Rapport de génie logiciel : Monopoly

Par LOUGANI Faouzi et GUARDIA Quentin
Licence 3 d'informatique à l'UPVD année 2019-2020

Table des matières

Sources : upload.wikimedia.org

Analyse des besoins

Pour commencer, nous avons réalisé des diagrammes de cas d'utilisation, pour répondre à la question : qui peut faire quoi et comment ? Grâce à eux, nous voyons plus clairement quelles actions un joueur aura la possibilité de réaliser et quelles sont les contraintes. Voici quelques cas que nous avons pu relever, à partir du sujet :

- Tirer les dés, qui correspond à la première action du joueur lorsque c'est son tour.
- Acheter une propriété. Cependant, il y a certaines conditions à respecter avant de pouvoir acheter. Il faut que la propriété n'appartienne à personne et que l'utilisateur ait assez d'argent. On fait donc deux includes avec ces conditions.
- Construire. On a deux possibilités, qui sont construire un hôtel ou bien une maison. Construire est donc une généralisation de ces deux cas. Ici, la condition a respecter est d'avoir tous les terrains de la même couleur avant de construire sur l'un d'eux. D'où l'include visible sur le diagramme.
- S'acquitter d'un dû. C'est la généralisation de trois cas. Premièrement il y a payer un loyer, à condition que le terrain appartienne à un joueur. Ensuite il y a payer les impôts, si on s'arrête sur les cases impôts sur le revenu. Enfin, vient payer les taxes de luxe. Pour cela, le joueur doit s'arrêter sur le case correspondante.
- Tirer une carte. Ce qui généralise tirer une carte « chance » et tirer une carte « caisse de communauté »
- Aller en prison. Cela se fait uniquement si le joueur tombe sur la case « Aller en prison » ou s'il tire une carte lui demandant d'aller en prison.
- Recevoir de l'argent, qui généralise « toucher un salaire de 20000€ », nécessitant de passer par la case départ et « bénéficier du parc gratuit », lorsque le tombe sur la bonne case

Voici pour les principaux. Comme évoqué, les includes traduisent les conditions. Le diagramme de cas d'utilisation se trouve dans les annexes.

Conception générale

Puis, pour mettre en relation tout cela nous avons créé un modèle du domaine, l'approche est un peu plus technique. C'est un diagramme de classe sans méthode. Il permet de donner une idée générale de l'ordonnance des différents éléments.

Nous avons réalisé des associations suivantes, entre :

- Joueur et Propriété, qui hérite de case. L'association est « vend / achète / possède »

- Propriété et Titre de propriété. L'association se nomme « possède »
- Terrain, héritant de Propriété et Maison et Hôtel avec l'association « construit »
- Joueur et dés. Le joueur peut tirer les dés
- Terrain et Titre constructible, héritant de Titre de propriété. Ici c'est « détient »

On a dû faire cette dernière association car Terrain est la seule Propriété où il est nécessaire de définir le prix des hôtels et des maisons.

La classe Case est la généralisation de tous les types de cases possibles :

- Chance et Caisse de Communautés, qui servent à tirer une carte
- Propriété
- Prison
- Transaction qui regroupent toutes les cases qui permettent au joueur de gagner ou perdre de l'argent.

Pour cette dernière, il suffira juste de jouer sur le signe de l'entier « montant » en attribut. Case a trois attributs : l'entier numero_case, le chaîne de caractères nom_case et la chaîne de caractère type_case. C'est une composition forte de la classe Plateau, qui a pour attribut le tableau de cases.

La classe Propriété regroupe, elle, Terrain, Gare et Compagnies. À savoir que Propriété a pour attribut l'entier propriétaire indiquant l'id du joueur propriétaire s'il y en a un, 0 sinon, et son prix. Et la classe Terrain à pour attribut l'entier noGroupe pour représenter les différents groupes de couleur sur le plateau.

La classe Titre de propriété a deux entiers pour attribut : numéro_titre_propriété et. Sa classe fille, Titre constructible, a également deux entiers pour attribut : prix_hotel et prix_maison.

La classe Joueur à quant à elle quatre attributs : les entiers num_joueur et solde, ainsi que la chaîne de caractères nom_joueur, l'entier solde et l'entier position, correspondant à la case où il se trouve. On a décidé de rajouter la chaine de caractère ListeProp qui contient la liste des propriétés du joueur, que l'on devra afficher.

