MODELISATION DU CAS TETENGRA

Baldo Jérôme - BALJ17058609

Table des matières

I. DESCRIPTION GENERALE DU BESOIN 3

A. Présentation du concept de Tetengra 3

B. Description générale du besoin 3

1. Tour => 3

2. Bloc => 3

3. Calcul des points => 4

4. Paramétrages => 4

5. Affichage et interfaçage => 4

II. DIAGRAMME GLOBAL DE LA SOLUTION 5

A. Le diagramme sans attributs 5

B. Explicatif de son fonctionnement global 6

III. SOLUTION POUR CHAQUE BESOIN 8

1. Tour => 8

2. Bloc => 11

3. Calcul des points => 13

4. Paramétrages => 15

5. Affichage et interfaçage => 17

I. DESCRIPTION GENERALE DU BESOIN

Dans le cadre d'un projet de développement du jeux « Tetengra », l'équipe de développement et de conception demande un rapport proposant une solution pour cette application.

A. Présentation du concept de Tetengra

Tetengra est application de jeux basé sur le concept du Tetris avec l'ajout de la contrainte de la gravité. Ainsi l'objectif du joueur est d'empiler des blocs pour constituer une Tour sujette à la gravité. Selon la difficulté, la tour pourra s'effondrer. La partie s'arrête quand la tour s'effondre. Les points sont calculés selon la difficulté donnée par le joueur en début mais aussi l'avancée de la tour. La gravité va jouer un rôle dans le calcul du centre de masse mais l'équilibre de la tour.

La conception devra interagir avec une interface sous android, javaFXe et sous console Java.

B. Description générale du besoin

1. Tour =>

- La tour accumule les blocs et elle peut s'effondrer en fonction de la gravité
- Si le centre de masse d'une pièce n'est pas supporté, la pièce dégringole en effectuant une rotation
- Chaque section de la tour peut s'effondrer
- Dès que la première pièce touche le sol, le sol se transforme en lave. Une pièce qui touche la lave est désintégrée

2. Bloc =>

- La tour possède des pièces de base mais on peut en ajouter des personnalisables
- Il faut gérer le centre de masse
- Les blocs sont « partiellement » aimantées
- Une pièce peut tomber à côté de la tour
- Une pièce a une couleur précise mais peut avoir plusieurs teintes, plus elle est sombre plus elle est lourde, plus elle est claire, plus elle est légère
- Une pièce peut soit atterrir doucement, soit atterrir normalement ou soit être propulser vers le bas. La force de l'impact doit être pris en considération

3. *Calcul des points =>*

La tour vaut des points en fonction de ces calculs

- Taille de la tour
- Balancement de la tour
- Le nombre de pièces en jeu
- Le minimum de temps de vol des pièces
- La force de gravité

4. *Paramétrages =>*

- Certains paramètres peuvent être personnalisables : gravité, vitesse et tolérance

5. *Affichage et interfaçage =>*

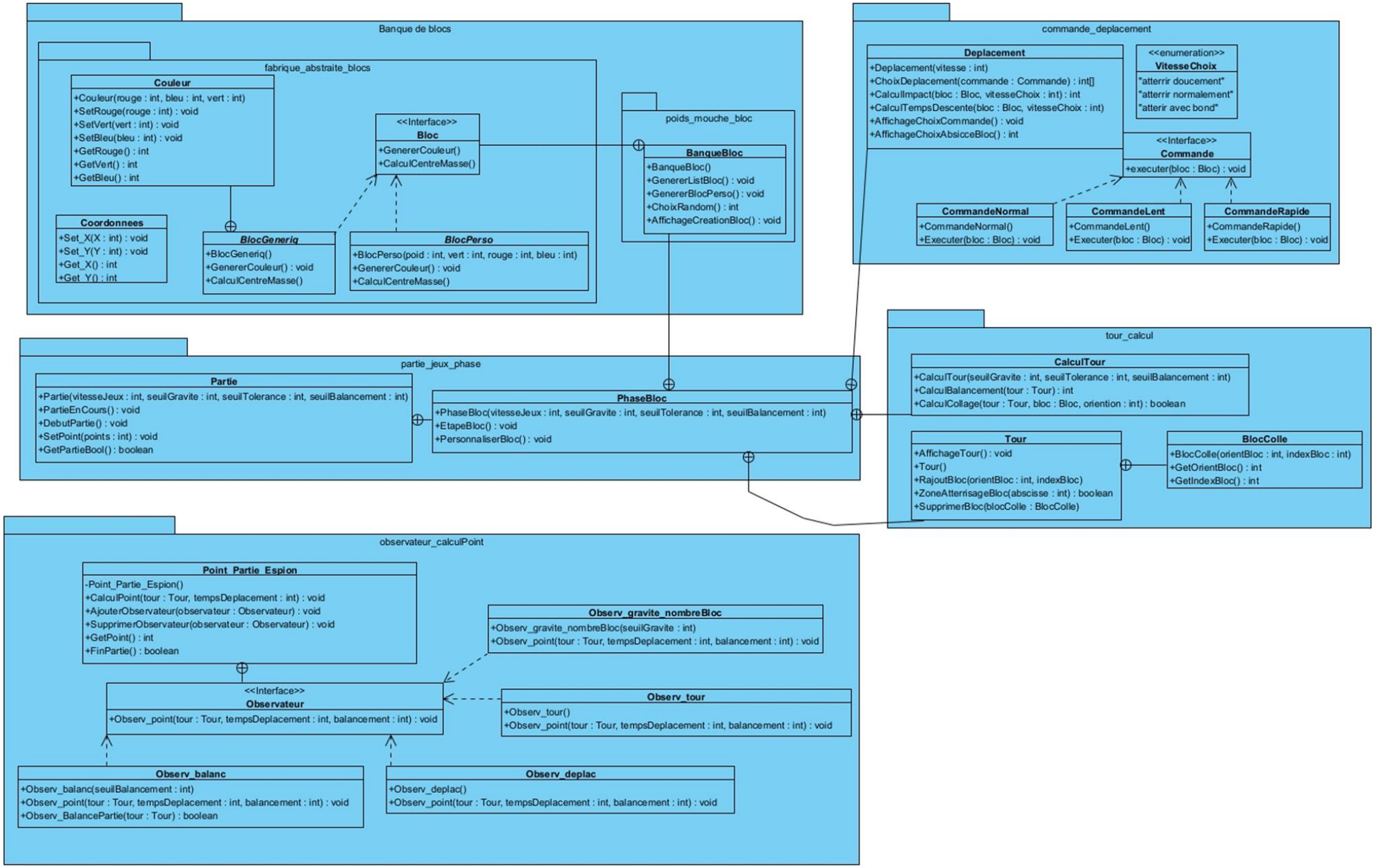
- Le modèle développé sera utilisé par une application JavaFx, un application Android et une application Java en console alors il doit être dépendant de la technologie
- La tour peut être affichée

6. *Fin de partie =>*

- La fin de partie est décrété quand le balancement généré par la tour suite à un collage d'un bloc est supérieur au seuil de balancement donné en début de partie.

II. DIAGRAMME GLOBAL DE LA SOLUTION

A. Le diagramme sans attributs



B. Explicatif de son fonctionnement global

Il y a plusieurs package permettant de séparer les fonctions :

