

Jeu d'échecs sur Unity

Dossier de projet - TPI

Écrit par : **Arthur Bottemanne**

Chef de projet : **Loïc Viret**

Premier expert : **Roberto Ferrari**

Deuxième expert : **Claude-Albert Muller Theurillat**

Glossaire :

Pièce clouée : Aux échecs, une pièce clouée est définie comme une pièce menacée ne pouvant pas se déplacer sans exposer une autre pièce de plus grande valeur à une capture.

Moteur de jeu : Un moteur de jeu est un logiciel qui facilite le développement de jeux vidéo en mettant à disposition des outils tels que la simulation de physique, le rendu graphique, la gestion des entrées utilisateur et autres fonctionnalités.

Unity : Unity est un moteur de jeu multiplateforme en deux dimensions et en trois dimensions.

Table des matières

Glossaire :	2
1 Analyse préliminaire.....	4
1.1 Introduction.....	4
1.2 Objectifs.....	4
1.3 Planification initiale.....	6
2 Analyse / Conception	7
2.1 Concept.....	7
2.1.1 Diagrammes de flux.....	7
2.1.2 Maquettes.....	12
2.2 Stratégie de test.....	14
2.3 Risques techniques	14
2.4 Planification détaillée.....	16
2.5 Dossier de conception.....	22
2.5.1 Diagramme de classes	22
2.5.2 Diagrammes de séquences	23
3 Réalisation	25
3.1 Dossier de réalisation.....	25
3.2 Description des tests effectués.....	30
3.3 Erreurs restantes	30
3.4 Liste des documents fournis.....	30
4 Conclusion	31
5 Annexes	32
5.1 Résumé du rapport du TPI / version succincte de la documentation.....	32
5.2 Sources – Bibliographie.....	32
5.3 Journal de travail	32
5.4 Manuel d'Installation.....	32
5.5 Manuel d'Utilisation	32
5.6 Archives du projet	32

1 Analyse préliminaire

1.1 Introduction

Ceci est le dossier de projet pour mon TPI qui est réalisé dans le cadre du CPNV. Pour réaliser ce projet, 90 heures sont mises à disposition.

L'objectif est de créer un jeu d'échecs représenté en deux dimensions en utilisant Unity comme moteur de jeu. Les règles officielles devront être implémentées et fonctionnelles pour que deux joueurs s'affrontent dans une partie.

Les parties seront chronométrées en fonction du temps choisi par les joueurs, et les parties seront enregistrées dans un fichier en notation algébrique.

1.2 Objectifs

L'objectif de ce projet est de développer un jeu d'échecs en deux dimensions à l'aide du moteur de jeu Unity.

Le jeu permettra de jouer aux échecs à deux sur le même ordinateur en incluant les coups spéciaux. Il permettra également de fournir le chronomètre pour chaque joueur afin de pouvoir faire des parties « officielles » ainsi que des parties rapides.

Le jeu doit permettre à deux joueurs de s'affronter aux échecs avec les règles officielles incluant :

1. Ce sont toujours les blancs qui commencent. Les joueurs choisiront eux-mêmes qui débute la partie.
2. Chaque joueur doit être capable de ne déplacer que ses propres pièces.
3. Il doit être possible d'activer/désactiver la prévision d'où pourront se déplacer les pièces en fonction de leur type.
4. Les coups spéciaux doivent être implémentés (Roque, prise en passant et promotion).
5. Un joueur ne doit pas pouvoir déplacer une pièce clouée.
6. Les conditions de victoire doivent être implémentées (échec, échec et mat, pat, 50 coups, triple répétition, accord et temps).
7. Un fichier de récapitulation de la partie en notation algébrique résumera la partie.

Pour les règles du jeu d'échecs, le candidat peut se baser sur cette page :

https://fr.wikipedia.org/wiki/R%C3%A8gles_du_jeu_d%27%C3%A9checs

Pour la notation algébrique, le candidat peut se baser sur cette page :

https://fr.wikipedia.org/wiki/Notation_alg%C3%A9brique

Pour les objectifs, il existe aussi 7 points techniques qui seront évalués pour ce projet.

1. Déplacement des pièces

- Respect des différents déplacements des pièces
- Prévision des positions possibles lors d'un déplacement
- Ergonomie du déplacement

2. Gestion du tour des joueurs

- Identification du joueur dont c'est le tour
- Possibilité de ne déplacer que ses pièces
- Pièce clouée

3. Coups spéciaux

- Le petit roque
- Le grand roque
- La prise en passant

4. Promotion

- Le déclenchement de la promotion
- Les possibilités de promotion
- L'ergonomie de la fonctionnalité

5. Conditions de victoire

- La gestion de la mise en échec
- La validation de l'échec et mat
- Le pat

6. Le fichier récapitulatif

- Les codes de pièces
- Les cases concernées
- Les prises

7. Setup fonctionnel avec son protocole d'installation

- Sans erreur sur une machine Windows 10 64bits
- Pertinence du nom de l'exécutable et de son emplacement
- Protocole d'installation clair

1.3 Planification initiale

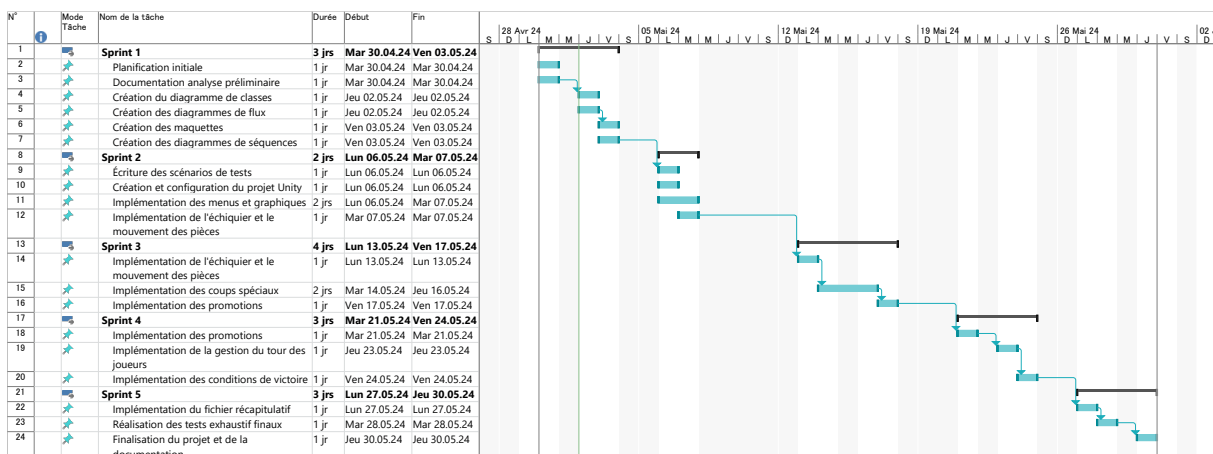


Figure 1: Planification initiale

La planification initiale a été réalisée en se basant sur le nombre de jours nécessaires pour terminer chaque tâche, afin d'obtenir une vue d'ensemble plus générale.

On peut attribuer à une tâche une durée d'une journée pour son exécution, mais elle peut finalement être réalisée plus rapidement.

Les tâches de tests et de documentation sont aussi prises en compte lors des tâches d'implémentation. En effet, j'ai décidé que durant l'implémentation, les documentations et les tests nécessaires seraient fait simultanément.

Ces tâches de tests et de documentation seront planifiées durant la révision de la planification initiale.

2 Analyse / Conception

2.1 Concept

2.1.1 Diagrammes de flux

Pour cette conception, cinq diagrammes de flux ont été réalisés pour démontrer la logique des aspects importants du programme.

Les diagrammes de flux représentent respectivement :

- La gestion du mouvement des pièces
- La gestion des mouvements spéciaux (roque, en passant)
- La gestion du tour des joueurs
- Les conditions de fin de partie
- La gestion du fichier récapitulatif

Avec ces diagrammes de flux, on peut avoir un aperçu entier sur la logique du programme.

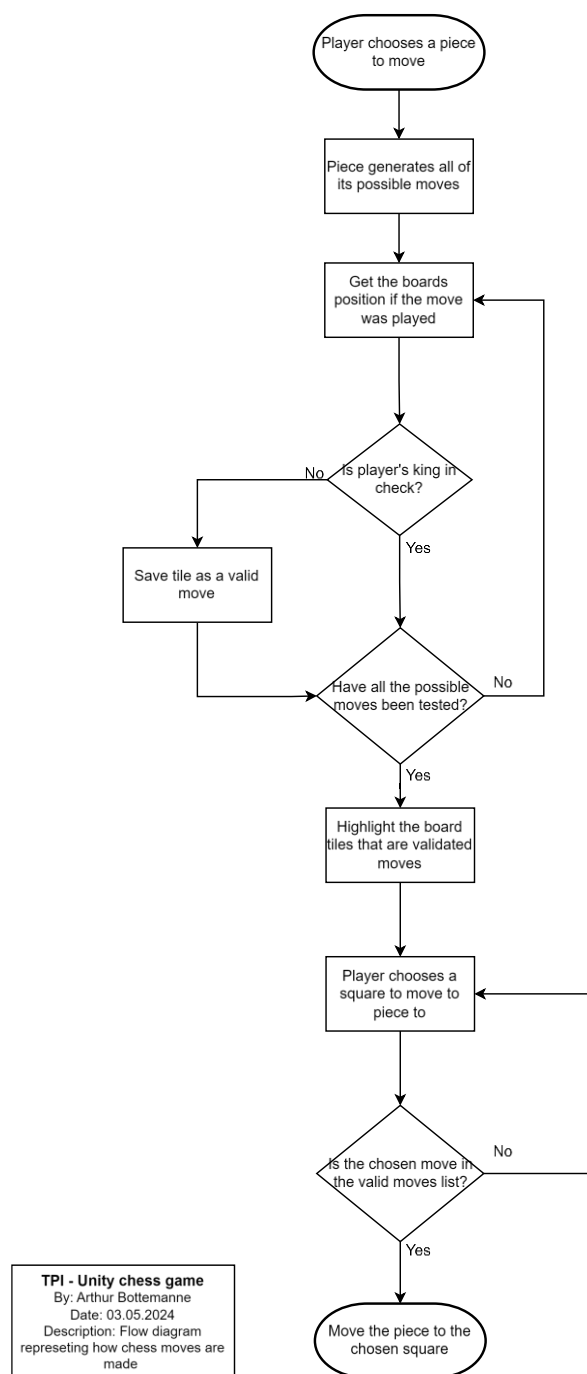


Figure 2: Diagramme de flux de la gestion du mouvement des pièces

Ce diagramme représente la logique de la génération de mouvement d'une pièce et de la validation du mouvement.

