

2025–2026

L3 INFORMATIQUE

CAPACI

PROGRAMMATION WEB

Rapport de l'étape 1

ÉQUIPE DE DÉVELOPPEMENT

Thomas BOUDEELE

Gr A

Louis HAGUES

Gr A

Killian REINE

Gr A

• Mise à jour : 18 janvier 2026 •

1 Introduction

1.1 Quelques mots pour commencer

Le jeu Capaci, de la catégorie des « jeux de plateau », est un jeu de stratégie se jouant à deux. Son principe est fondé sur le mélange de deux autres jeux bien connus de tous :

- Les échecs
- Le Chi-Fou-Mi (Pierre Papier Ciseau)

Sur un plateau 6×6 , initialement chaque joueur possède 12 pièces de trois catégories différentes : 4 pierres, 4 feuilles et 4 ciseaux qui sont disposées de la manière suivante. Les 12 cases centrales sont initialement vides pour permettre le déplacement.

On considère que le premier joueur qui jouera en premier est celui qui a créé la partie.

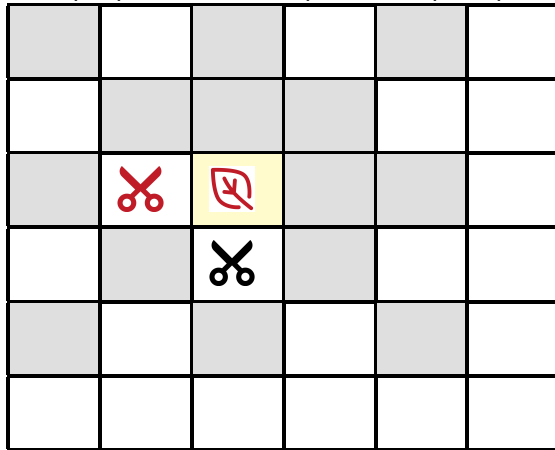
1.2 Règles du jeu

1.2.1 Les déplacements des pièces

Les différents déplacements des pièces du jeu suivent les principes suivants :

- Toutes les pièces se déplacent exactement de la même manière. Celle-ci est basée sur le déplacement de la reine aux échecs. C'est à dire que chaque pièce peut se déplacer à gauche, à droite, en haut et en bas mais aussi sur les diagonales.
- La portée de déplacement d'une pièce est limitée par le nombre de pièces adjacentes (excluant les diagonales). La pièce a une portée de déplacement de $i \in [1; 4]$ où i correspond au nombre de pièces amis et adverses se trouvant autour de celle-ci **excluant les diagonales**.
- Une pièce peut passer « au dessus » d'autres pièces peu importe à qui elles appartiennent dans le cas où la portée est assez grande pour le permettre et si elle peut se positionner sur la case souhaitée.

Voici un exemple pour mieux comprendre le principe du déplacement



Considérons une pièce située au centre de la grille (*nous ne prenons pas en compte les autres pièces du jeu encore présentes*). Celle-ci correspond alors à une **feuille rouge**.

En regardant aux 4 cases situées en haut, en bas, à gauche et à droite de la feuille rouge, on peut affirmer que celle-ci est entourée de deux autres pièces :

- Un ciseaux rouge
- un ciseaux noir

Ainsi la portée maximale de la feuille rouge est de 2 cases maximum. (*voir cases grises*)

1.2.2 Règles des éliminations

En considérant une pièce d'une certaine couleur, voici les règles qui permettent d'éliminer une pièce adverse :

- Si la portée de déplacement le permet, une pierre peut éliminer un ciseau adverse et prendre sa place.
- Si la portée de déplacement le permet, une feuille peut éliminer une pierre adverse et prendre sa place.
- Si la portée de déplacement le permet, un ciseau peut éliminer une feuille adverse et prendre sa place.

Le mécanisme d'élimination entre les différentes pièces est inspirée directement du jeu « Pierre, Feuille, Ciseaux ». Ainsi, une pièce d'un certain type T peut en manger une autre d'un type T' et se mettre à sa place si c'est une pièce adverse et si les types coïncident avec les règles citées ci-dessus.