- Package banque bloc => stockage et génération des blocs
    - Il comprend une fabrique abstraite de blocs avec un poids mouche.
    - La fabrique abstraite crée des objets de types bloc (interface bloc). Les classes abstraites Bloc_Gener et Bloc_Perso obligent le développeur à coder les blocs dans cette partie. Bloc_Gener fournit un modèle qui servira à coder en dur les blocs génériques (triangle, rectangle, ronds). Bloc_Perso permet de d'implémenter les blocs personnels. Il n'y a pas de classe classique qui permet de coder un bloc personnalisable. Cela sera à la charge du développeur de coder le comportement de ce bloc personnalisable (même s'il doit fournir le même constructeur et utiliser les méthodes)
    - Le poids mouche permet de stocker et d'instancier une seule tous les types de blocs. Il retournera seulement l'index d'un bloc au hasard (ChoixRandom).
- Package partie jeux phase => gère la partie de jeux et déclenche une phase de bloc avec à chaque fois les mêmes étapes.
    - Il comprend la classe Partie et PhaseBloc. C'est Partie qui a la charge de débuter la partie de jeux (détermination des paramètres de jeux et instanciation des différents objets comme les observateurs ou les objets servant à PhaseBloc).
    - PhaseBloc est le cœur de l'application puisqu'elle est une sorte de classe événementielle. Chaque partie comprend plusieurs phases de bloc (en gros génération d'un bloc , déplacement, calcul tour, et incrémentation des points). Elle ne se sert que d'un objet BanqueBloc, Deplacement, Tour, CalculTour et Point_Partie_Espion(singleton) . Partie va avoir une méthode PartieEnCours qui sera une boucle appelant à chaque fois une phase de bloc (PhaseBloc.EtapeBloc) jusqu'à que le Balancement soit supérieur au seuil de balancement donnée en début de partie.
- Package déplacement => gère uniquement le déplacement
    - Déplacement est constitué du patron commande.
    - Elle est déclenché par PhaseBloc qui va utiliser AffichageChoixCommande et AffichageChoixAbsicceBloc pour le déplacement de la pièce.
    - Les commandes sont prédéfini => doucement / normalement / avec bond et déclenchent une commande. Selon la commande, il y aura un calcul différent pour le temps de vol et la force d'impact.
- Package Calcul tour => implémentation des blocs sur la tour ainsi que les calculs pour le collage du bloc et du balancement de la tour (tombe selon les sections)
- Package Observateur point partie => patron de conception observateur couplé avec un singleton stockant l'incrémentation des points et permettant à la partie de consulter les points (pour mise à jour) et consultation du balancement pour déterminer la fin de partie.
- Chaque observateur sera déclenché par le singleton dans la phaseBloc pour calcul et incrémentation des points.
- Le nombre de points stockés dans le singleton sera consulter par Partie après chaque appel de EtapeBloc de PhaseBloc ainsi que le balancement.

La couleur est définie selon le code couleur RVB en integer

La dimension d'un bloc est un ensemble de points. Chaque point est constitué d'une coordonnée X et y en Integer. Même concept pour le centre de masse.

EtapeBloc comprend donc les étapes suivantes =>

1. Génération d'un bloc avec ChoixRandom()
2. Génération de l'orientation du bloc au hasard
3. Déplacement choix de la commande
4. Déplacement retour temps de vol et force d'impact
5. Déplacement choix de l'abscisse du bloc et calcul avec zone d'atterrissage de la tour pour déterminer si le bloc tombe dans la lave ou sur la tour
6. Si sur la tour alors détermination de son atterrisage (impact,
7. Si pas collage alors changement de rotation et tombe dans la lave.
8. Si collage alors détermination du balancement de la tour et si tombe des pièces + retourne balancement
9. Déclenchement des calculs de points pour avec les données retournées (balancement, tour, temps de vol)

PartieEnCours comprend donc les étapes suivantes =>

1. Appel EtapeBloc
2. SetPoint => récupérer les points du singleton et mettre à jour les points de son compteur
3. GetPartieBool => appel de la méthode Observ_balance_Partie pour savoir si le balancement est supérieur au seuilBalancement dans ce cas fin de partie.

ATTENTION => je n'ai pas pu mettre le lien d'inclusion entre coordonnées avec Bloc_Gener et Bloc_Perso

III. SOLUTION POUR CHAQUE BESOIN

1. Tour =>

LES BESOINS =>

- Chaque section de la tour peut s'effondrer
- Dès que la première pièce touche le sol, le sol se transforme en lave. Une pièce qui touche la lave est désintégrée
- La tour accumule les blocs et elle peut s'effondrer en fonction de la gravité
- Si le centre de masse d'une pièce n'est pas supporté, la pièce dégringole en effectuant une rotation
- Une pièce peut tomber à côté de la tour
- Les blocs sont « partiellement » aimantées

LE TEXTE EXPLICATIF =>

- Chaque section de la tour peut s'effondrer. Lorsque l'atterrissage se fait sur la tour, la méthode EtapeBloc appelle CalculTour et sa méthode CalculBalancement. Cette méthode va calculer, bloc par bloc et section par section, le balancement de la tour en prenant en compte la gravité et le seuil de tolérance (aimanté). Le calcul prend également en compte les dimensions, le poids et le centre de masse de chaque bloc. À la manière d'un arbre AVL, il vérifie si une section est équilibrée ou non et localise le bloc ou la section ne répondant pas à ce critère. S'il y a un déséquilibre, les blocs ou les sections sont supprimés de la liste de la tour.
- Dès que la première pièce touche le sol, le sol se transforme en lave. Une pièce qui touche la lave est désintégrée. Le premier bloc qui touche le sol déclenche la mise en place du sol de lave, qui est vérifiée par la méthode EtapeBloc de la classe PhaseBloc. Si le bloc est au niveau 0 de la tour, il est supprimé et ne compte pas de points.
- La tour accumule les blocs, et elle peut s'effondrer en fonction de la gravité. À chaque atterrissage sur la tour d'un bloc, CalculTour est appelé pour déterminer si la tour va s'effondrer selon la gravité.
- Si le centre de masse d'une pièce n'est pas supporté, la pièce dégringole en effectuant une rotation. Si un bloc ne tient pas sur la tour, la méthode CalculCollage de l'objet CalculTour est appelée pour déterminer si le bloc va tenir sur la tour. Si le bloc tient, PhaseBloc appelle la méthode RajoutBloc() de la tour pour ajouter le bloc à sa liste. Sinon, la méthode de déplacement est appelée à nouveau pour un nouvel essai.
- Une pièce peut tomber à côté de la tour. Lorsque la méthode EtapeBloc est appelée, elle demande à l'utilisateur de choisir où le bloc doit tomber en appelant la méthode AffichageChoixAbscicceBloc. Avec cette donnée, PhaseBloc appelle la méthode ZoneAtterrisageBloc() de la tour pour déterminer si l'abscisse choisie permet au bloc de tomber sur la tour grâce à sa dimension. Si c'est le cas, la méthode renvoie True. Sinon, cela signifie que le bloc va tomber à côté de la tour et tomber dans la lave. Dans ce cas, EtapeBloc s'arrête et on passe à un prochain bloc.
- Quand un bloc atterrit sur la Tour, PhaseBloc appelle la méthode CalculCollage et CalculBalancement de l'objet CalculTour. CalculCollage va prendre en entrée la Tour ainsi que le bloc (son index), son orientation ainsi que la force d'impact pour calculer si le bloc va coller

à la Tour. Ensuite CalculBalancement va calculer si les blocs vont tenir. Pour cela il va prendre en compte la Tour et calculer bloc par bloc et section par section, s'il y a adhérence en ce basant sur le seuil de gravité, le seuil de tolérance et le seuil de balancement. Il déterminera s'il y a un ou plusieurs blocs qui tombent et dans ce cas ils les supprimera de la liste de bloc de la tour en appelant la méthode de SuprimerBloc(en utilisant une boucle). De plus CalculBalancement() détermine le taux de balancement de la Tour et retourne le taux de Balancement sous forme de int.