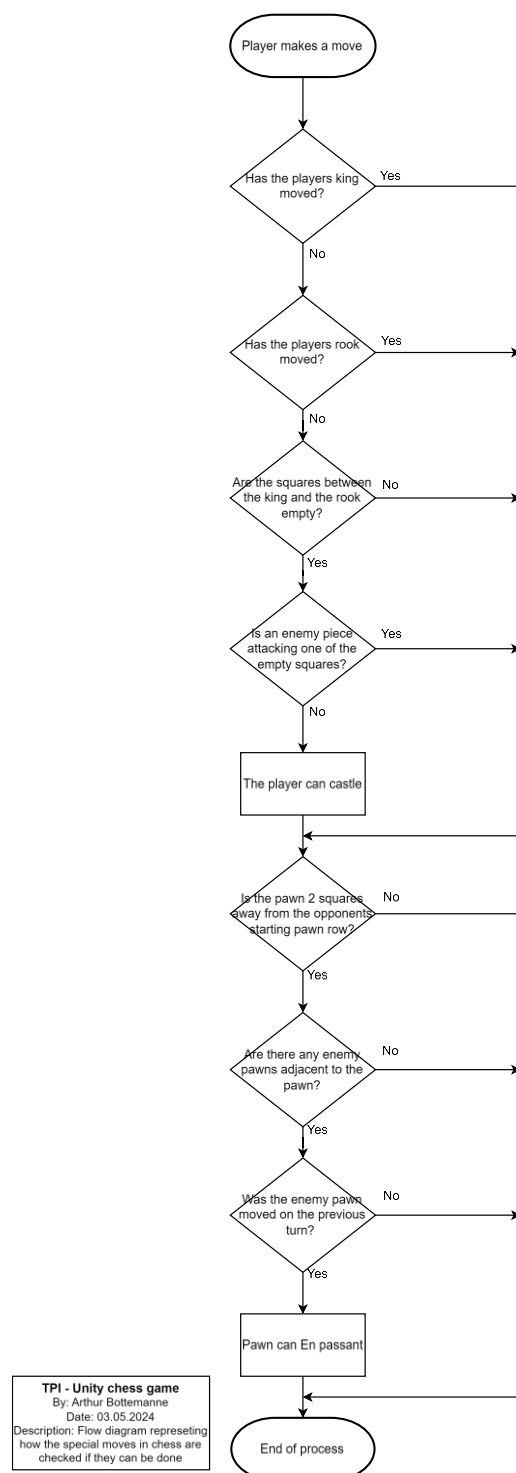


Figure 3: Diagramme de flux de la gestion des mouvements spéciaux

Ce diagramme représente les vérifications des conditions pour le roque et l'en passant aux échecs.

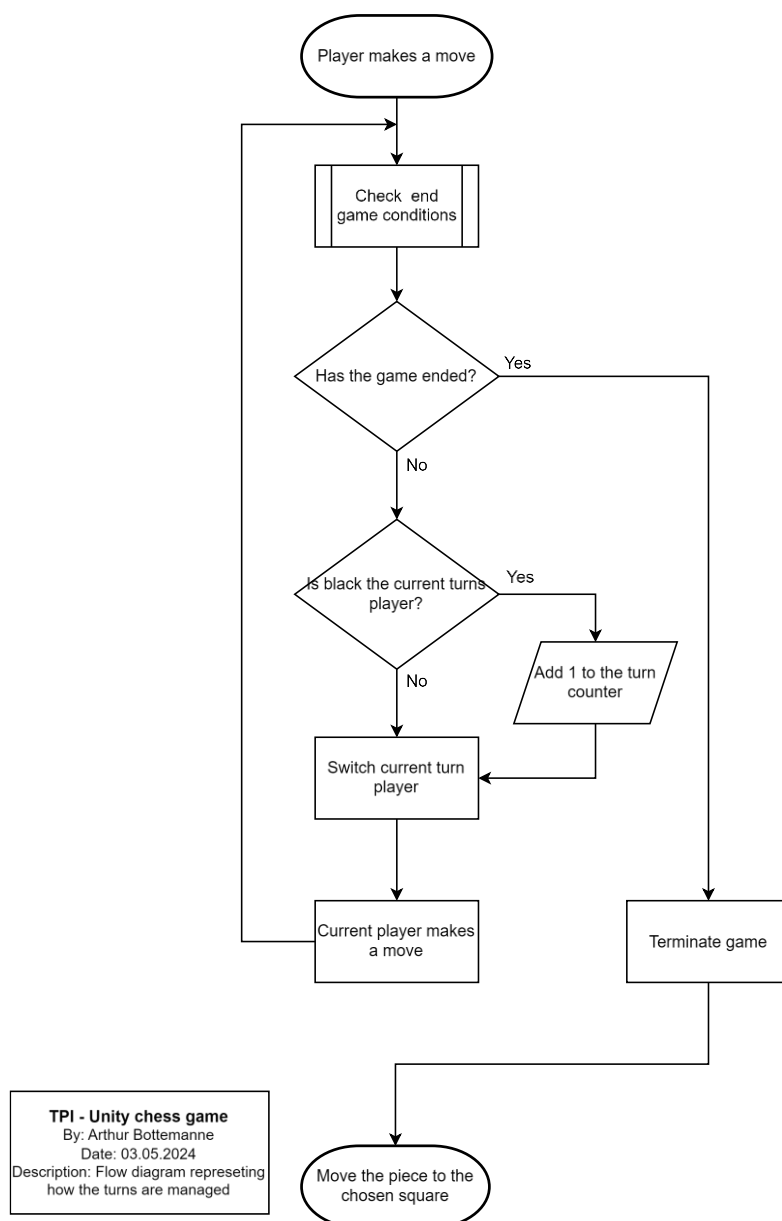


Figure 4: Diagramme de flux de la gestion du tour des joueurs

Ce diagramme représente la logique pour le changement et le compte des tours.

