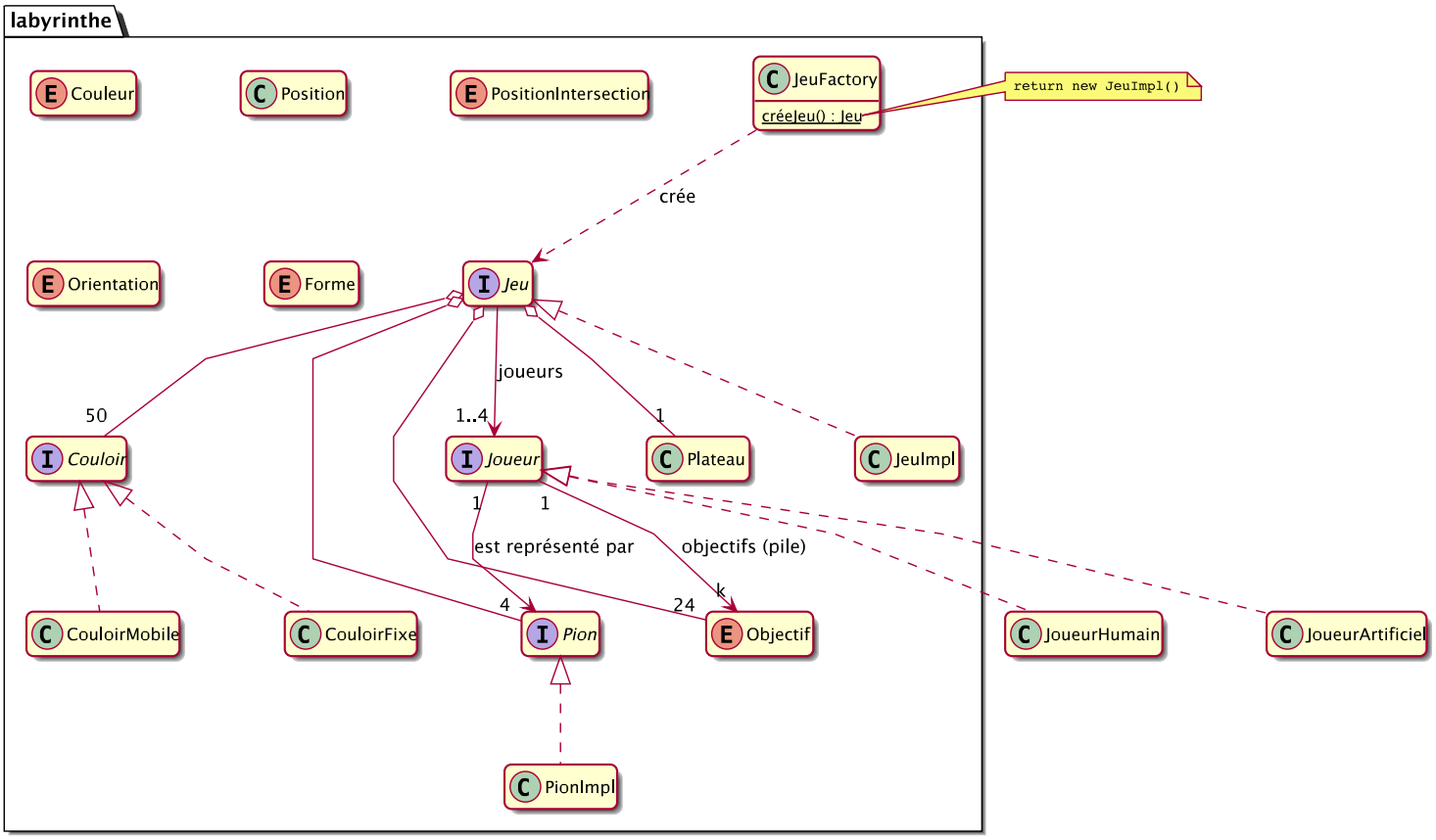
Contrôleur

Qui contrôle l'interaction avec l'utilisateur ? Il s'agit dans notre cas du jeu. On passe cependant par un pion pour le déplacement. Ne faudrait-il pas simplement demander au jeu de déplacer le pion ?