DIAGRAMME STATIQUE =>

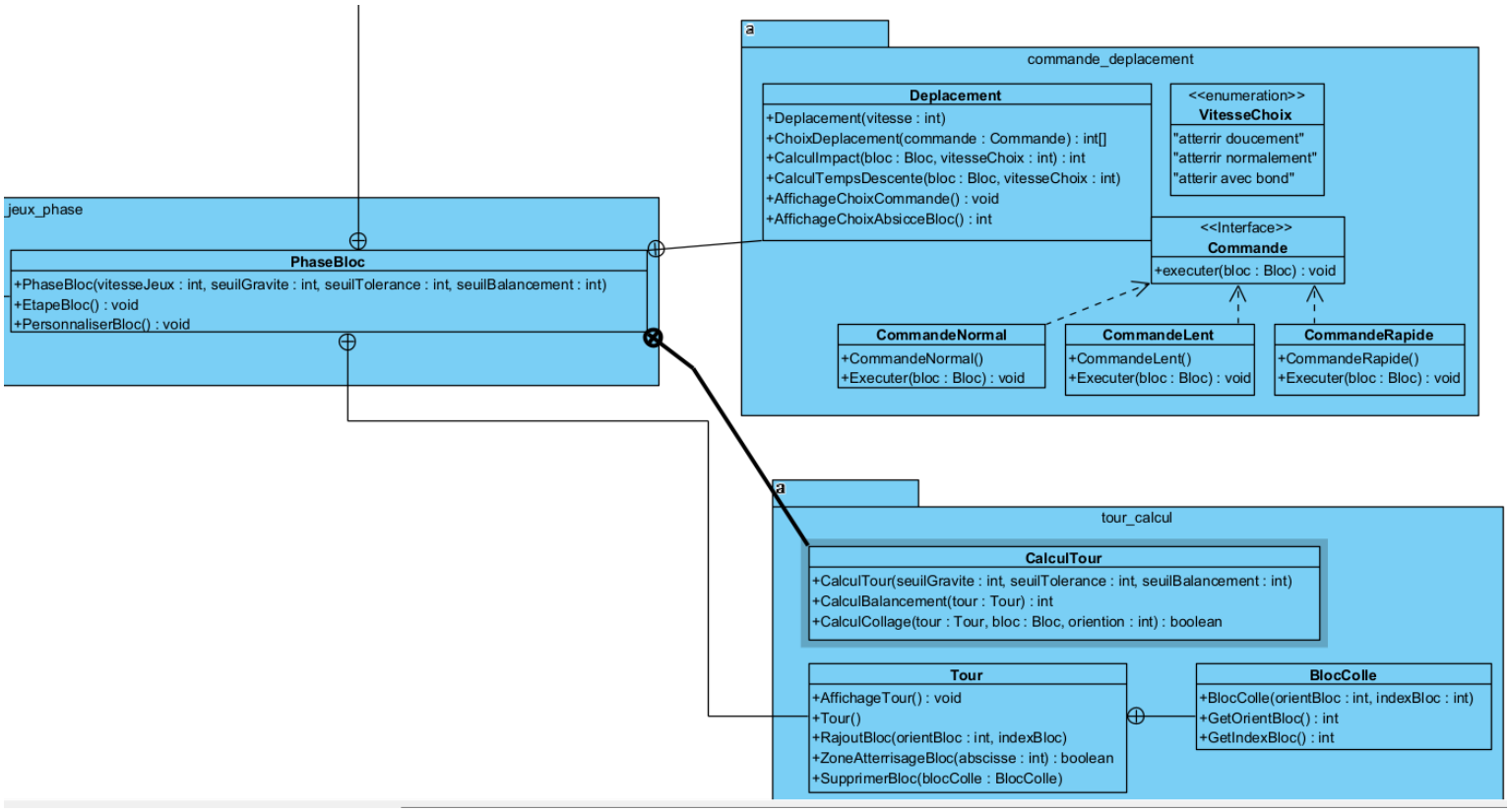
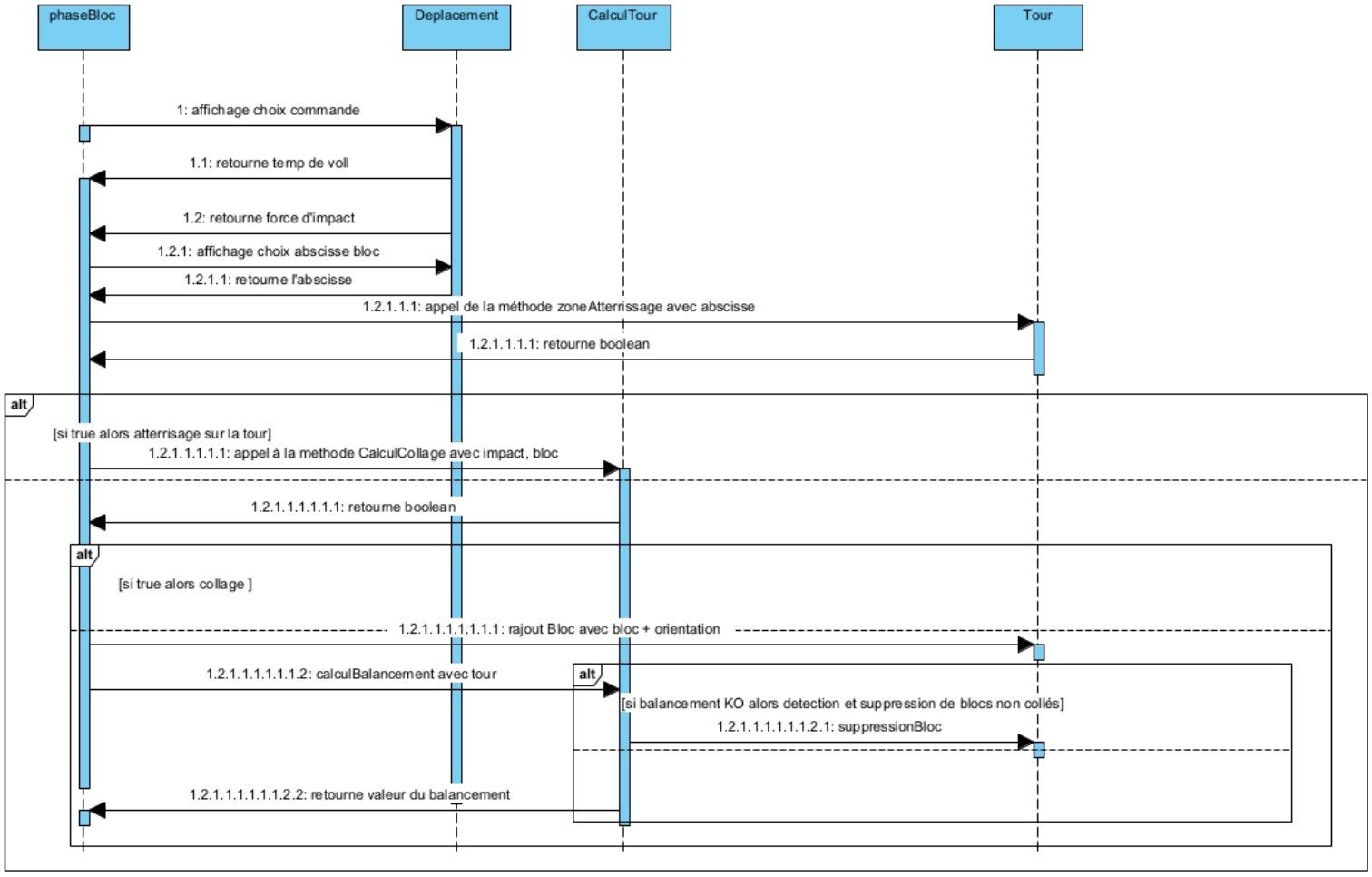


DIAGRAMME DYNAMIQUE =>



2. *Bloc =>*

LES BESOINS =>

- Une pièce a une couleur précise mais peut avoir plusieurs teintes, plus elle est sombre plus elle est lourde, plus elle est claire, plus elle est légère
- Une pièce peut soit atterrir doucement, soit atterrir normalement ou soit être propulser vers le bas. La force de l'impact doit être pris en considération
- La tour possède des pièces de base mais on peut en ajouter des personnalisables
- Il faut gérer le centre de masse

LE TEXTE EXPLICATIF =>

- Lors de la création des pièces et de leur stockage dans la liste de blocs, la méthode "GénérerCouleur" est utilisée pour modifier la couleur de départ. Cette méthode jouera sur la teinte de la couleur en faisant varier le code RVB du bloc. Cette méthode est active pour les blocs génériques ainsi que les blocs personnalisés. Cela enlève cette tâche au développeur pour l'élaboration des blocs génériques (mais pas pour les blocs personnalisables qui sont générés en début de partie).
- "PhaseBloc" utilise l'objet "Deplacement" pour déplacer le bloc qui vient d'être généré. Il déclenche la méthode "afficherChoixDeplacement", qui à son tour déclenche une commande : "lent", "normal", ou "avec bond". Cette commande déclenche le calcul de l'impact et du temps de vol. Ces deux méthodes retournent les résultats respectifs dans la méthode "EtapBloc", qui recueille les résultats pour le calcul du collage et des points.
- Le package "BanqueBloc" comprend une fabrique abstraite de blocs et un poids mouche stockant ces blocs. La fabrique abstraite comprend les blocs génériques et les blocs personnalisables. Les blocs génériques sont déterminés à l'avance en s'appuyant sur la classe abstraite "Bloc_Gener". Ils sont instanciés et stockés dans la liste de blocs. Ils sont créés automatiquement par la méthode "GenererBlocPerso()", appelée à partir du constructeur "PhaseBloc".
- Les blocs personnalisables sont créés dans une boucle dans la méthode "PersonnaliserBloc()" de "PhaseBloc". "PersonnaliserBloc()" est déclenché par la méthode "DebutPartie()" de "Partie" (en début de partie, lors du paramétrage). "PersonnaliserBloc()" appelle la méthode "AffichageCreationBloc()" de "BanqueBloc". Cette classe affiche elle-même son interface pour personnaliser les blocs.
- Pour les blocs génériques, les dimensions, la couleur et le poids sont codés en dur. Le centre de masse et la couleur finale (teinte selon le poids) sont déterminés grâce aux méthodes "GenererCouleur()" et "CalculCentreMasse()" dans le constructeur de l'objet. Pour les blocs personnalisables, les dimensions, la couleur et le poids sont donnés par l'utilisateur par l'intermédiaire de la méthode "AffichageCreationBloc()". Le centre de masse et la couleur finale (teinte selon le poids) sont déterminés grâce aux méthodes "GenererCouleur()" et "CalculCentreMasse()"