1.2.3 Fin de la partie

La partie de Capaci se termine lorsque l'une des conditions suivante est validée :

- Un joueur ne possède plus aucune pièce d'une des catégories. *Si le joueur rouge ne possède plus de ciseaux, la partie est terminée et il a perdu.*
- Toutes les pièces d'un joueur ont une portée de déplacement nulle. *On parle de pièces isolées.*

2 Mise en place des schémas de classe

Pour commencer, le jeu de Capaci se déroule sur un **Plateau** de taille 6×6 où s'affronte deux **joueurs** qui possèdent un certain nombre de **pièces**. Avec cette première réflexion, on distingue déjà trois classes :

- Plateau permettant de représenter et de gérer les interactions de l'utilisateur avec celui-ci.
Le plateau est représenté par une grille contenant des pièces rouges ou noires.
- Joueur permettant de représenter les deux joueurs qui s'affrontent sur la partie.
Chaque joueur se distingue grâce à une couleur, qui définira ensuite la couleur des pièces, la liste des pièces qu'il possède et par un pseudo.
- Piece permettra elle de représenter les informations de chaque pièce comme leur type, leur position dans la grille, ...

2.1 La classe Plateau

2.1.1 Diagramme de classe

Plateau
- Grille : Piece[6][6]
+ getVoisin(ligne : int, colonne : int) : Piece[]
+ estDehors(ligne : int, colonne : int) : boolean
+ majPlateau() : void
+ getPiece(ligne : int, colonne : int) : Piece
+ supprimer(p : Piece) : void

2.1.2 Description de la classe `Plateau`

L'attribut `Grille`

La grille va représenter le plateau. Puisque la taille du plateau est de 6 lignes et 6 colonnes, la grille va alors être un tableau de `Piece` à deux dimensions de $6_{lig} \times 6_{col}$ soit 36 cases au total. En précisant qu'une case de la grille peut avoir trois « états » différents :

- Aucune pièce
- Une pièce rouge
- Une pièce noire

La méthode `getVoisin(ligne : int, colonne : int) : Piece[]`

Prenant en paramètres deux entiers `ligne` et `colonne`, représentant respectivement la ligne et la colonne d'une case, et renvoyant un tableau de `Piece`, cette méthode permet de récupérer la liste des voisins de la pièce située aux coordonnées (`ligne`, `colonne`) dans la grille.

Le tableau pourra contenir **au maximum** 4 pièces représentant les voisins gauche et droit ainsi que les voisins haut et bas. *On pourra aussi utiliser cette méthode pour définir la portée de déplacement d'une pièce.*

La méthode `estDehors(ligne : int, colonne : int) : boolean`

La méthode `estDehors` permet de déterminer si la case étudiée est en dehors des frontières de la grille c'est à dire dépasse si l'indice de la ligne (*resp. colonne*) est inférieur à 0 ou supérieur à 5. Cela permettra de gérer les cas où on étudie une case sur les côtés de la grille.

La méthode `majPlateau() : void`

La méthode `majPlateau` a pour rôle de mettre à jour l'état du plateau après chaque action ou tour de jeu. Elle permet notamment de :

- Vérifier l'état de chaque case de la grille et appliquer les règles du jeu pour les pièces concernées.
- Mettre à jour les positions des pièces si certaines sont déplacées ou capturées.
- Détecter et gérer les événements particuliers liés au plateau.

Cette méthode ne prend aucun paramètre et ne renvoie aucune valeur, car elle agit directement sur l'attribut `Grille` de la classe. Elle garantit que la représentation interne du plateau reste cohérente avec l'état réel du jeu à tout moment.

La méthode `getPiece(ligne : int, colonne : int) : Piece`

La méthode `getPiece` prend en paramètres deux entiers `ligne` et `colonne`, correspondant respectivement à la ligne et à la colonne d'une case du plateau. Elle renvoie la `Piece` située à ces coordonnées dans la grille. Cette méthode permet d'accéder à une pièce précise sans exposer directement l'attribut `Grille` afin de :

- Vérifier l'état d'une case avant un déplacement ou une action.
- Récupérer la couleur ou le type d'une pièce pour appliquer les règles du jeu.
- Faciliter l'affichage du plateau dans une interface graphique ou un terminal.