Enfin, la classe Dés a pour attribut la valeur des dés tirés.

On abordera les méthodes dans la partie suivante, comme par exemple celle permettant de calculer le loyer. Par soucis de clarté, nous n'avons pas recopié les cardinalités ni les types des attributs. Se référer aux annexes pour plus de précision ?

Afin d'ajouter de la temporalité à ce diagramme, nous lui avons associé un diagramme de séquence. Et ce pour chaque cas d'utilisation. Ils sont pour l'instant minimalistes, il n'y a que deux lignes de vie qui correspondent à « Joueur » et « Système ». Cette dernière regroupe en réalité plusieurs acteurs du diagramme de classe. Chaque diagramme commence par un message synchrone du joueur vers le système indiquant qu'il souhaite tirer les dés. Le système s'envoie ensuite un message pour effectuer le déplacement du

joueur vers sa nouvelle case. Nous avons choisis de rendre trois diagrammes de séquence avec le projet final :

- achat d'une propriété : Le joueur demande à acheter la propriété. Le système vérifie si la case a un propriétaire et l'entier solde du joueur. Si la case a un propriétaire(OPT : propriétaire=1) alors on effectue le traitement suivant. Si son solde est inférieur au prix de la propriété (ALT), alors le Système retourne un message à l'utilisateur lui indiquant que l'achat est impossible. Sinon Le système soustrait le prix de la propriété au joueur. Et rien ne se passe si la case n'a pas de propriétaire.
- tirer une carte : Le Système vérifie le type de la case. Si c'est une case de type chance (ALT), le Système fait tirer au joueur une carte de type chance. Sinon si c'est une case communauté, il fait tirer au joueur une case caisses & communautés.
- s'acquitter d'un dû : d'abord, Système vérifie si la nouvelle case est une case transaction (qui permet de gagner ou bien perdre de l'argent) ou une propriété. Si la case est une case transaction (ALT), alors on vérifie le montant(OPT) : s'il est négatif il s'agit d'un dû. Le système retire alors l'argent au joueur et lui envoie un message afin de l'en informer. Si c'est une case propriété(ALT), on vérifie si la case a un propriétaire (OPT). Si elle en a un, alors le Système transfère le loyer vers le propriétaire et informe également le joueur concerné.

Les diagrammes finaux se trouvent dans les annexes. Malheuresement, Umbrello ne permet pas d'afficher le contenu des opt.

Conception détaillée

Pour agrémenter les diagrammes précédents, nous avons dû intégrer les méthodes au diagramme de classe. Pour, cela nous avons commencé en créant deux nouvelles classes : l'une nommée MonopolyMain et les interfaces jeux et menu. La classe MonopolyMain contient toutes les méthodes du système : Tirer_Dé(), Tirer_carte_caisse(), Tirer_carte_chance(), acheterMaison() et annoncerGagnant(). Nous l'avons liée à la classe Plateau, qui contient toutes les cases et utilise l'interface Jeu.

Il y a deux classes d'interface. La première s'appelle « Jeu » comme expliqué précédemment et Menu qui est utilisée par la classe Joueur.

Toutes les méthodes ajoutées sont pour l'instant publiques. Il y a getNom() dans Case afin d'avoir le nom de la case et tirer_carte() dans les cases Chance et Caisses de communautés pour obtenir la consigne d'une carte. On a également ajouté setProprio() à

utiliser lors d'un achat dans Propriété pour signaler le propriétaire. Dans la même classe, on a rajouté Disponibilité() qui est en réalité un getter permettant de voir si la propriété est libre ainsi que getPrix(). Le montant du loyer est calculé par les méthodes calculerLoyer(), dans titre_de_propriété et titre_constructible selon si l'on peut construire sur la propriété ou non. Comme le loyer se calcule différemment selon le type de case, on utilisera peut-être le polymorphisme de surcharge afin d'avoir l'algorithme adéquat au type de case, aux éventuels hôtels, maisons et valeur des dés. Pour faciliter la tâche, nous avons rajouter des méthodes permettant de renseigner le nombre et le prix de chaque hôtel et le groupe dans la classe Terrain, ainsi que de les construire.

Le Plateau est désormais muni d'une méthode permettant de récupérer la case à l'index souhaité. De 0 à 39 donc.