DIAGRAMME STATIQUE =>

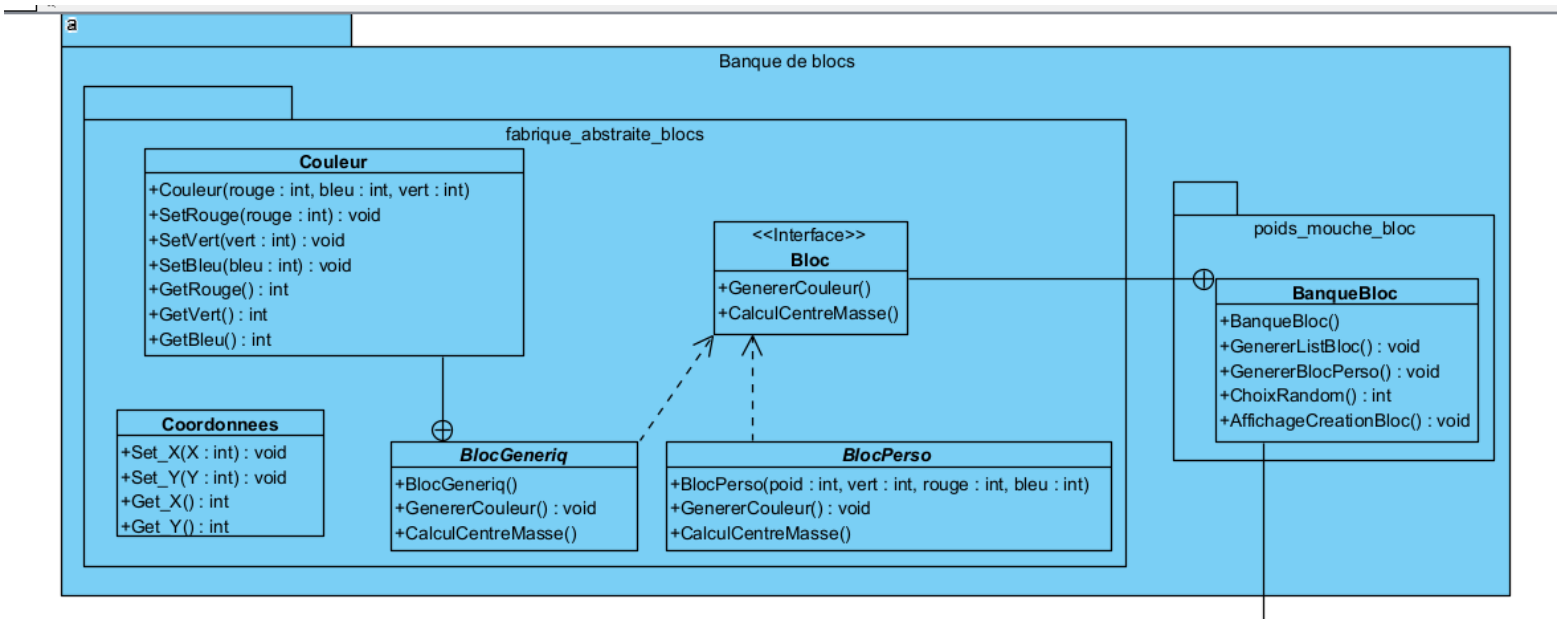
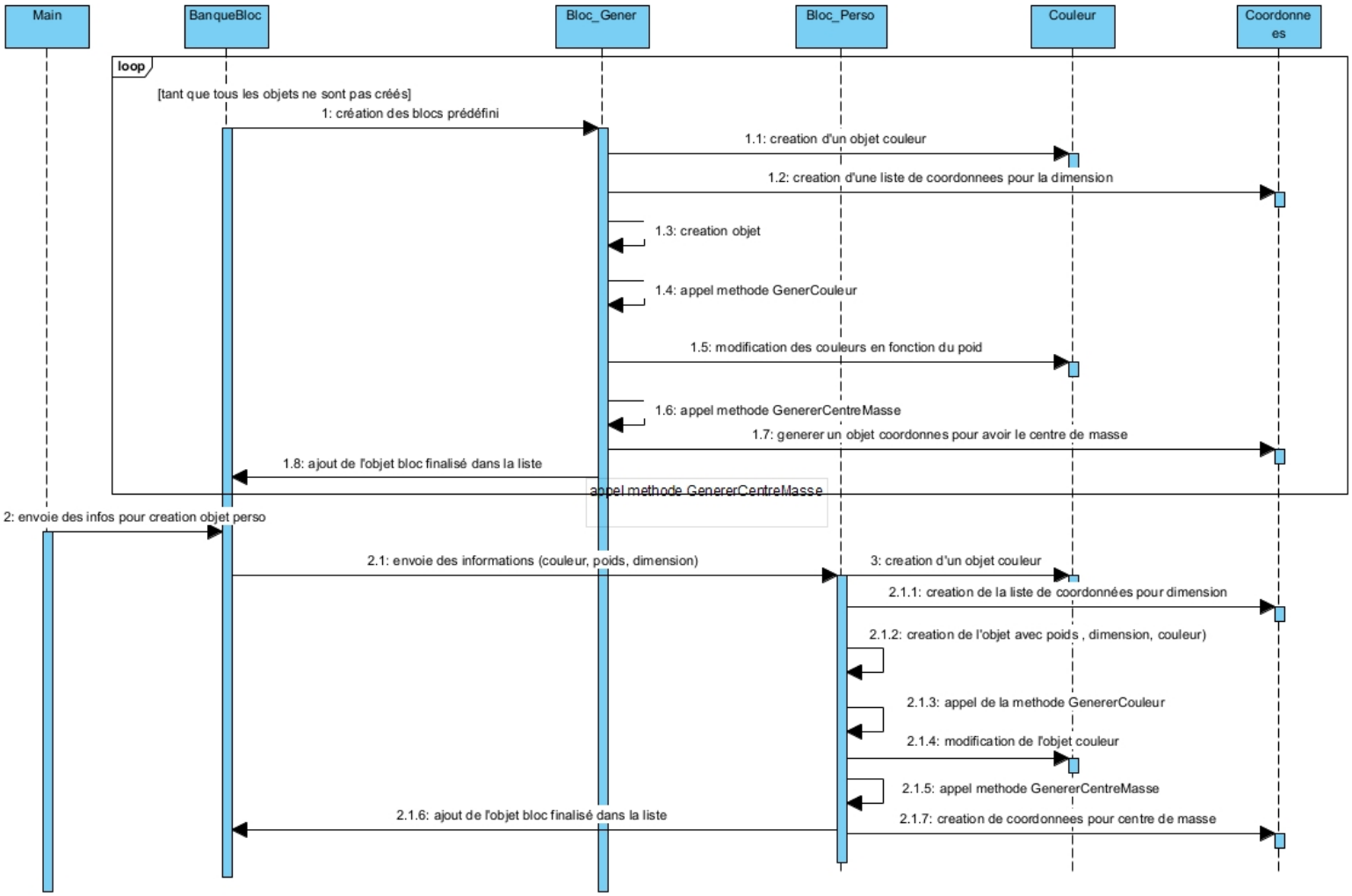


DIAGRAMME DYNAMIQUE =>



3. *Calcul des points =>*

LES BESOINS =>

La tour vaut des points en fonction de ces calculs

- Taille de la tour
- Balancement de la tour
- Le nombre de pièces en jeu
- Le minimum de temps de vol des pièces
- La force de gravité

LE TEXTE EXPLICATIF =>

A chaque appel de l'étape bloc de la classe phaseBloc, le singleton espion point est appelé avec la méthode calculPoint. Cette méthode prend en entrée les calculs du temps de vol de la pièce, la tour après incrémentation de la pièce ainsi que le balancement de la tour.