Indirection

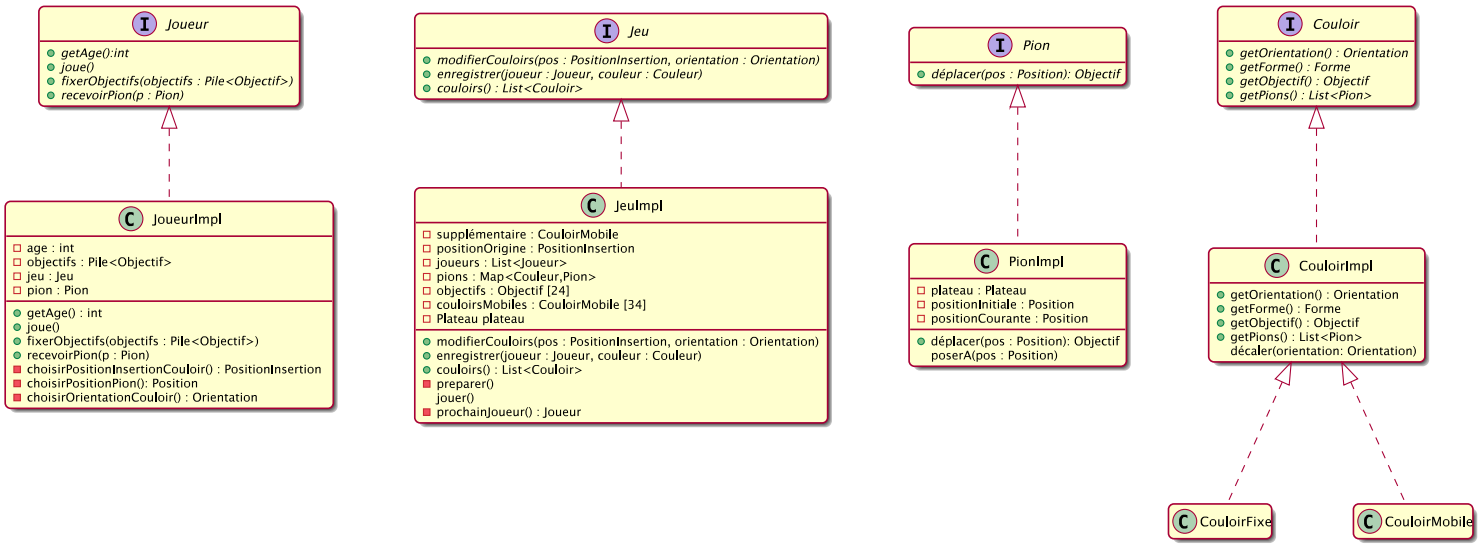
Pour l'instant, on travaille directement sur des classes concrètes. Il faudrait séparer ces classes en interface et implémentation pour avoir plus de flexibilité dans l'évolution de l'application.

On a besoin d'interfaces pour tous les objets manipulés par le joueur et pour le joueur lui-même. Les cartes objectif n'ayant pas d'état, on peut les représenter par une énumération par exemple.