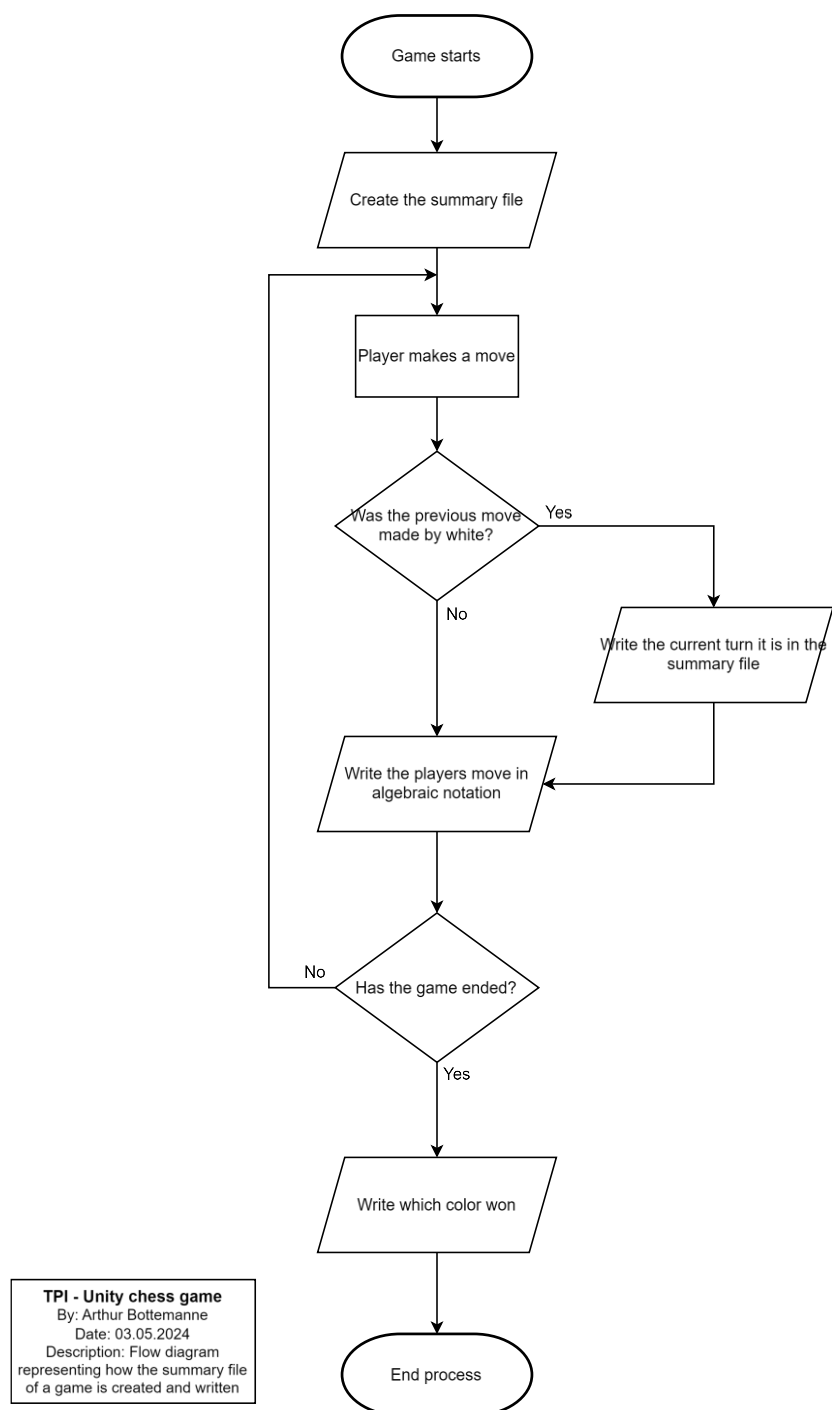


Figure 5: Diagramme de flux de la gestion du fichier récapitulatif

Ce diagramme représente la logique du fichier récapitulatif quand il écrit les coups et note les tours.

2.1.2 Maquettes

Cinq maquettes ont été réalisées pour ce jeu :

- Le menu principal
- La sélection du chronomètre
- La partie en cours
- La proposition d'un match nul
- Le message de fin de partie

Voici ci-dessous les maquettes :