CalculPoint qui est une boucle sur tous les observateurs mis dans la liste observateur en attribut du singleton. Les paramètres d'entrée ne sont pas tous utilisés à chacune des méthodes, mais permettent une généralisation. Cet appel est effectué dans la méthode EtapeBloc() de PhaseBloc. Elle se produit à la fin de la méthode en injectant les données utiles pour le calcul des points (temps de vol, tour, balancement). CalculPoint déclenche la méthode Observ_point pour chaque observateur. Ceux-ci incrémentent le compteur de point du singleton Point_Partie_Espion.

Pour chaque boucle dans PartieEnCours dans l'objet Partie, Partie appelle la méthode SetPoint qui à son tour appelle la méthode GetPoint du singleton pour mettre à jour son compteur de points global.

Les observateurs =>

- Taille de la tour : A chaque déclenchement de la méthode Observ_Point de l'objet Observ_tour, celle-ci prend en entrée la tour et récupère la hauteur de la tour pour calculer le nombre de points. Elle incrémente ensuite le compteur de points du singleton. Par ailleurs elle conserve le balancement en cours dans l'attribut balanceEnCours qui servira pour savoir si la partie se finit. A chaque déclenchement de la méthode, cette attribut est réinitialisé avec la nouvelle valeur.
- Calcul des points selon le balancement : A chaque déclenchement de la méthode Observ_Point de l'objet Observ_balanc, celle-ci prend le taux de balancement reçu de la tour et calcule le nombre de points qui sera incrémenté dans le compteur de points du singleton.
- Le nombre de pièces en jeu et la force de gravité : A chaque déclenchement de l'observateur Observ_gravite_nombreBloc et de sa méthode Observ_Point(), il incrémente un compteur de nombre de bloc en jeu et calcule les points selon le nombre de bloc passé et la gravité.
- Le minimum de temps de vol des pièces : A chaque déclenchement de l'observateur Observ_deplac et de sa méthode Observ_Point(), il calcule les points selon le temps de vol. Il incrémente ensuite le compteur de points du singleton.
- La gravité et le nombre de pièces : le nombre de pièces en jeu est incrémenté à chaque appel de la méthode Observ_point de la classe Observ_gravite_nombreBloc. Avec le seuil de gravité et le nombre de bloc en jeux , il va calculer le nombre de points et l'incrémenter dans le compteur du singleton.

DIAGRAMME STATIQUE =>

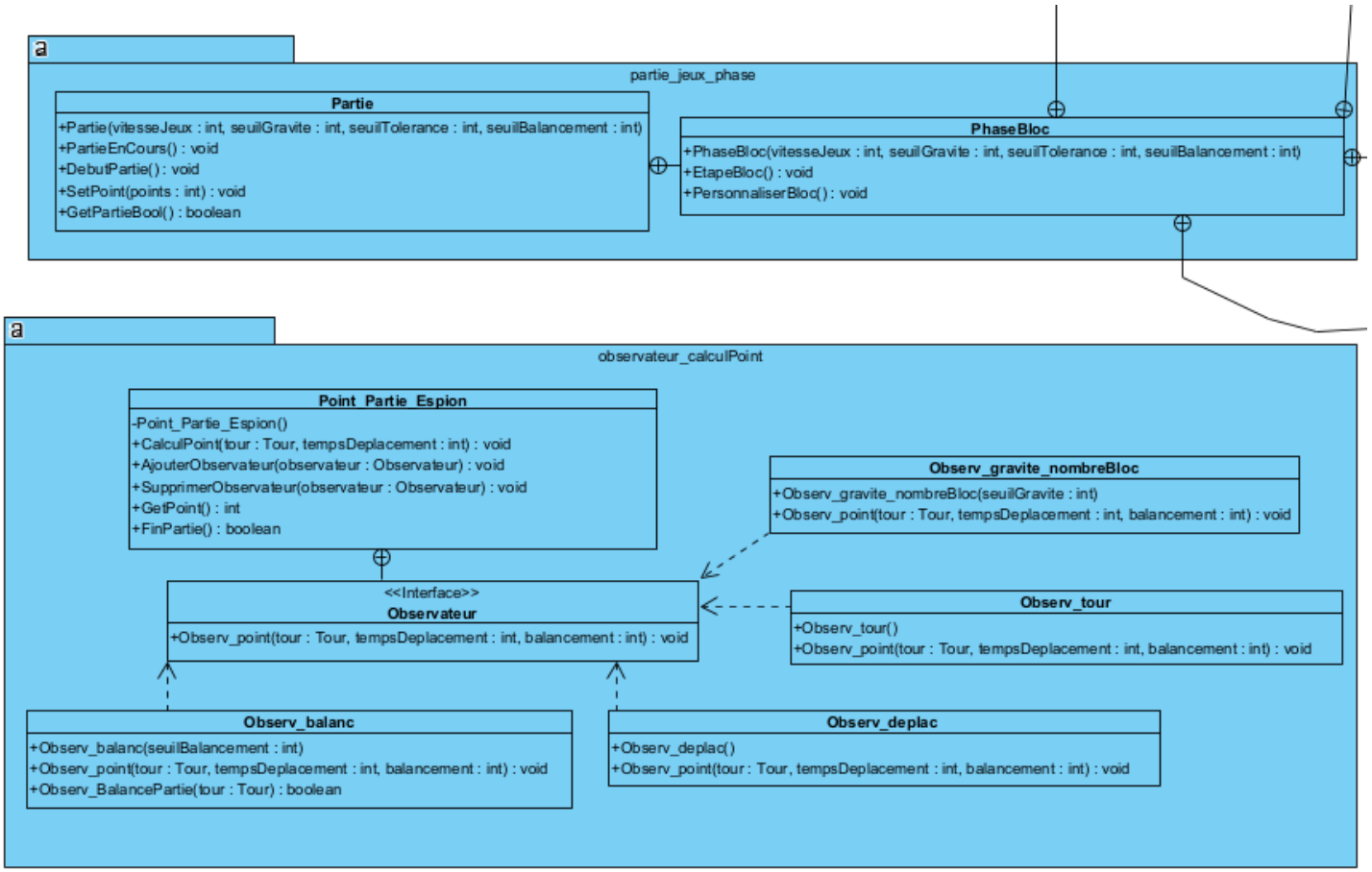
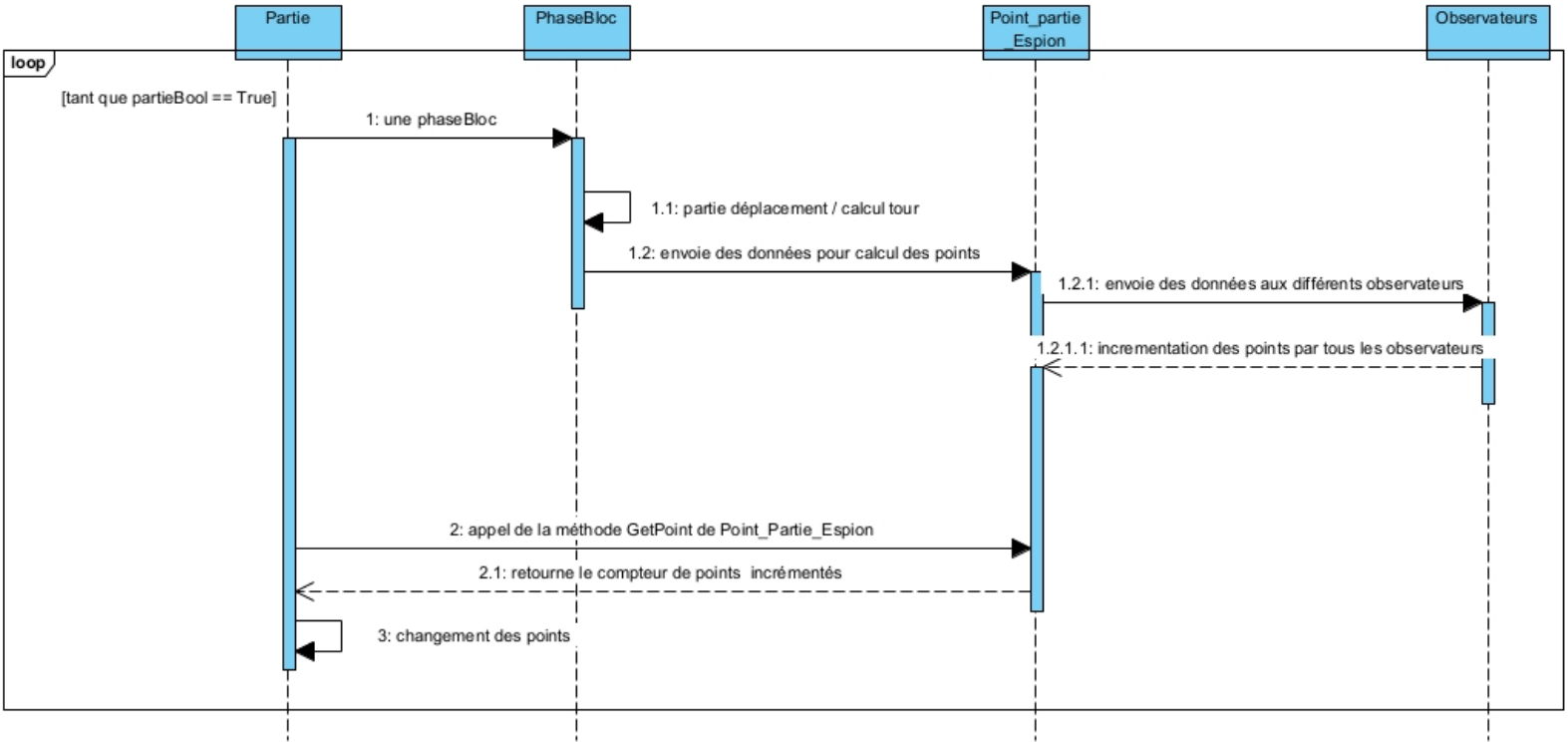