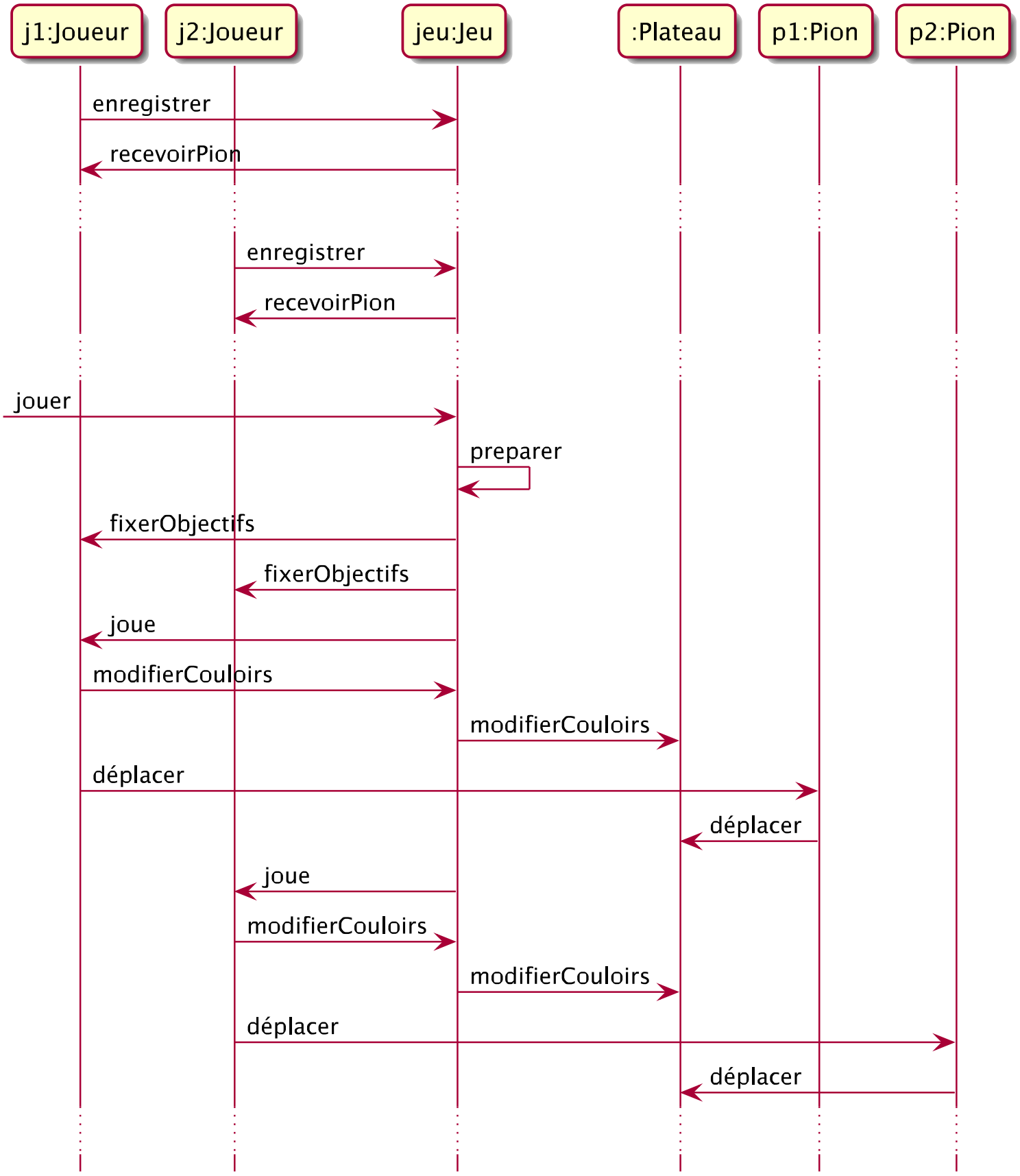
A priori, le plateau peut rester une classe concrète. La question se pose cependant pour les couloirs. Est ce que les couloirs doivent être disponibles pour le joueur ? C'est le cas si on souhaite un joueur artificiel, il faut donc une interface pour les représenter.

On obtient les définitions suivantes pour nos interfaces et leurs réalisations. On note qu'un certain nombre de méthodes ne sont pas publiques, pour respecter le principe d'encapsulation. Seules les méthodes de l'interface sont publiques.