Enfin, nous avons ajouter à la classe Joueur les méthodes gérant le nom, le solde, les transaction, l'achat de bien, la liste de propriétés et le déplacement.

Nous avons, suite à cela, pu remplacer les lignes de vie génériques des diagrammes de séquences nommées « Systèmes » par les vraies classes et les interactions par les vraies fonctions. Pour plus de détail, il faut voir les annexes.

Réalisation

Le langage qui nous semblait le plus adapté au vu du projet et de nos connaissances personnelles et le Java. Nous avons suite aux étapes précédentes, généré le code en Java à partir du logiciel. Chaque classe demandé était présente et nous étions globalement satisfait de la structure fournie, hormis quelques doublons qui généraient des erreurs. Nous avons par la suite créés les constructeurs nécessaires, le comportement des méthodes qu'Umbrello ne pouvait pas prévoir ainsi que le main. De plus Umbrello a mal généré le code qui touche à l'héritage de classe. Nous avons pu régler ce problème grâce à l'usage de la fonction super.

À propos du développement, il serait fastidieux d'expliquer tout ce que nous avons fait ici. Le mieux est de se référer aux commentaires dans le code. Pour illustrer, nous avons implémenté la méthode « Disponibilité » de la classe « Propriété ». Cette méthode récupère la valeur entière de l'attribut « propriétaire » qui vaut 0, par défaut, s'il n'y a pas de propriétaire, l'id du propriétaire sinon. Puis elle retourne un booléen True si la case est disponible, False sinon. En implémentant de la sorte, nous avons pu ré-ajuster quelques détails sur le diagramme de classe notamment.

À noter que certaines classes comme le Plateau ou les Cases ont étaient beaucoup éditées par rapport à la version originale d'Umbrello. En effet, le diagramme de classe sur

lequel se base Umbrello ne contient pas toutes les informations de l'interface graphique comme l'abscisse, l'ordonnée et les tailles des différents objets. Ce qui a dû être rajouté dans les constructeurs voire certaines méthodes.

Intégration

Nous avons au début rencontré quelques problèmes concernant ce qui touche à événementiel et au rendu graphique. L'événement permettant de tirer les dés affichait bel et bien des numéros aléatoires. En revanche, on n'arrivait pas à récupérer la valeur pour déplacer le Joueur dans le main. Ce type de problème ayant été rencontré plusieurs fois, on a compris que l'usage de variable « static » était primordial. Nous avons eu des problèmes d'affichage car le plateau démarre en bas et il fallait que chaque case s'agence bien et que les pions puissent circuler dessus.

Par soucis de temps, le code fonctionne pour deux jours bien qu'il puisse être fonctionnel pour plus. Dans l'idéal il y aurait le choix du nombre de joueurs, de leur nom et de leur couleur. Aussi, le premier joueur

En l'état actuel des choses, le programme se déroule comme suit : il y a des variables globales permettant d'accéder au joueur courant et à la case courante du joueur. Ceci grâce aux listes des joueurs et des cases. Le premier joueur tire les dés. Selon la case sur laquelle il tombe, soit il gagne ou perd de l'argent (case transaction), soit il peut acheter (case propriété), soit il doit tirer une carte (cases chances et caisse de communautés). La gestion de l'ensemble des possibilités est réalisée par la fonction verifier() qui est appelée par l'événement du tir de dés et contient un switch case. Si le joueur achète une propriété constructible, alors il peut acheter des maisons jusqu'à obtenir un hotel. Si le joueur doit tirer une carte, le joueur d'après ne peut pas tirer ses dés tant que la carte n'est pas tirée. On a utilisé la possibilité en Java d'activer ou désactiver les boutons pour sécuriser les actions des joueurs.

On a pensé à tous les cas. Par exemple, si le joueur tombe sur une case déjà achetée et qu'il n'est pas propriétaire alors il doit payer le loyer, qui est calculé par les fonctions définies dans l'étape précédente. Puis une transaction s'effectue selon, une fois de plus à l'aide des fonctions précédemment définies.

On s'est ensuite arrangé pour que l'affichage soit suffisant. On a calculé le placement des pions des joueurs dans les cases grâce à une liste d'entiers. On a fait des bords plus épais de couleur différentes en fonction du type de la case. Une zone permet d'afficher les informations sur le joueur. Pour ne citer que cela.