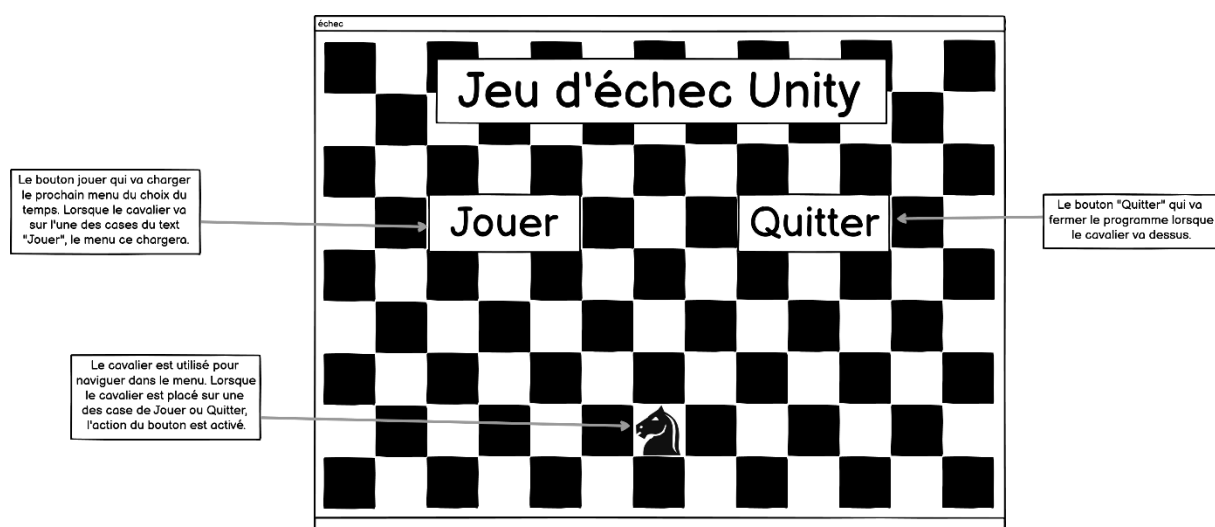


Figure 6: Maquette du menu principal

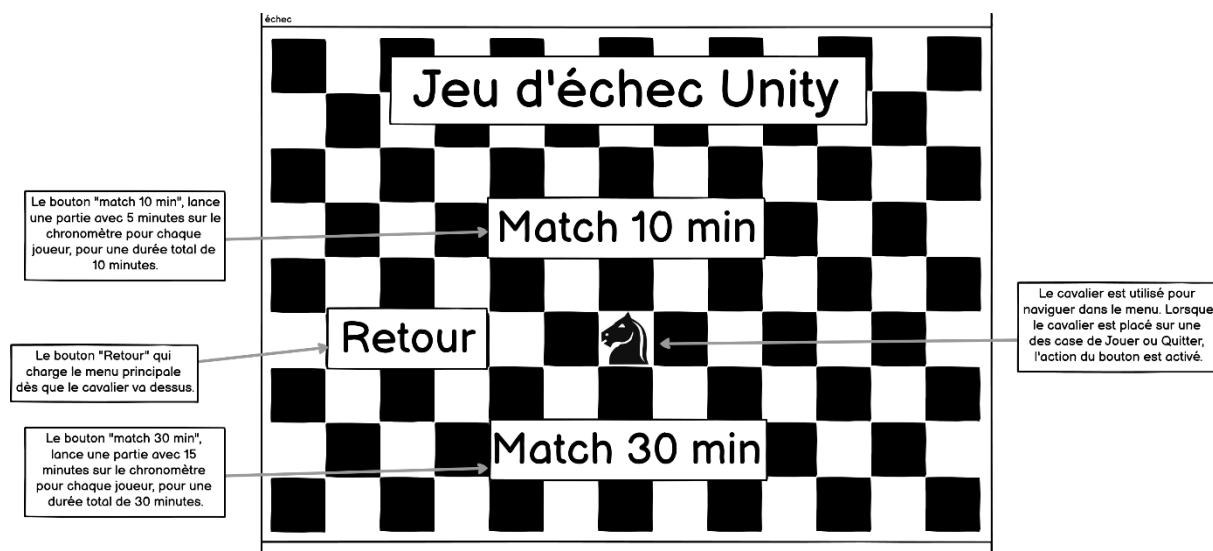


Figure 7: Maquette de la sélection du chronomètre

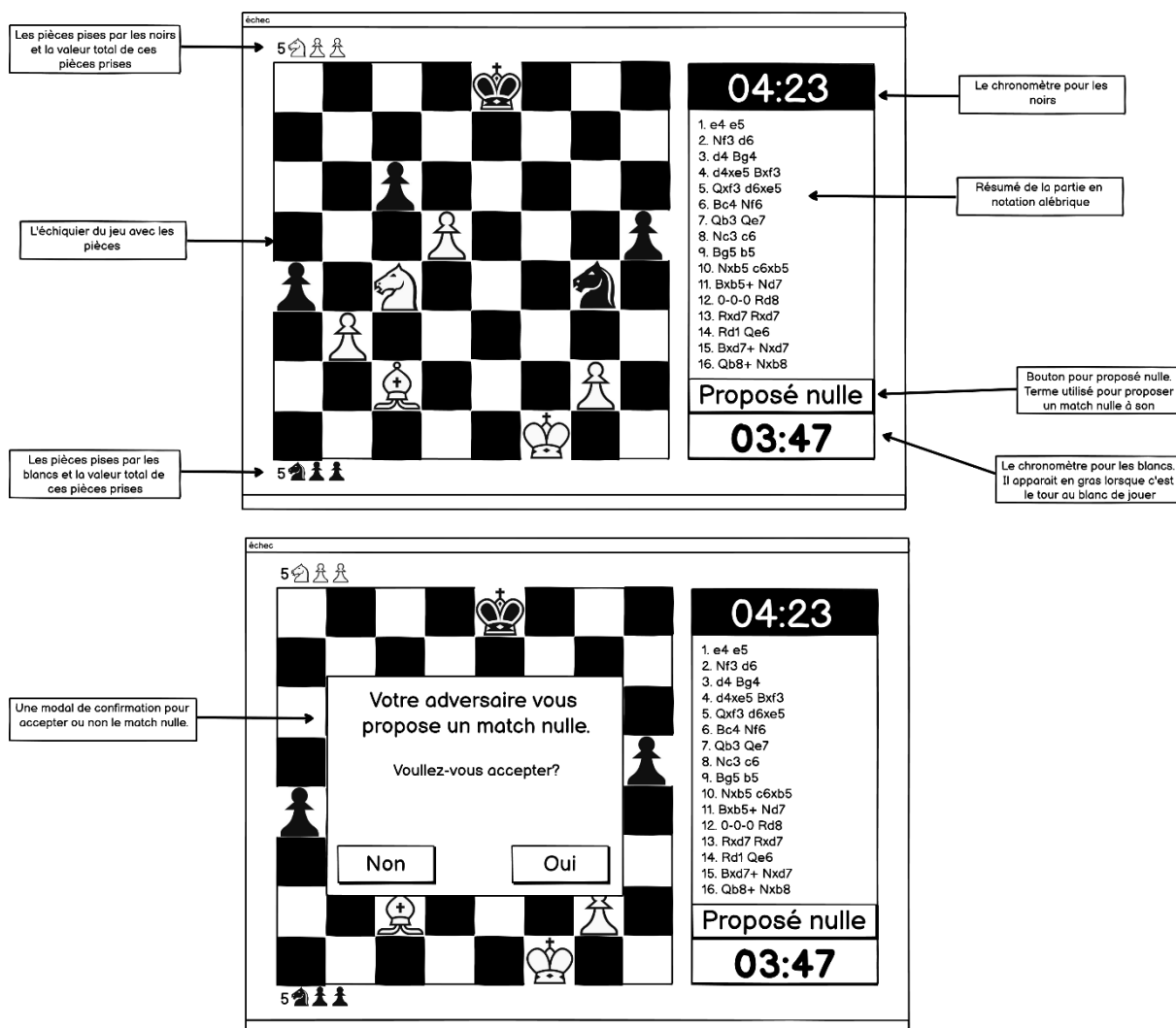


Figure 8: Maquette d'une partie en cours

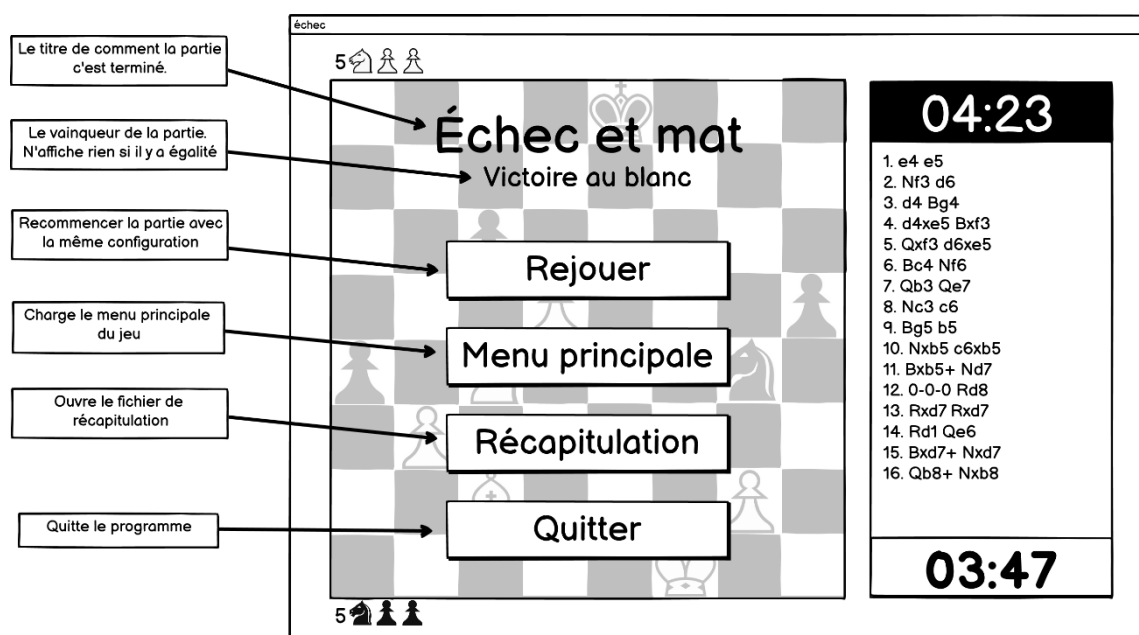


Figure 9: Maquette de la fin d'une partie

2.2 Stratégie de test

La stratégie de test comporte deux parties, les tests unitaires automatisés et les tests de bout en bout manuels.

Les tests unitaires automatisés :

Pour les tests unitaires, le « Test Runner » de Unity sera utilisé en « Play mode ». Les tests couvriront les fonctionnalités essentielles du programme. La raison est le temps alloué pour la réalisation des tests et mon inexpérience avec les tests automatisés avec Unity. Donc une grande majorité des tests pour assurer le bon fonctionnement du programme seront des tests manuels de bout en bout. Ces tests seront écrits et exécutés tout du long de la réalisation du projet.

Les tests d'intégration ne sont pas prévus en raison de mon inexpérience avec les tests sous Unity et du temps limité alloué pour les réaliser. Le but est de garder les tests unitaires pour les fonctionnalités nécessaires, cependant, si le temps le permet, il est possible que certains tests d'intégrations soient réalisés.

Les tests manuels de bout en bout :

Ils garantissent de manière exhaustive le bon fonctionnement du programme. Avant d'être réalisés, les scénarios de tests, ainsi que les cas de tests, seront définis au préalable. Ils seront réalisés moins fréquemment que les tests unitaires, mais ils sont impératifs pour assurer que le programme fonctionne correctement étant donné que les tests unitaires ne sont pas exhaustifs.

2.3 Risques techniques

Le premier risque majeur concerne mes compétences avec Unity. Bien que j'aie utilisé Unity quelques fois avant ce projet, certaines connaissances me font encore défaut. Pour certaines implémentations que je n'ai pas faites précédemment, cela pourrait se montrer un peu difficile à réaliser. Afin de remédier à ce risque, des recherches préalables sur Unity seront nécessaires pour bien comprendre comment implémenter la fonction désirée, les différentes manières possibles de le faire et la meilleure façon de procéder.

Le deuxième risque que j'anticipe est de m'assurer que mon programme respecte à 100 % les règles des échecs. Ceci est compliqué car il y a environ 10^{40} positions légales aux échecs. Dans ces positions, chaque pièce a ses propres règles de déplacement et de capture ce qui peut créer des configurations complexes avec de nombreuses interactions. Dans ces configurations, tester et assurer le bon fonctionnement du programme nécessite une analyse minutieuse et exhaustive. Pour éviter ces erreurs, je vais devoir tester le programme avec une variété de positions et de scénarios pour m'assurer qu'il fonctionne correctement dans toutes les situations possibles.

2.4 Planification

La planification du projet est gérée avec GitHub pour la gestion des tâches et des sprints. Pour la planification initiale et la révision de la planification, MS Project a été utilisé.

La méthodologie choisie est hybride car elle se base sur la méthode Waterfall avec quelques aspects de la méthode Agile. La raison de ce choix est la contrainte de la planification imposée pour le TPI. Toutes les tâches doivent être planifiées dès le premier jour, ce qui correspond à l'approche Waterfall. Cependant, je ne pense pas que cette approche soit suffisamment souple, ce qui ne me semble pas avantageux. C'est pourquoi certains aspects de la méthode Agile seront également appliqués pour introduire la flexibilité qui manque au modèle Waterfall.

Pour les sprints, des projets Kanban de GitHub ont été créés, et les tâches à réaliser sont des issues dans le répertoire GitHub. La liaison est ensuite faite entre les issues et les sprints pour les planifier.