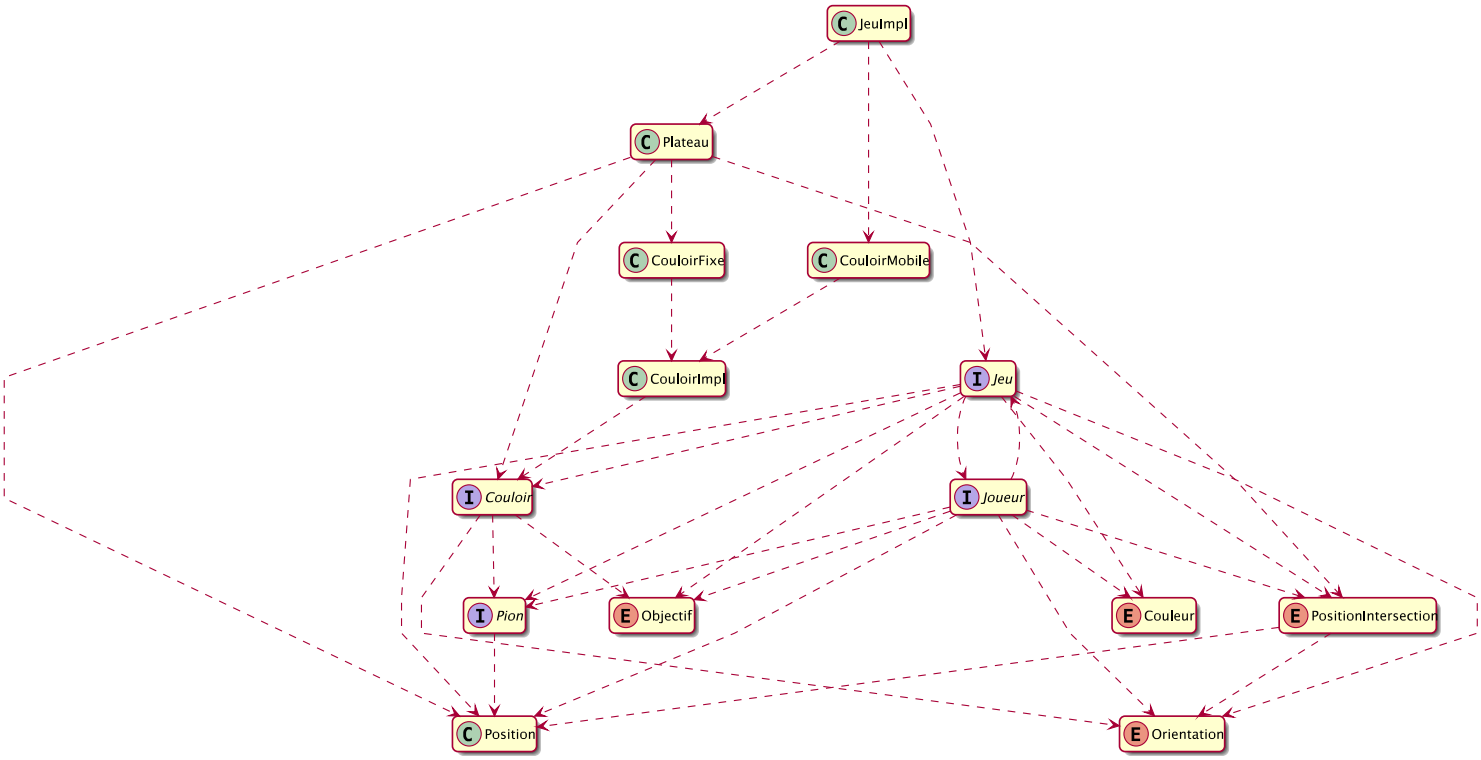
On aura besoin d'une fabrique pour obtenir une instance de Jeu sans dévoiler sa classe concrète.

On représente dans le diagramme de séquence ci-dessous le début d'une partie comportant deux joueurs.



Faible couplage

Il faut limiter les dépendances entre les classes. On obtient le diagramme de dépendances suivant.



On note la dépendance circulaire entre `Jeu` et `Joueur`. Le jeu a besoin de donner un pion et des objectifs au joueur. Le joueur utilise le jeu comme intermédiaire pour insérer le couloir libre sur le plateau.

On note que les classes concrètes dépendent généralement des interfaces ou des énumérations, ce qui respecte le principe d'inversion des dépendances (DIP) de l'oncle bob. Reste le cas de la classe concrète `Position` dont beaucoup d'interfaces dépendent. Cela viole le principe DIP. Ce n'est cependant pas très grave puisqu'elle ne dépend d'aucune autre classe. On pourrait cependant là aussi séparer l'interface de l'implémentation pour respecter le principe DIP.