On se permettra d'utiliser la méthode `estDehors(ligne, colonne)` pour éviter des erreurs lors de l'accès à la grille.

La méthode `supprimer(p : Piece)`

La méthode `supprimer` prend en paramètre une instance de `Piece` `p` et a pour rôle de retirer cette pièce du plateau. Concrètement, elle localise la pièce dans l'attribut `Grille` et remplace sa case par une case vide. Cette méthode est utile pour :

- Gérer la capture d'une pièce par l'adversaire.
- Mettre à jour l'état du plateau après un déplacement ou une action spécifique du jeu.
- Maintenir la cohérence de la grille en supprimant les pièces qui ne doivent plus apparaître sur le plateau.

Elle agit directement sur l'attribut `Grille` et ne renvoie aucune valeur.

2.2 La classe Joueur

2.2.1 Diagramme de classe

Joueur
- nom : String - couleur : String - nbCiseaux : int - nbFeuille : int - nbPierre : int

2.2.2 Description de la classe Joueur

L'attribut nom : String

Cet attribut privé *sera accessible via accesseur et mutateur* permettra d'associer un pseudo à chaque joueur de la partie.

Les attribut nbCiseaux, nbFeuille, nbPierre

Ces trois attributs sont des compteurs qui permettent au joueur de savoir combien il lui reste de pièces dans chaque catégorie. Dans le cas où un des trois compteurs tombe à 0, le joueur perd.

L'attribut couleur : String

Cet attribut permet d'identifier la couleur associée au joueur (*rouge ou noir*). Il est utilisé pour distinguer les deux joueurs de la partie et pour déterminer la couleur des pièces qui leur appartiennent.

2.3 La classe Piece

2.3.1 Diagramme de classe

Piece
- joueur : Joueur - type : Forme - ligne : int - colonne : int - nbCaseDeplacement : int - estIsole : boolean
+ deplacement(ligneA : int, colonneA : int) : void + verifDeplacement(ligneA : int, colonneA : int, plateau : Plateau) : boolean

2.3.2 Description de la classe Piece

Attribut joueur : Joueur

L'attribut joueur permet d'identifier le joueur auquel appartient la pièce. Il référence directement une instance de la classe Joueur, ce qui permet d'associer chaque pièce à son propriétaire. La couleur de la pièce (*rouge ou noire*) peut ainsi être déduite indirectement à partir du joueur auquel elle est liée.

Lors du déroulement de la partie, cet attribut permet notamment de :

- déterminer quelles pièces peuvent être jouées pendant le tour d'un joueur donné ;
- distinguer les pièces alliées des pièces adverses ;
- appliquer correctement les règles de déplacement et de capture.

On considérera qu'une pièce tente d'attaquer une pièce adverse lorsque les joueurs associés aux deux pièces sont différents, c'est-à-dire lorsque `piece1.joueur != piece2.joueur`.

Attribut `type` : `Forme`

Le type de la pièce permet de savoir si la pièce est une pierre, une feuille ou un ciseau. On utilisera pour cela une énumération `Forme` qui contiendra les trois valeurs possibles : `PIERRE`, `FEUILLE`, `CISEAUX` ainsi qu'une méthode permettant de savoir qui elle peut manger.

Les attribut `ligne` : `int` **et** `colonne` : `int`

Ces deux derniers permettent de déterminer la position de la pièce sur la grille de jeu. On devra cependant faire en sorte qu'ils ne doivent pas peuvent être :

- inférieur à l'indice 0 *qui représente la première ligne/colonne*
- supérieur à 5 *qui représente la dernière ligne/colonne*

Attribut `nbCaseDeplacement` : `int`

L'attribut `nbCaseDeplacement` est capital pour le fonctionnement de notre jeu car c'est grâce à lui qu'on pourra savoir si une pièce est isolée ou alors le cas échéant de déterminer sa portée de déplacement.