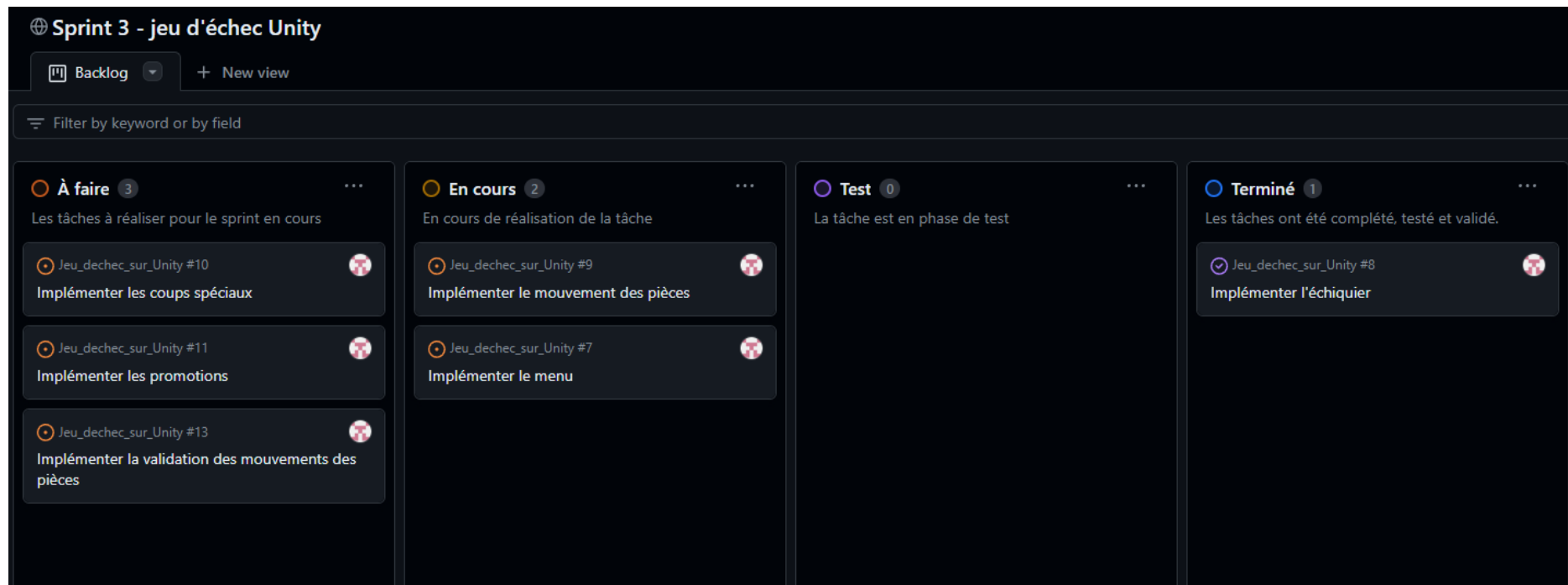
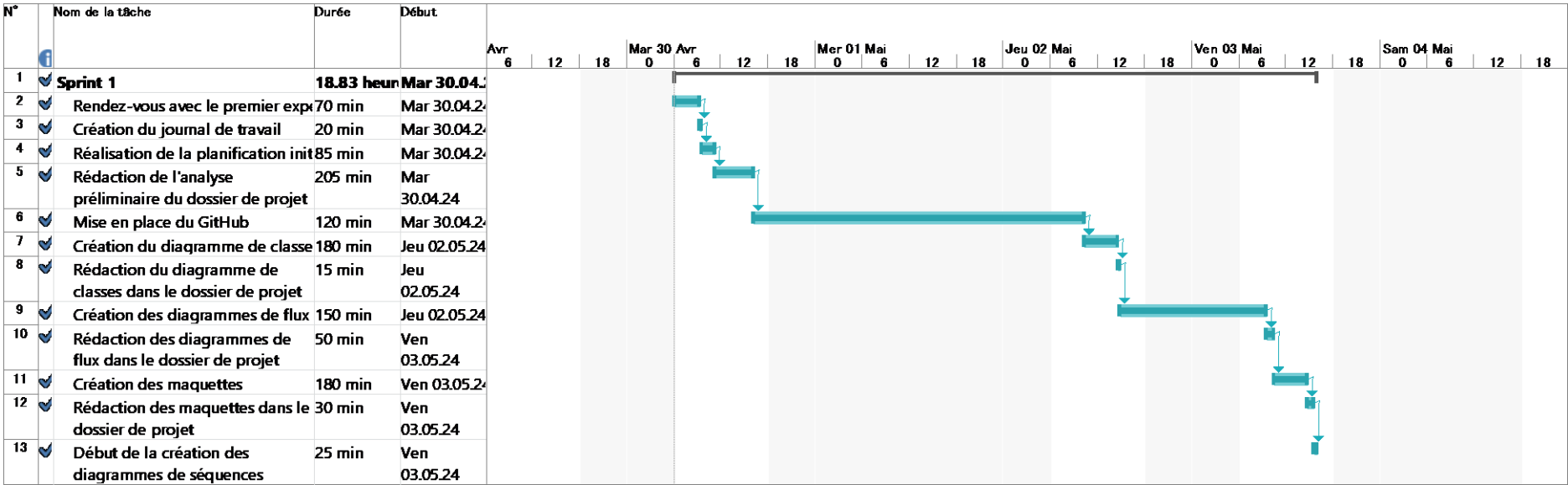
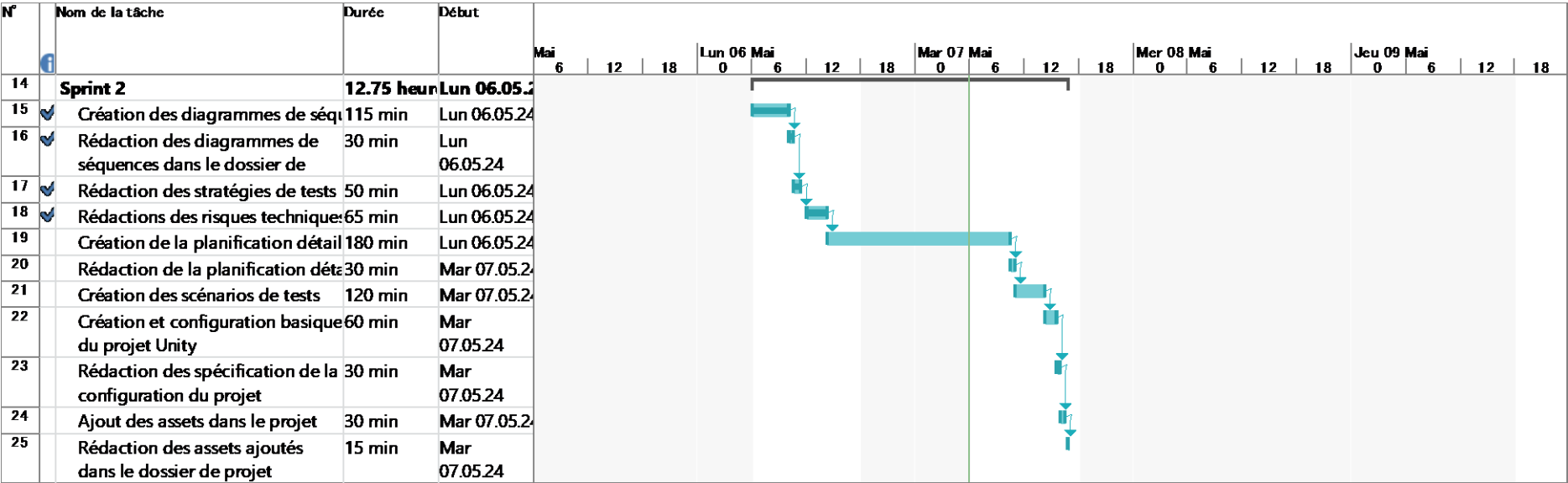
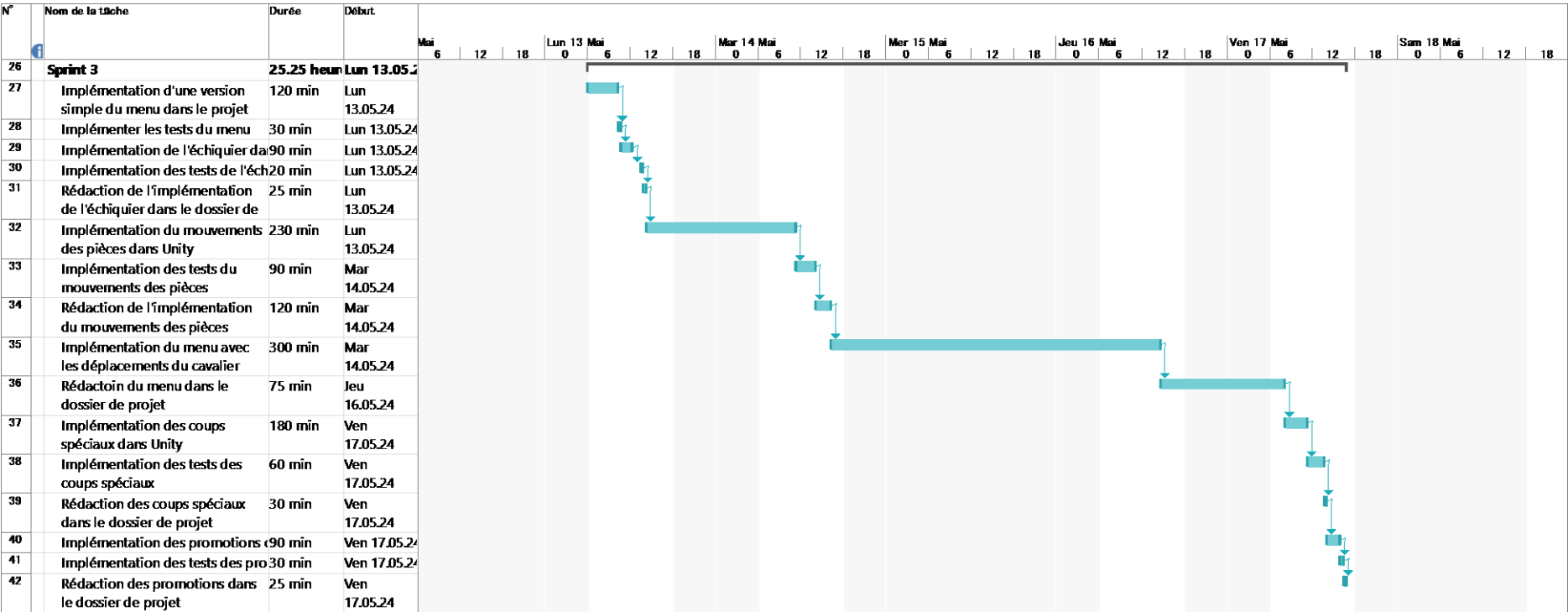


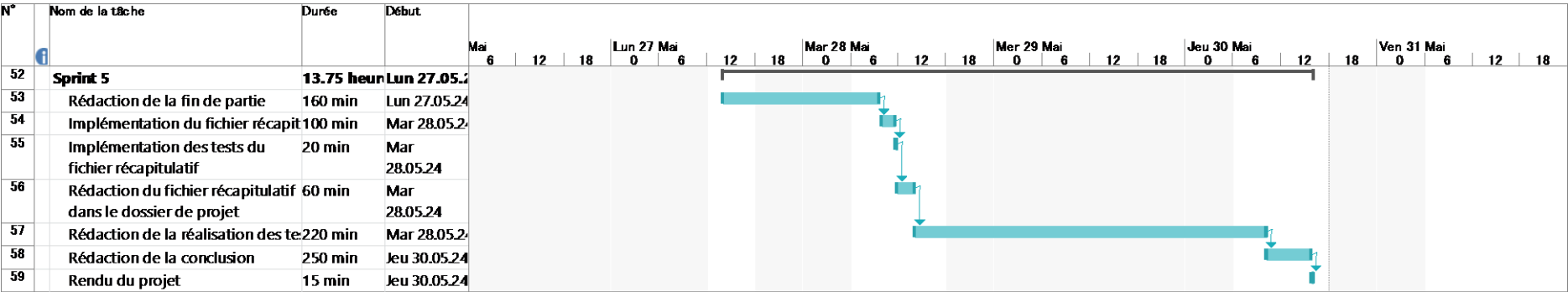
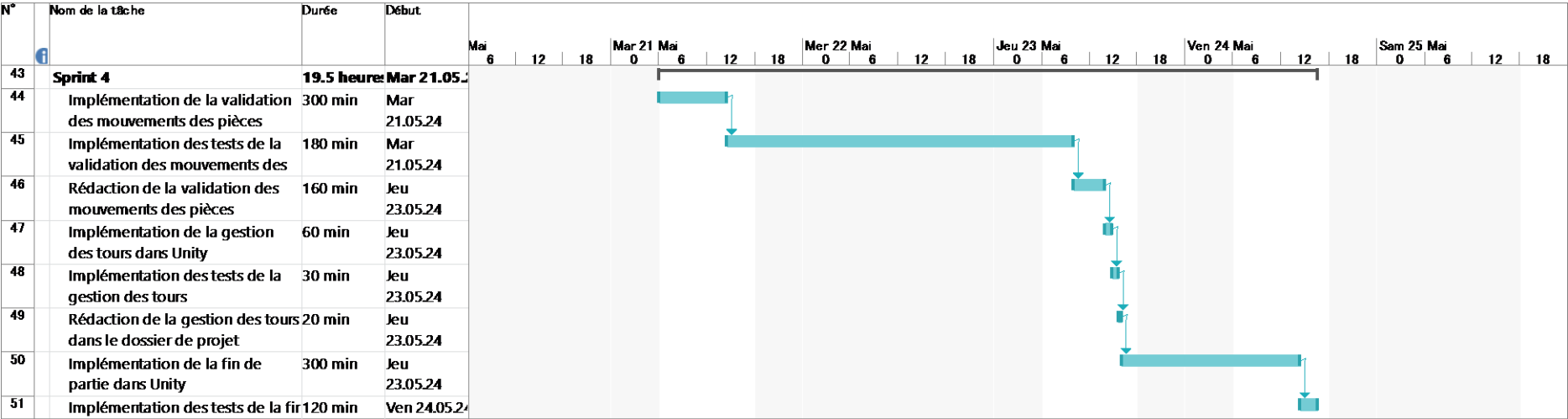
Figure 6: Exemple d'un sprint et de ces issues