On a du légèrement modifier certaines parties du programme, tout en conservant la structure du projet.

Annexes

Toutes les images sont personnelles et tirées du logiciel Umbrello et illustrent le rapport. Elles sont également disponibles dans le dossier afin de pouvoir zoomer dessus plus aisément.

Voici le diagramme cas d'utilisation :

Modèle du domaine :

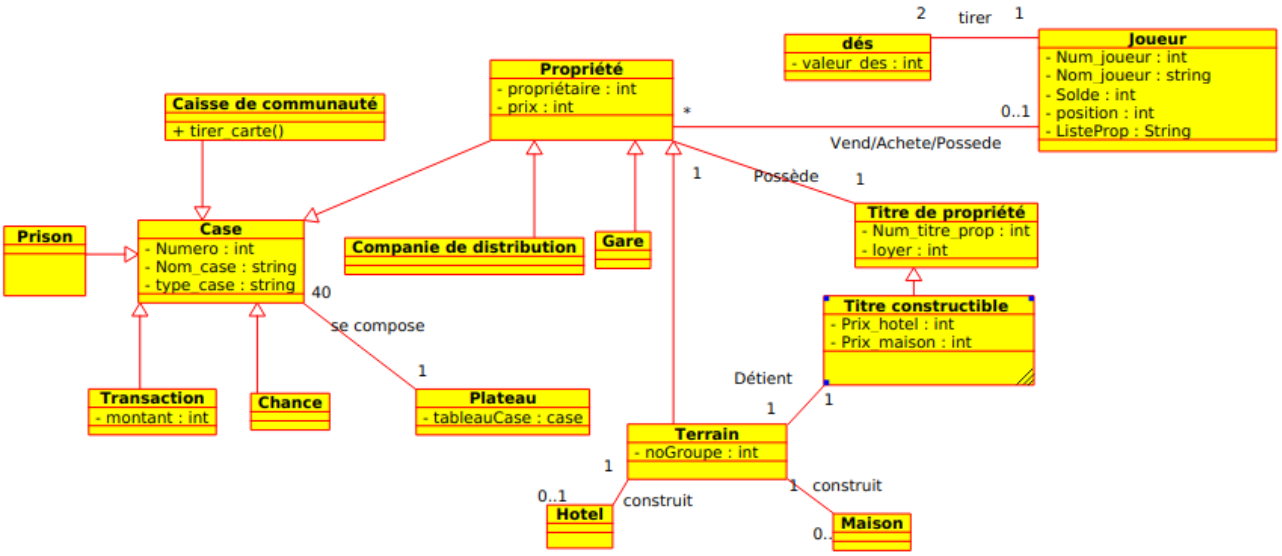
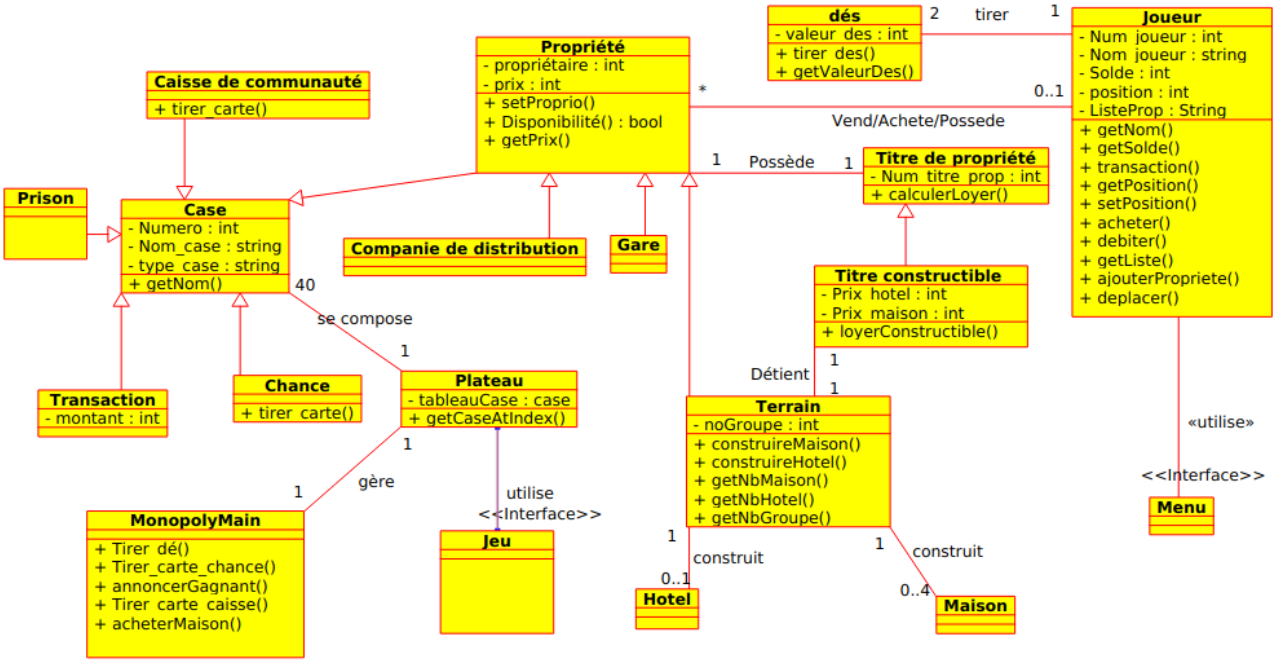
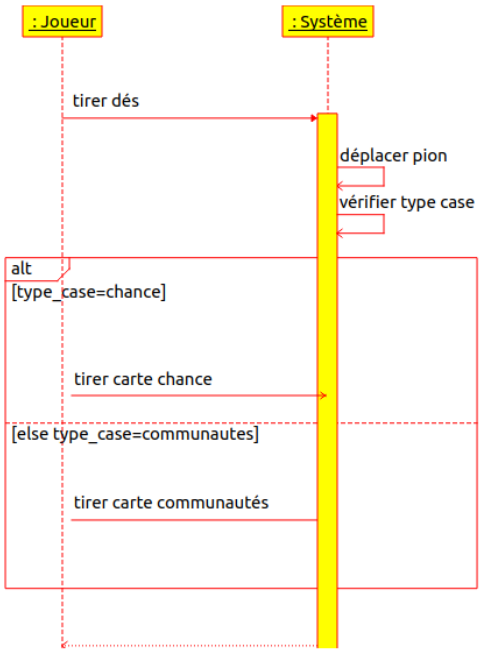
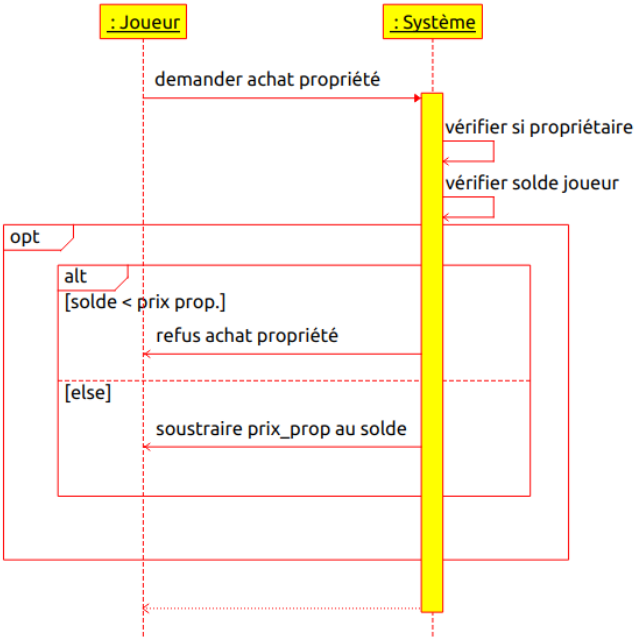