DIAGRAMME DYNAMIQUE =>



4. *Paramétrages =>*

LES BESOINS =>

- Certains paramètres peuvent être personnalisables : gravité, vitesse et tolérance
- Il y a un seuil de tolérance

LE TEXTE EXPLICATIF =>

Certains paramètres peuvent être personnalisables, tels que la gravité, la vitesse et la tolérance. La tolérance est divisée en deux facettes, la première étant la tolérance du balancement instancié par le seuilBalancement, qui est utilisé pour le calcul des points, la fin de partie et le CalculTour.

Au début de la partie, l'objet Partie récupère la gravité, la vitesse de jeu et la tolérance pour le collage et le balancement. Ces paramètres sont transmis à l'objet PhaseBloc qui crée les objets Deplacement, CalculTour et BanqueBloc pour faire fonctionner le jeu. Les paramètres sont également utilisés pour le calcul des points et l'objet Partie instancie les objets Observ_gravite et Observ_balanc.

DIAGRAMME STATIQUE =>

La suite sur la page suivante

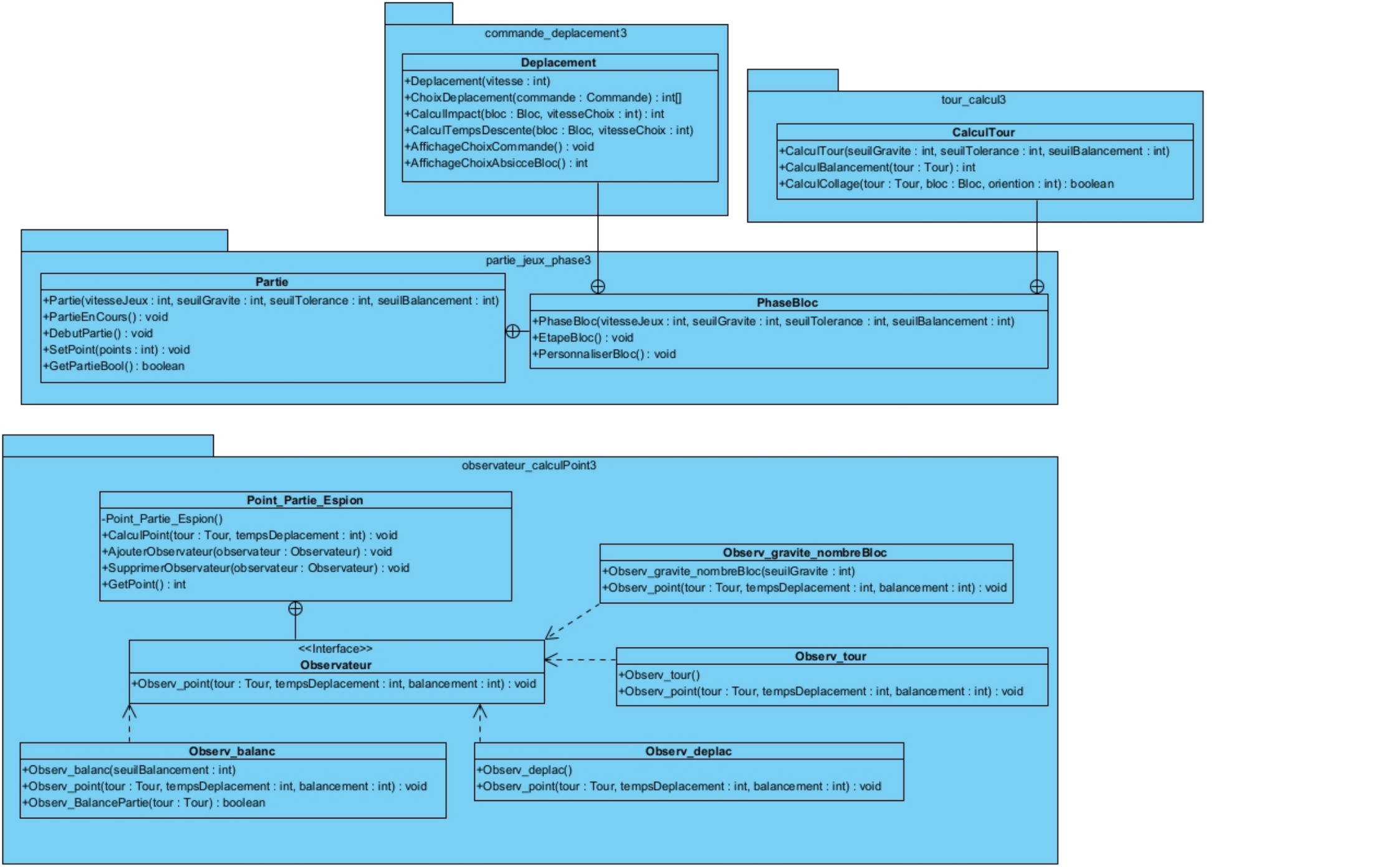
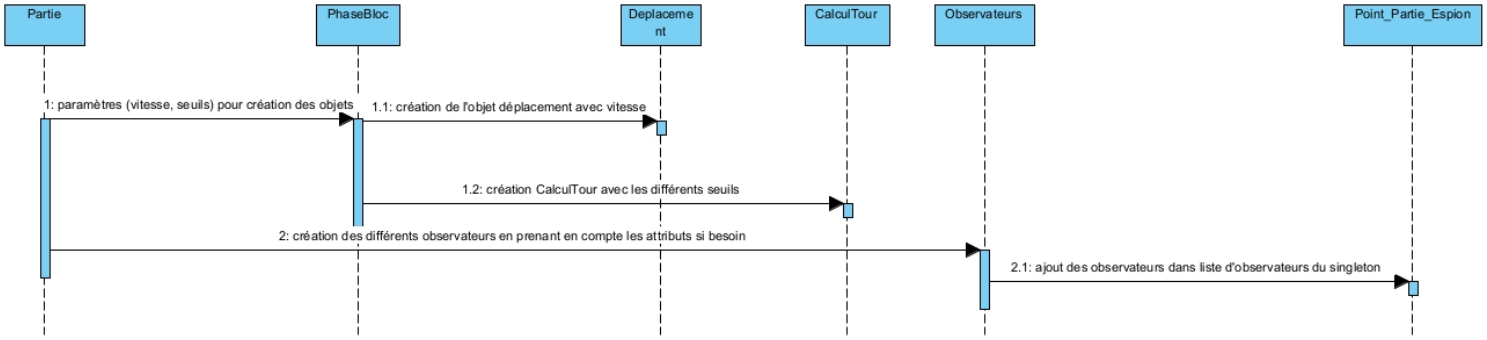