Ces tâches sont planifiées à l'avance et découpées en sprints d'une durée d'une semaine. Si du retard est pris sur la réalisation des tâches, celles-ci peuvent être replanifiées pour le prochain sprint.









2.5 Dossier de conception

2.5.1 Diagramme de classes

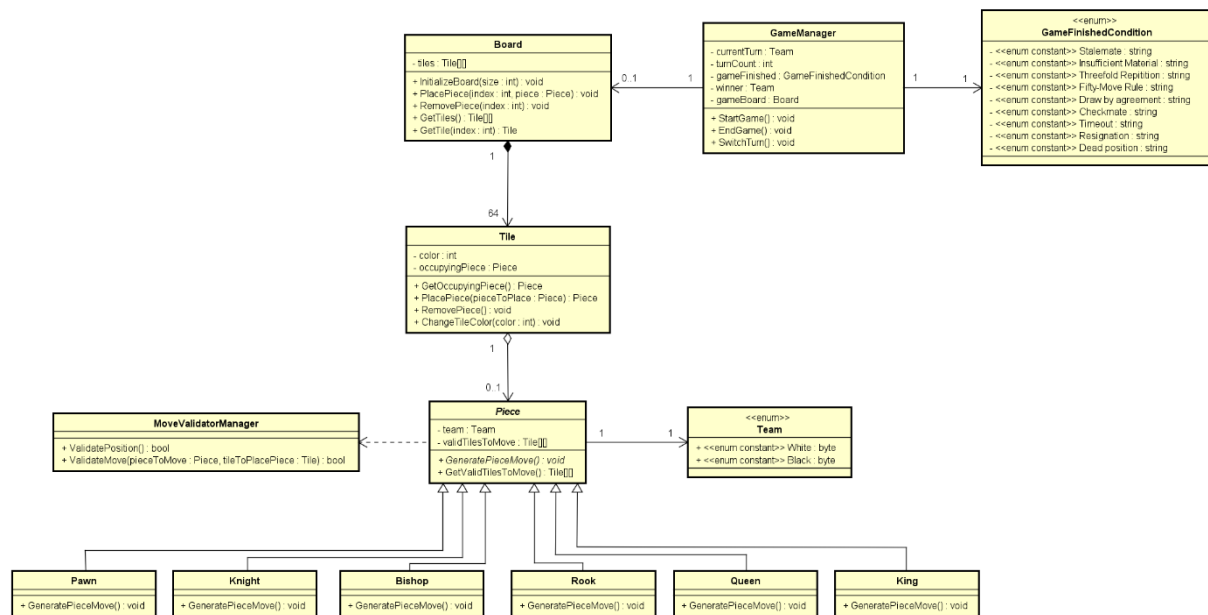


Figure 10: Diagramme de classes

Le diagramme de classes ci-dessus représente la structure des classes que j'envisage d'implémenter. À la fin de la réalisation du projet, un diagramme de classes à jour sera comparé à celui-ci.

Dans ce diagramme, la classe « GameManager » gère les fonctionnalités de condition de victoire et de tour. Le « Board » représente l'échiquier et agit sur les mouvements des pièces. Les classes des pièces (« Pawn », « Bishop » etc...) utilisent des concepts de polymorphisme pour implémenter leur propre méthode de mouvement.

2.5.2 Diagrammes de séquences

Les diagrammes de séquences réalisés sont concentrés sur le fonctionnement d'une partie, plus spécifiquement sur l'initialisation d'une partie, du mouvement d'une pièce et de la fin d'une partie.