Attribut `estIsole` : `boolean`

On appelle **pièce isolée** une pièce qui ne possède aucun voisins. Dans ces cas là la variable sera mise à `true` et le nombre de pièces autour de celle ci sera mis à 0.

Si toutes les pièces d'un joueur sont isolées la partie se termine car le joueur ne peut plus se déplacer. C'est un peu le « échec-et-mat » de nos bon vieux échecs.

La méthode `deplacement(ligneA : int, colonneA : int) : void`

La méthode `deplacement` permet de déplacer une pièce d'une position de départ (`ligneD, colonneD`) vers une position d'arrivée (`ligneA, colonneA`) sur le plateau. Elle prend en paramètres (`ligneA, colonneA`) : les coordonnées de la case où la pièce doit être déplacée.

Cette méthode devra effectuer les opérations suivantes :

- Vérifier que la position d'arrivée est bien à l'intérieur du plateau (en utilisant la méthode `estDehors` de la classe `Plateau`).
- Vérifier que la case d'arrivée est libre ou que le déplacement respecte les règles du jeu.
- Mettre à jour l'attribut `Grille` du plateau en déplaçant la pièce de la case initiale vers la case finale.
- Supprimer ou gérer toute pièce qui pourrait être capturée lors du déplacement.

La méthode ne renvoie aucune valeur, car elle agit directement sur le plateau et sur l'état interne de la pièce. Elle garantit que le déplacement reste conforme aux règles et que le plateau reste cohérent après l'action.

La méthode `verifDeplacement(ligneA : int, colonneA : int, plateau : Plateau) : boolean`

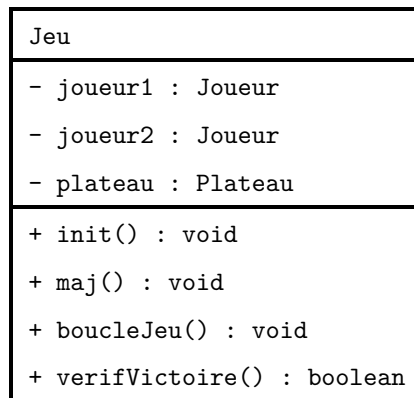
La méthode `verifDeplacement` prend en paramètres les coordonnées de destination (`ligneA, colonneA`) et permet de vérifier si le déplacement de la pièce vers cette case est autorisé selon les règles du jeu. Elle renvoie un booléen : `true` si le déplacement est possible, `false` sinon.

Les vérifications effectuées par cette méthode peuvent inclure :

- La validité des coordonnées (`ligneA, colonneA`) par rapport aux dimensions du plateau (en utilisant la méthode `estDehors` de la classe `Plateau`).
- L'occupation de la case d'arrivée : une pièce ne peut se déplacer que vers une case vide.
- La portée de la case d'arrivée, compatible avec la portée de la pièce à déplacée.
- Les contraintes liées à la capture de pièces adverses.

2.4 La classe Jeu

2.4.1 Diagramme de classe



2.4.2 Description de la classe Jeu

Rôle général de la classe Jeu

La classe `Jeu` représente le cœur logique de la partie de Capaci. Elle joue le rôle de contrôleur principal en guidant le déroulement global du jeu, depuis l'initialisation de la partie jusqu'à la détection de sa fin. Elle ne gère pas directement les règles fines du jeu (*déplacements, captures, voisinage*), mais s'appuie sur les autres classes du modèle, notamment `Plateau`, `Joueur` et `Piece`.

Les attributs `joueur1 : Joueur` et `joueur2 : Joueur`

Ces deux attributs représentent les deux joueurs participant à la partie. Chaque instance de `Joueur` contient les informations propres à un joueur, telles que son nom, sa couleur ainsi que les compteurs de pièces restantes pour chaque catégorie.