commande_deplacement3
Deplacement
+Deplacement(vitesse : int)
+ChoixDeplacement(commande : Commande) : int[]
+CalculImpact(bloc : Bloc, vitesseChoix : int) : int
+CalculTempsDescente(bloc : Bloc, vitesseChoix : int)
+AffichageChoixCommande() : void
+AffichageChoixAbsicceBloc() : int
tour_calcul3
CalculTour
+CalculTour(seuilGravite : int, seuilTolerance : int, seuilBalancement : int)
+CalculBalancement(tour : Tour) : int
+CalculCollage(tour : Tour, bloc : Bloc, orientation : int) : boolean
partie_jeux_phase3
Partie
+Partie(vitesseJeux : int, seuilGravite : int, seuilTolerance : int, seuilBalancement : int)
+PartieEnCours() : void
+DebutPartie() : void
+SetPoint(points : int) : void
+GetPartieBool() : boolean
PhaseBloc
+PhaseBloc(vitesseJeux : int, seuilGravite : int, seuilTolerance : int, seuilBalancement : int)
+EtapeBloc() : void
+PersonnaliserBloc() : void
observateur_calculPoint3
Point_Partie_Espion
-Point_Partie_Espion()
+CalculPoint(tour : Tour, tempsDeplacement : int) : void
+AjouterObservateur(observateur : Observateur) : void
+SupprimerObservateur(observateur : Observateur) : void
+GetPoint() : int
Observ_gravite_nombreBloc
+Observ_gravite_nombreBloc(seuilGravite : int)
+Observ_point(tour : Tour, tempsDeplacement : int, balancement : int) : void
<<Interface>>
Observateur
+Observ_point(tour : Tour, tempsDeplacement : int, balancement : int) : void
Observ_tour
+Observ_tour()
+Observ_point(tour : Tour, tempsDeplacement : int, balancement : int) : void
Observ_balanc
+Observ_balanc(seuilBalancement : int)
+Observ_point(tour : Tour, tempsDeplacement : int, balancement : int) : void
+Observ_BalancePartie(tour : Tour) : boolean
Observ_deplac
+Observ_deplac()
+Observ_point(tour : Tour, tempsDeplacement : int, balancement : int) : void

DIAGRAMME DYNAMIQUE =>



5. Affichage et interfaçage =>

LES BESOINS =>

- Le modèle développé sera utilisé par une application JavaFx, un application Android et une application Java en console alors il doit être dépendant de la technologie
- La tour peut être affichée

LE TEXTE EXPLICATIF =>

Les méthodes d'affichage sont propres aux différentes classes tel que Deplacement, BanqueBloc, Tour et Partie. À chaque fin de la méthode etapeBloc, phaseBloc appelle la méthode d'affichage de la classe Tour, qui itère sur la liste de blocs et transmet les informations suivantes au Main de l'application : l'index de chaque bloc, son orientation, son poids, sa couleur, son centre de masse (coordonnées) ainsi que ses dimensions (liste de points) et son code couleur (RVB).

DIAGRAMME STATIQUE =>

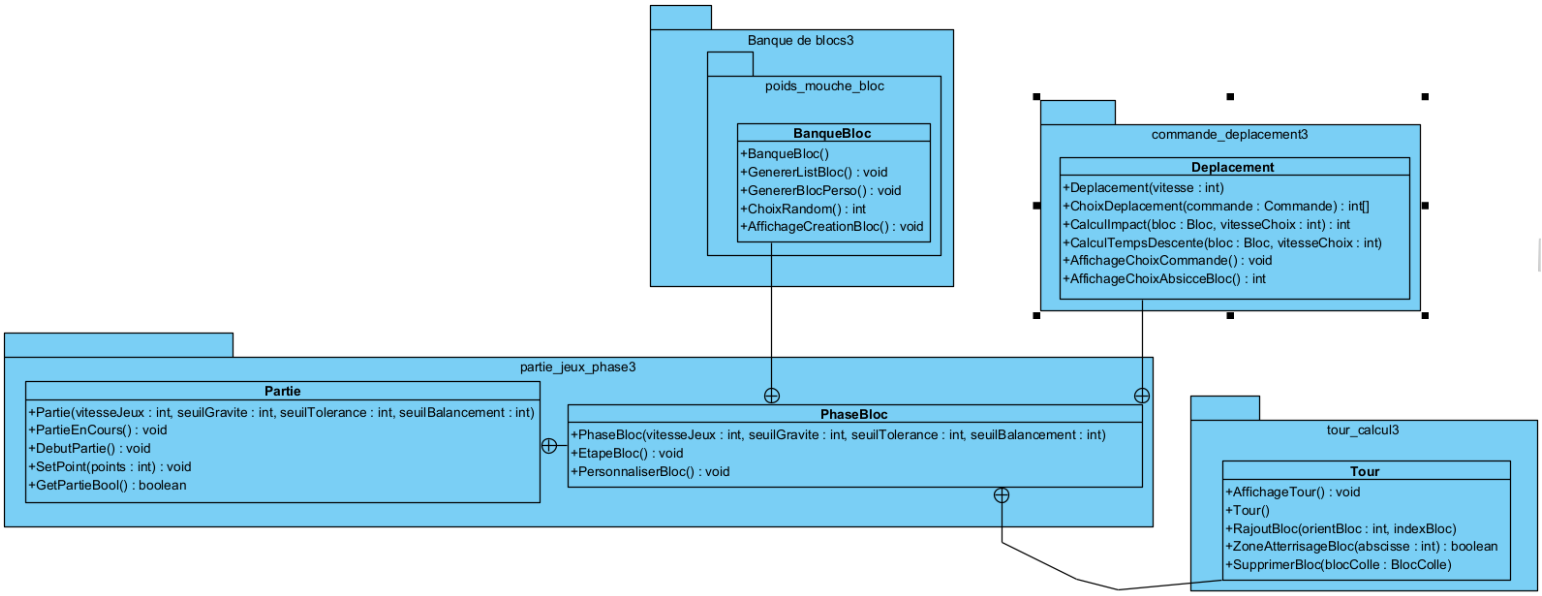
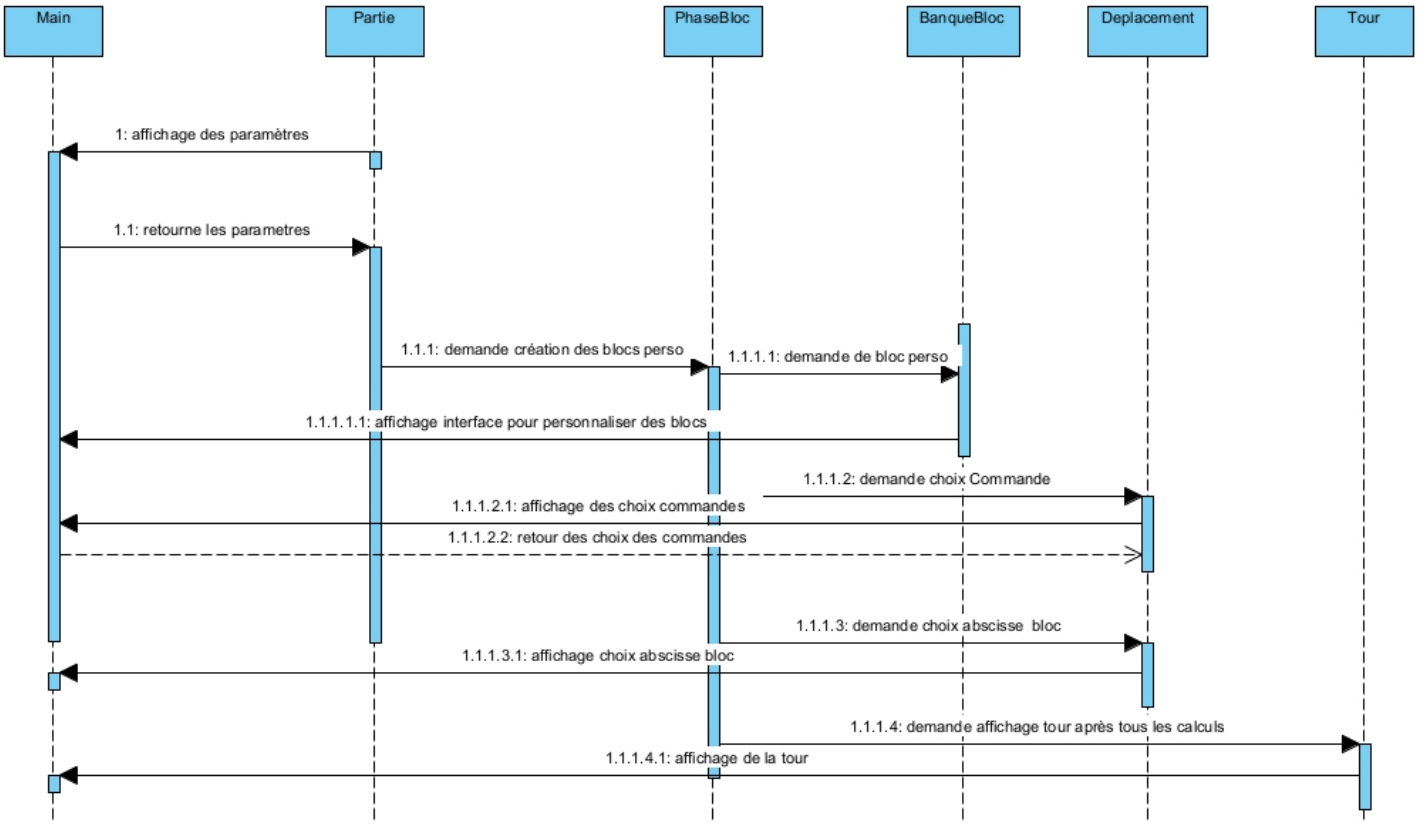