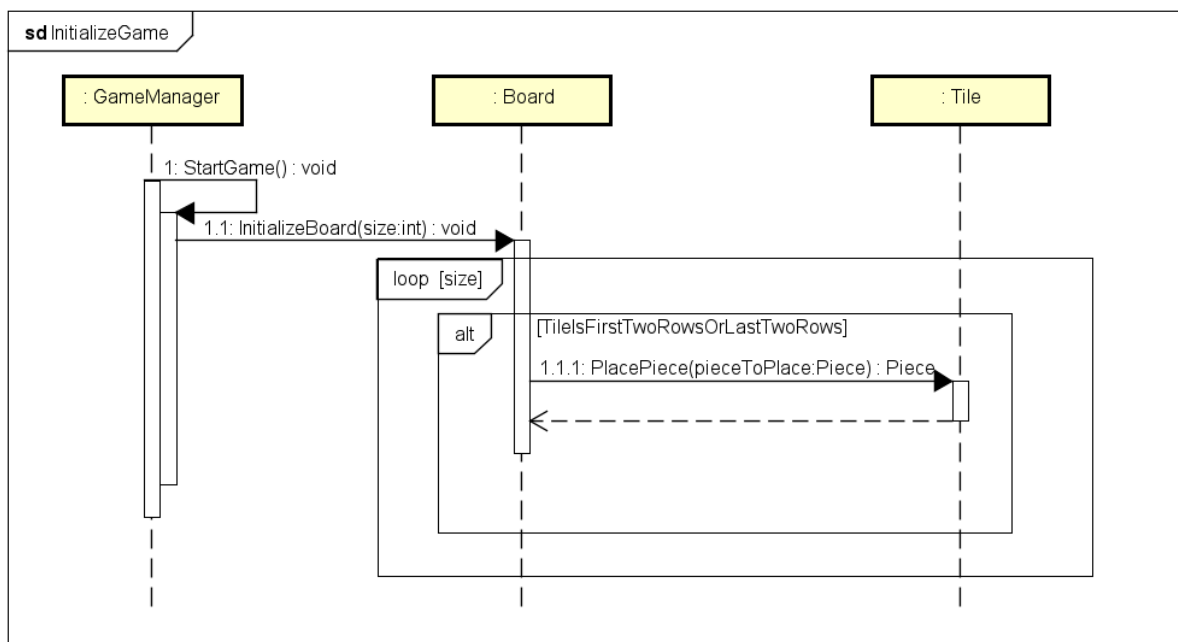


Figure 11: Diagramme de séquence de l'initialisation d'une partie

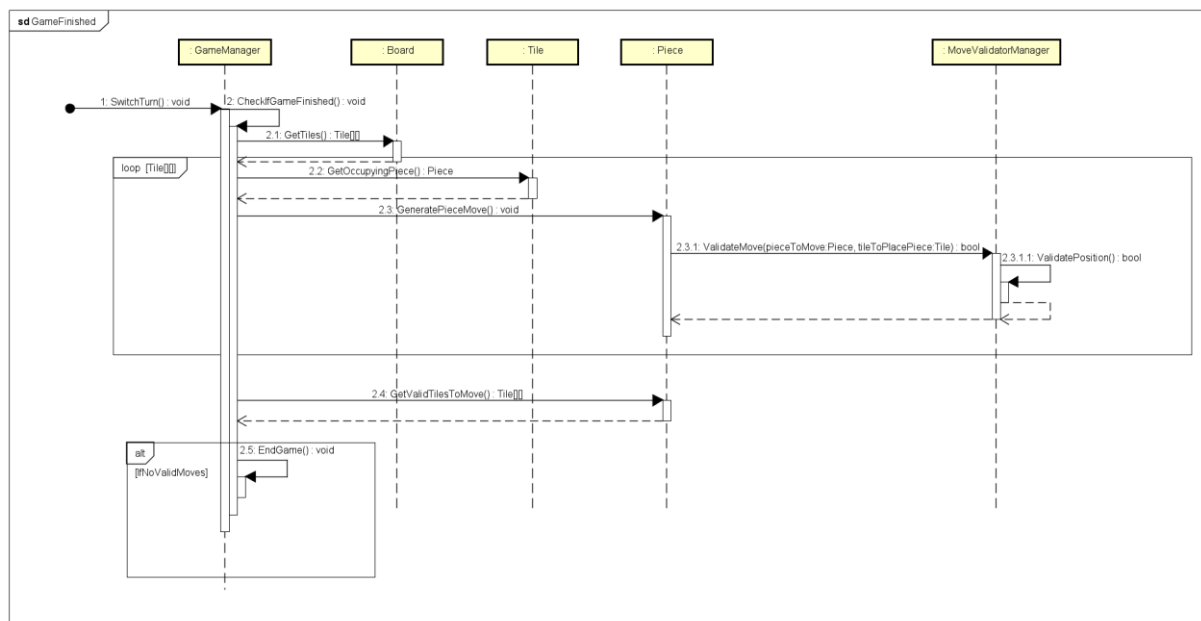


Figure 8: Diagramme de séquence de la fin d'une partie

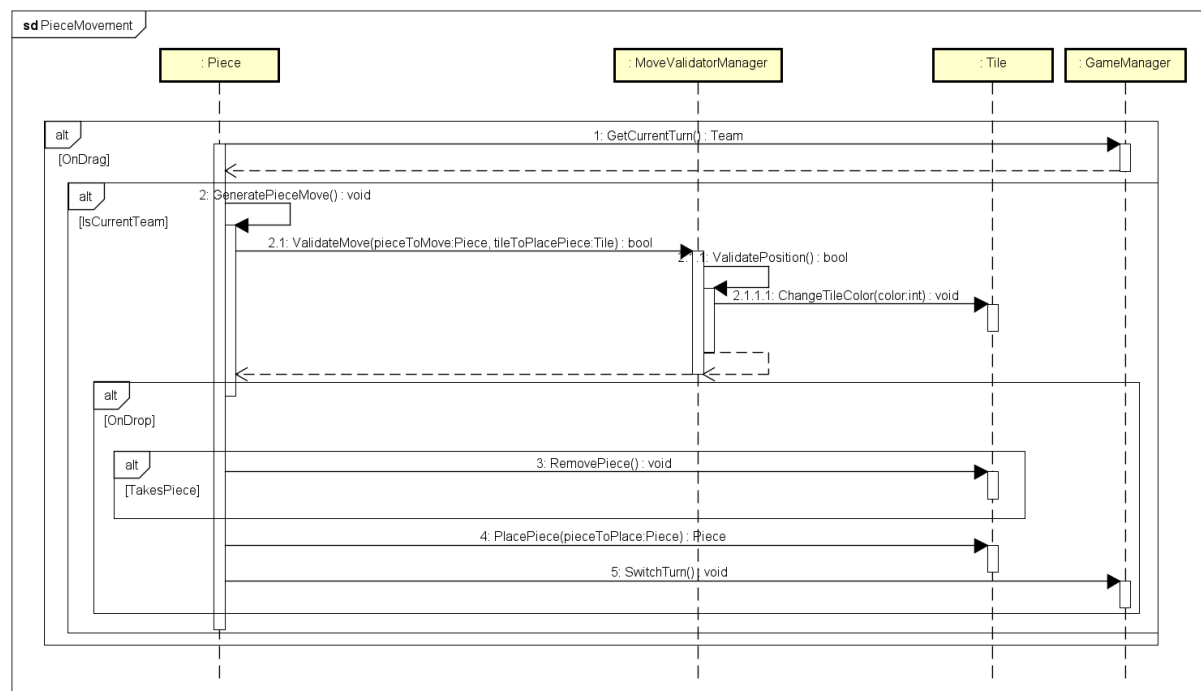


Figure 7: Diagramme de séquence du mouvement des pièces

3 Réalisation

3.1 Dossier de réalisation

3.1.1 Mise en place

Le projet Unity a été créé avec la version LTS (Long term support) 2022.3.19f1.

3.1.2 L'échiquier

3.1.2.1 Génération de l'échiquier

L'initialisation de l'échiquier est réalisée en créant les cases à partir d'un prefab « Tile » en tant qu'enfant du GameObject « Board ».

Elles sont placées avec un décalage initial pour centrer l'échiquier sur l'écran, puis la distance est multipliée par le nombre de case déjà initié dans la rangée pour la position X et dans la colonne pour la position Y.

La couleur de la case est basée sur le nombre de rangées et de colonnes déjà existante. S'il est divisible par 2, la case sera claire, dans le cas inverse, la case sera foncée.

```
1 reference | ArthurCPNV, 28 minutes ago | 1 author, 1 change
public void InitializeBoard()
{
    for (int rank = 0; rank < boardSize; rank++)
    {
        for (int file = 0; file < boardSize; file++)
        {
            // Get the position in which the tile should be placed.
            RectTransform tilePrefabRectTransform = _tilePrefab.GetComponent<RectTransform>();
            float tileWidth = tilePrefabRectTransform.sizeDelta.x;
            float tileHeight = tilePrefabRectTransform.sizeDelta.y;
            Vector3 tilePosition = new Vector3(_xOffset + (tileWidth * rank), _yOffset + (tileHeight * file), 0);

            // Initialise the tile
            GameObject tile = Instantiate(_tilePrefab, tilePosition, Quaternion.identity);
            tile.transform.parent = transform;
            tile.name = (char)('a' + rank) + (file + 1).ToString();

            // Change the color of the tile alternating from dark to light squares
            if ((rank + file) % 2 == 0)
            {
                tile.GetComponent<Image>().color = darkSquareColor;
            }
            else
            {
                tile.GetComponent<Image>().color = lightSquareColor;
            }
        }
    }
}
```

Figure 9: Méthode de l'initialisation de l'échiquier

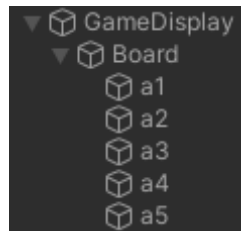
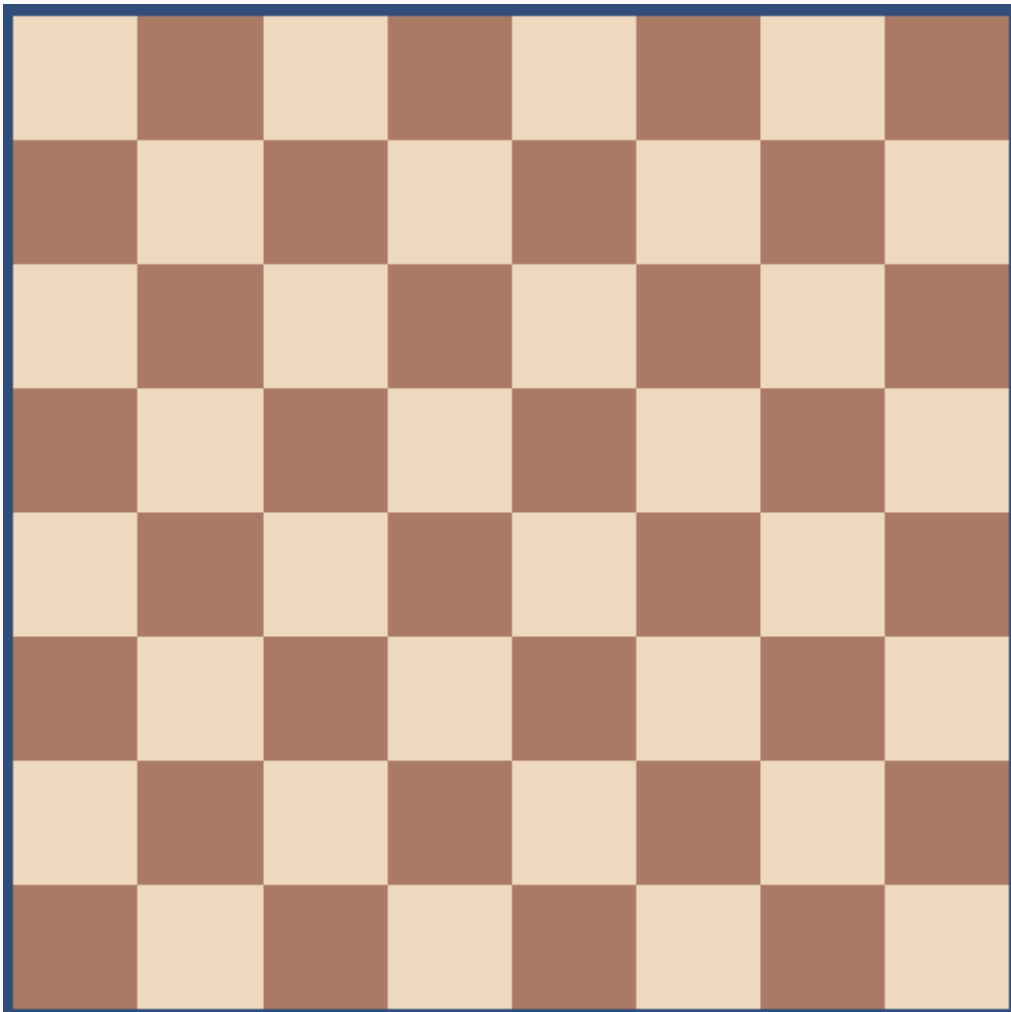


Figure 10: Arborescence des GameObjects dans la scène

Voici un exemple de l'échiquier une fois initialisé :



3.1.2.2 Position initiale

La position initiale des pièces est définie par un tableau en deux dimensions où chaque entrée décrit le type de la pièce qui devrait être placée, comme un pion blanc ou une tour noire.