La classe `Jeu` utilise ces deux attributs afin de :

- gérer l'alternance des tours entre les joueurs.
- déterminer quel joueur est actif à un instant donné.
- vérifier les conditions de victoire ou de défaite associées à chaque joueur.

L'attribut `plateau : Plateau`

L'attribut `plateau` référence une instance de la classe `Plateau`, qui représente l'état global du jeu. Il contient la grille de jeu ainsi que l'ensemble des pièces positionnées sur celle-ci.

La présence de cet attribut est essentielle, car toutes les informations relatives :

- aux positions des pièces.
- aux déplacements possibles.
- aux voisinages et portées de déplacement.
- aux captures et suppressions de pièces.

sont centralisées au sein du plateau.

La classe `Jeu` s'appuie donc sur le `Plateau` pour faire évoluer l'état de la partie et garantir la cohérence globale du jeu à chaque tour.

La méthode `init() : void`

La méthode `init` a pour rôle d'initialiser une nouvelle partie de Capaci. Elle permet notamment de :

- créer les instances des deux joueurs.
- instancier le plateau de jeu.
- placer les pièces sur la grille selon la configuration initiale définie par les règles du jeu.

Cette méthode ne renvoie aucune valeur, car elle agit directement sur les attributs internes de la classe `Jeu`. Elle garantit que l'état initial du jeu est valide avant le début de la partie.

La méthode `maj()` : void

La méthode `maj` permet de mettre à jour l'état global du jeu après une action ou un tour de jeu. Elle s'appuie principalement sur les méthodes fournies par la classe `Plateau` afin de :

- appliquer les déplacements effectués par les joueurs.
- mettre à jour la position des pièces.
- gérer les captures éventuelles.
- maintenir la cohérence de l'état du jeu.

La méthode `boucleJeu()` : void

La méthode `boucleJeu` représente la boucle principale de la partie. Elle gère l'enchaînement des tours de jeu en alternant entre les deux joueurs tant que les conditions de fin de partie ne sont pas atteintes. Concrètement, cette méthode permet de :

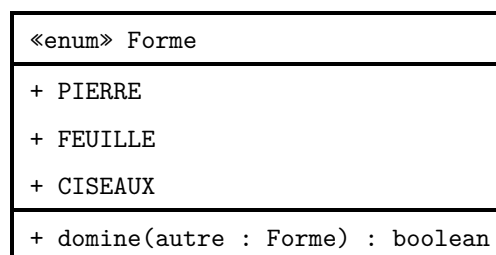
- déterminer le joueur actif.
- attendre et traiter une action de jeu.
- mettre à jour l'état du plateau.
- vérifier si la partie doit se terminer.

La méthode `verifVictoire()` : boolean

La méthode `verifVictoire` permet de vérifier si une condition de fin de partie est atteinte. Elle s'appuie sur les règles définies précédemment, à savoir :

- la disparition complète d'une catégorie de pièces pour un joueur ;
- l'impossibilité de déplacement de toutes les pièces d'un joueur (*pièces isolées*).

La méthode renvoie `true` si la partie est terminée et `false` sinon. Elle est appelée régulièrement au cours de la partie, en particulier dans la boucle principale du jeu.

2.5 L'énumération `Forme`**2.5.1 Diagramme de classe****2.5.2 Description de l'énumération `Forme`****Rôle général de l'énumération `Forme`**

L'énumération `Forme` représente les trois types de pièces disponibles dans le jeu de Capaci : `PIERRE`, `FEUILLE` et `CISEAUX`. Elle permet de définir clairement les catégories de pièces et de faciliter la gestion des interactions entre elles, notamment en ce qui concerne les règles de capture inspirées du jeu « Pierre, Feuille, Ciseaux ».

La méthode `domine(autre : Forme) : boolean`

La méthode `domine` permet de déterminer si une forme domine une autre selon les règles du jeu « Pierre, Feuille, Ciseaux ».

Elle prend en paramètre une instance de `Forme` appelée `autre` et renvoie un booléen :

- `true` si la forme courante domine la forme `autre`.
- `false` sinon.

3 Le diagramme de classe