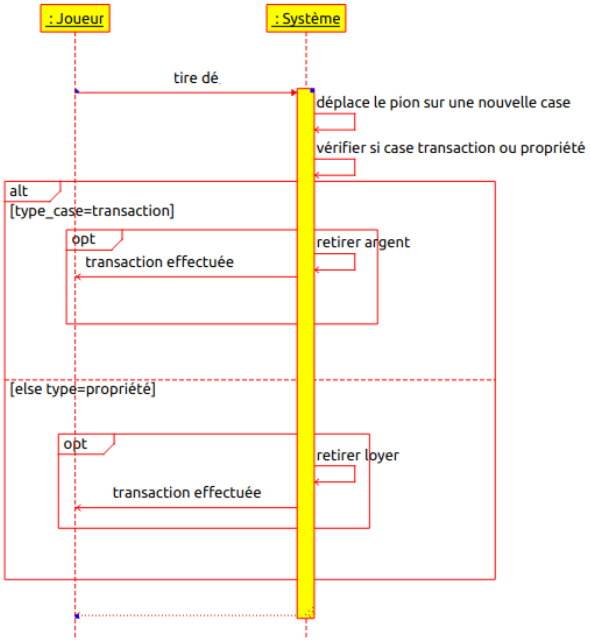
Diagramme de classe :