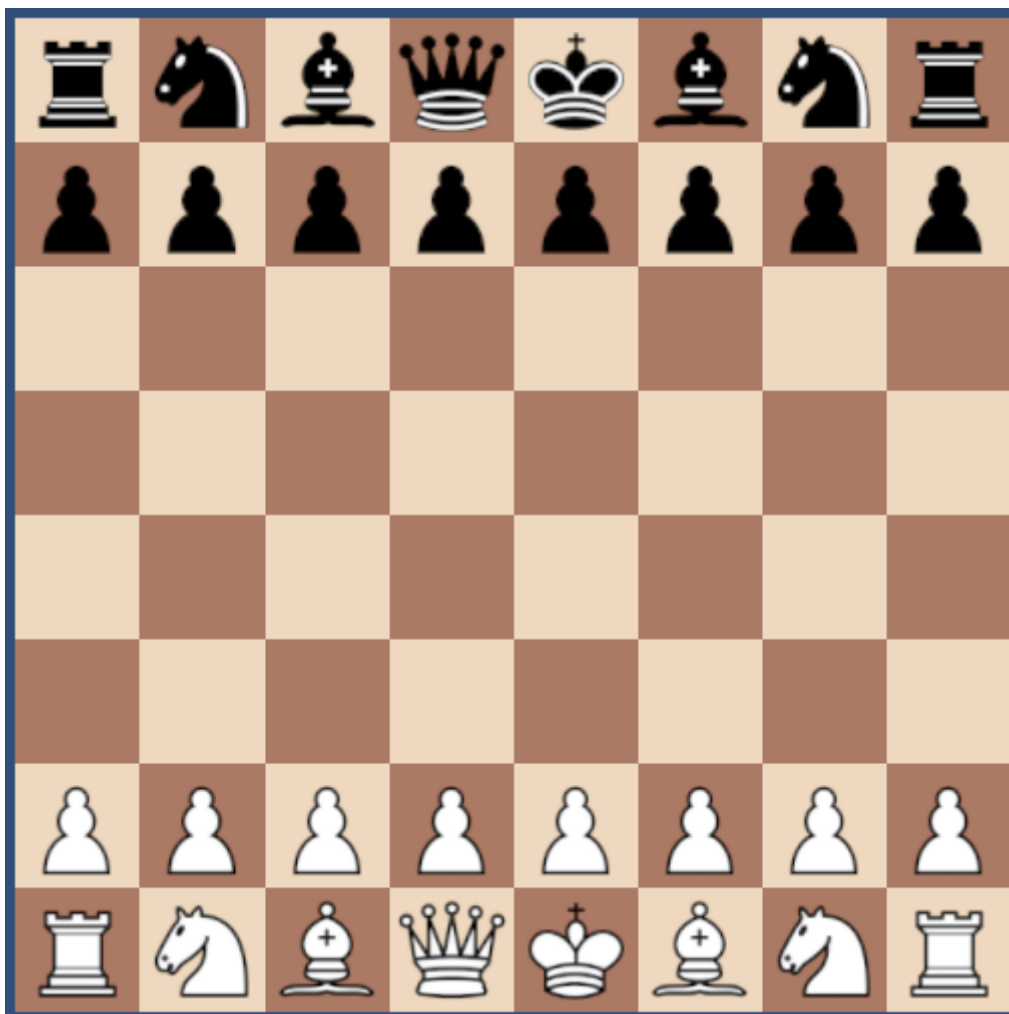
Voici un exemple de la table pour la position initiale pour un échiquier de 5 par 5 cases.

```
private static readonly PieceType[,] _initialBoardPosition = new PieceType[5, 5]
{
    {PieceType.BlackRook, PieceType.BlackKnight, PieceType.BlackBishop, PieceType.BlackQueen, PieceType.BlackKing},
    {PieceType.BlackPawn, PieceType.BlackPawn, PieceType.BlackPawn, PieceType.BlackPawn, PieceType.BlackPawn},
    {PieceType.None, PieceType.None, PieceType.None, PieceType.None, PieceType.None},
    {PieceType.WhitePawn, PieceType.WhitePawn, PieceType.WhitePawn, PieceType.WhitePawn, PieceType.WhitePawn},
    {PieceType.WhiteRook, PieceType.WhiteKnight, PieceType.WhiteBishop, PieceType.WhiteQueen, PieceType.WhiteKing}
};
```

L'exemple choisi permet de démontrer un exemple sans que le code soit illisible.

Le type « PieceType » est un énumérateur de chaque pièce qui existe dans les échecs.

Voici comment les pièces sont affichées sur l'échiquier basique :



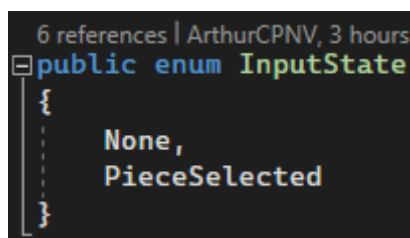
3.1.3 Le mouvement des pièces

3.1.3.1 *La gestion de saisie*

Le mouvement des pièces est réalisé par « l'InputManager ». Cette classe gère la saisie de la souris et la sélection des pièces et des cases pour bouger une pièce.

Avec cette classe, on peut sélectionner une pièce, bouger la pièce, sélectionner et annuler notre sélection.

Pour gérer la saisie, l'état de la saisie est constamment gardé en mémoire. Les états de la saisie sont définis dans un énumérateur comme ci-dessous :



```
6 references | ArthurCPNV, 3 hours
public enum InputState
{
    None,
    PieceSelected
}
```

Figure 11: L'énumérateur des états de la saisie actuel.

3.1.3.2 *La gestion des mouvements des pièces*

Les mouvements des pièces sont générés indépendamment des classes et doivent chacun définir leur logique de génération de mouvement. Cette génération est basée sur le rang et la file de la position de la pièce.

La génération des mouvements est généralement faite en prenant la case diagonale ou orthogonale, en vérifiant si cette case dispose d'une pièce de l'équipe adverse ou est vide.

Si la case est vide, elle est ajoutée comme une case vers laquelle la pièce peut se déplacer, et la vérification continue avec la prochaine case dans la même direction.

Si la case possède une pièce adverse, elle est également ajoutée comme une case possible vers laquelle on peut se déplacer, mais les cases suivantes ne sont pas vérifiées.

Si la case contient l'une de nos pièces, le processus de génération dans cette direction s'arrête et passe à la direction suivante.

Ceci est le processus de génération global pour chaque type de pièce. Certaines contraintes ou différentes règles peuvent modifier cette logique, comme les pions, le cavalier et le roi.

Il est à noter que la génération des mouvements ne prend pas en compte si le coup est légal mais seulement pseudo-légal. Ceci sera vérifié et les mouvements possibles seront modifiés une fois que tous les coups auront été générés.

Exemple de la génération des coups du pion, seulement pour l'avancement :

```
// Move forward
Tile moveTile = _gameTiles[_pieceFile + moveDirection, _pieceRank];

if (moveTile.OccupyingPiece == null )
{
    _validTilesToMove.Add(moveTile);

    if (_hasMoved == false)
    {
        moveTile = _gameTiles[_pieceFile + moveDirection * 2, _pieceRank];

        if (moveTile.OccupyingPiece == null)
        {
            _validTilesToMove.Add(moveTile);
        }
    }
}
```

Ici, on regarde la case devant le pion dépendant du « moveDirection ». Le « moveDirection » définit dans quelle direction le pion peut bouger. Ceci est nécessaire pour les pions de différentes couleurs.

Ensuite, on vérifie pour cette case s'il y a une pièce. Peu importe si la pièce nous appartient ou pas, car les pions ne peuvent pas prendre les pièces dans cette direction.

Finalement, si le pion n'a jamais bougé dans le jeu, cela signifie qu'il est dans la rangée initiale. Il peut alors bouger de deux cases en avant. On fait donc une vérification supplémentaire de deux cases en avant.

Décrire la réalisation "physique" de votre projet

- *Les répertoires où le logiciel est installé*
- *La liste de tous les fichiers et une rapide description de leur contenu (des noms qui parlent !)*
- *Les versions des systèmes d'exploitation et des outils logiciels*
- *La description exacte du matériel*
- *Le numéro de version de votre produit !*
- *Programmation et scripts: librairies externes, dictionnaire des données, reconstruction du logiciel - cible à partir des sources.*

3.2 Description des tests effectués

Pour chaque partie testée de votre projet, il faut décrire :

- *Les conditions exactes de chaque test*
- *Les preuves de test (papier ou fichier)*
- *Tests sans preuve : fournir au moins une description*

3.3 Erreurs restantes

S'il reste encore des erreurs :

- *Description détaillée*
- *Conséquences sur l'utilisation du produit*
- *Actions envisagées ou possibles*

3.4 Liste des documents fournis

Lister les documents fournis au client avec votre produit, en indiquant les numéros de versions

- *Le rapport de projet*
- *Le manuel d'Installation (en annexe)*
- *Le manuel d'Utilisation avec des exemples graphiques (en annexe)*
- *Autres...*

4 Conclusions

Développez en tous cas les points suivants :

- *Objectifs atteints / non-atteints*
- *Points positifs / négatifs*
- *Difficultés particulières*
- *Suites possibles pour le projet (évolutions & améliorations)*

5 Annexes

5.1 Résumé du rapport du TPI / version succincte de la documentation

5.2 Sources – Bibliographie

Liste des livres utilisés (Titre, auteur, date), des sites Internet (URL) consultés, des articles (Revue, date, titre, auteur)... Et de toutes les aides externes (noms)

5.3 Journal de travail

Date	Durée	Activité	Remarques

5.4 Manuel d'Installation

5.5 Manuel d'Utilisation

5.6 Archives du projet

Media, ... dans une fourre en plastique