DIAGRAMME DYNAMIQUE =>



6. *Fin de partie =>*

LE BESOIN =>

La partie finit quand le balancement de la tour est supérieur au seuil de balancement donné en début de partie.

TEXTE EXPLICATIF =>

Quand l'objet a fini de récupérer le nombre de points et à réaliser la mise à jour, Partie appel dans sa méthode PartieEnCours() la méthode FinPartie() du singleton Partie_Jeux_Espion. Cette méthode déclenche la méthode Observ_BalancePartiede l'observateur Observ_Balancement. En comparant le seuilBalancement donné en début de partie, et le balancement stocké dans l'attribut balanceEnCours, la méthode renvoit un true ou false à Partie. Si true, la boucle dans Partie continue sinon elle s'arrete et le jeux termine.

DIAGRAMME STATIQUE =>

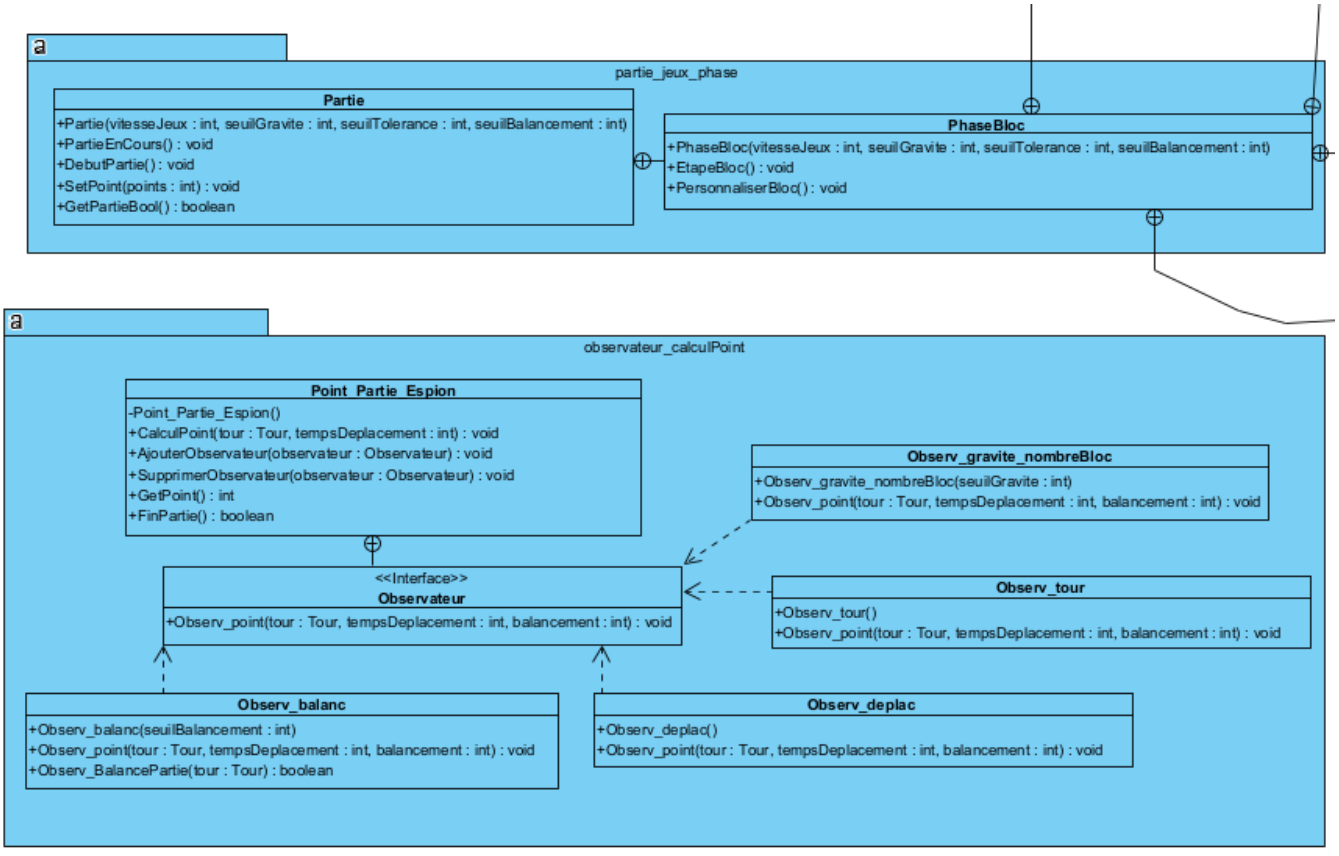
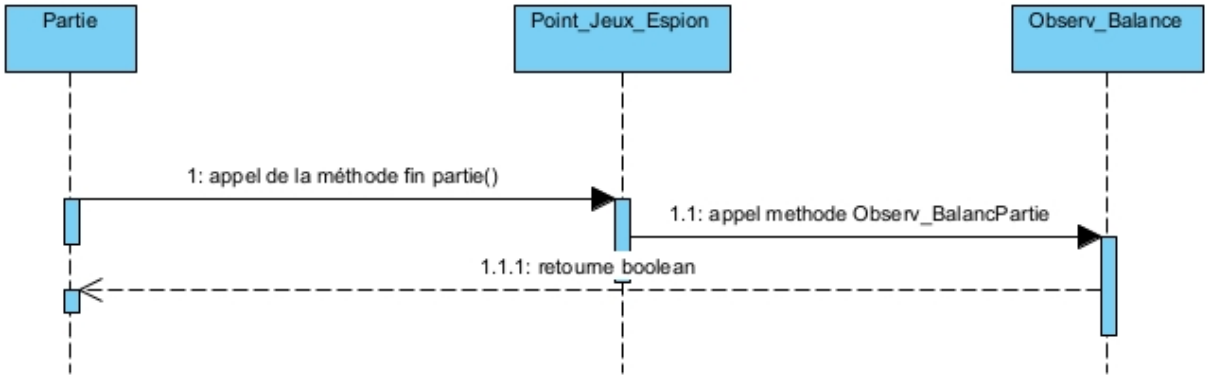


DIAGRAMME DYNAMIQUE =>


Partie
Point_Jeux_Espion
Observ_Balance
1: appel de la méthode fin partie()
1.1: appel methode Observ_BalancPartie
1.1.1: retourne boolean