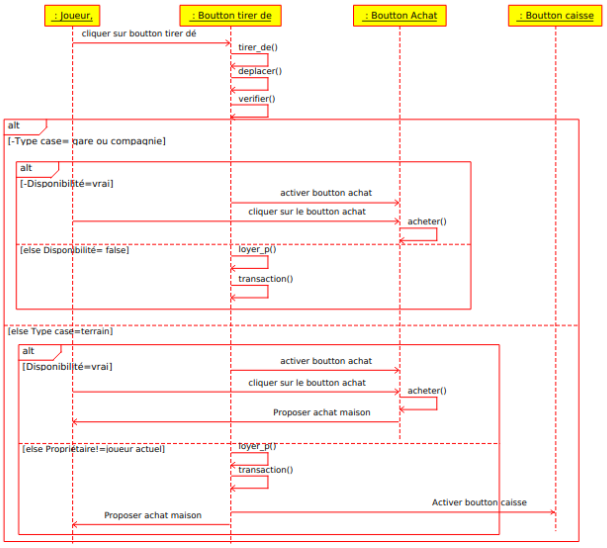
Diagrammes de séquence avec l'acteur « Systeme » :

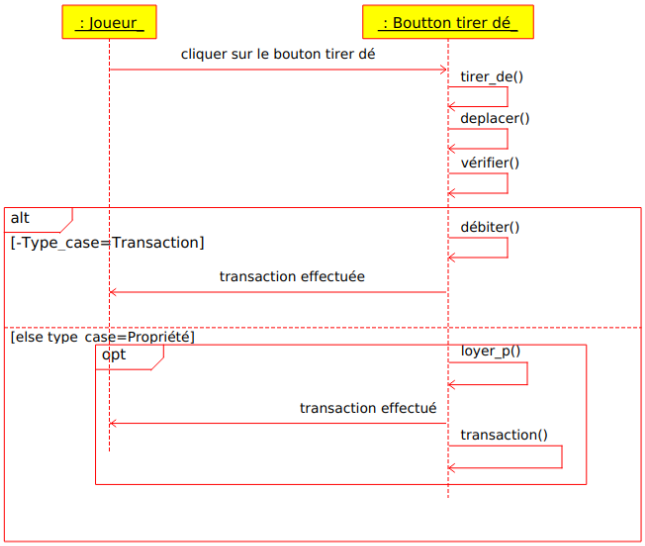


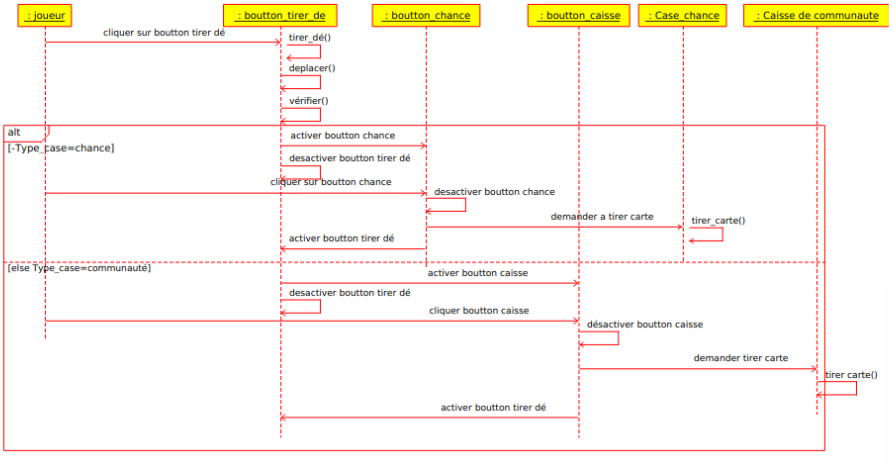




Diagrammes de séquence complets :







Plateau final :