

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 :

Component : Module de comportement attaché à un GameObject, ajoutant des fonctionnalités spécifiques.

Coup pseudo-légal : Un coup pseudo-légal est un coup aux échecs, qui correspond aux mouvements d'une pièce, sans vérifier si le déplacement de la pièce met en danger le roi.

GameObject : Un « GameObject » est l'élément de base dans Unity qui représente tout objet dans une scène. Il peut contenir divers composants, comme des scripts, des modèles 3D, des lumières, des caméras, etc., qui définissent son comportement et son apparence.

Méthodologie agile : La méthodologie agile est une approche de gestion de projet qui privilégie l'adaptabilité, la collaboration et les livraisons itératives et incrémentales. Elle se concentre sur la satisfaction du client grâce à des cycles de développement courts et à une réévaluation constante des priorités.

Méthodologie waterfall : La méthodologie Waterfall, ou en cascade, est une approche de gestion de projet linéaire et séquentielle où chaque phase du projet doit être complétée avant de passer à la suivante. Cette méthode est souvent critiquée pour son manque de flexibilité face aux changements et aux imprévus.

Moteur d'échecs : Un moteur d'échecs est un logiciel informatique qui analyse les positions sur l'échiquier et joue des coups en utilisant des algorithmes avancés. Il évalue les positions, calcule les meilleures séquences de coups et prévoit les réponses de l'adversaire.

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.

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.

Plugin Unity : Un plugin Unity est une extension ou un ensemble de fonctionnalités supplémentaires qui peuvent être ajoutées à un projet Unity. Les plugins peuvent fournir des outils, des bibliothèques ou des composants pour améliorer le développement de jeux, comme des moteurs physiques, des intégrations de services, etc.

Prise en passant : La prise en passant est un coup spécial aux échecs qui permet à un pion de capturer un pion adverse situé sur une case adjacente après que ce dernier a avancé de deux cases depuis sa position initiale. Ce coup doit être réalisé immédiatement après le déplacement du pion adverse.

Prefab : Un Préfabriqué est un modèle ou une instance réutilisable d'un « GameObject » ou d'un ensemble de « GameObjects » dans Unity. Il permet de créer et de gérer des objets de manière efficace et cohérente dans plusieurs scènes d'un projet.

Promotion : La promotion est une règle des échecs qui permet de transformer un pion en une autre pièce (dame, tour, fou ou cavalier) lorsqu'il atteint la dernière rangée de l'échiquier. La promotion en dame est la plus courante en raison de sa puissance.

Roque : Le roque est un coup spécial aux échecs où le roi et l'une des tours se déplacent simultanément. Il existe deux types de roque : le petit roque (côté roi) et le grand roque (côté dame). Le roque permet de mettre le roi en sécurité et de développer la tour.

SerializeField : Les « SerializeField » est un attribut en C# dans Unity qui permet de rendre une variable privée visible et modifiable dans l'inspecteur de l'éditeur Unity, sans avoir à la rendre publique dans le code.

Sprites : Les sprites sont des images 2D utilisées dans les jeux pour représenter des personnages, des objets, des arrière-plans, etc. Dans Unity, les sprites sont des éléments graphiques utilisés principalement dans le développement de jeux en 2D.

Transform : Le composant Transform dans Unity définit la position, la rotation et l'échelle d'un objet dans l'espace 3D ou 2D. Chaque objet dans Unity possède un Transform, qui est essentiel pour définir où et comment l'objet est placé dans la scène. GameObject.

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

User Interface (UI) : Ensemble d'éléments visuels et interactifs pour l'interaction utilisateur, comme les boutons et les menus.

Table des matières

Glossaire :	2
Analyse préliminaire	6
Introduction	6
Objectifs	6
Planification initiale	8
Analyse / Conception	9
Concept	9
Diagrammes de flux	9
Maquettes	14
Stratégie de test	16
Risques techniques	16
Planification	18
Dossier de conception	24
Diagramme de classes	24
Diagrammes de séquences	25
Réalisation	27
Dossier de réalisation	27
Mise en place	27
Limitations	27
L'échiquier	28
Le mouvement des pièces	31
Les coups spéciaux	35
Description des tests effectués	37
Erreurs restantes	37
Liste des documents fournis	40
Conclusions	41
Objectifs non atteints	41
La promotion	41
Le chronomètre	41
Les conditions de fin de partie	42
Le fichier de récapitulation	42
Points positifs	42
Points négatifs	43
Difficultés rencontrées	43
Baisse de régime	43
Validation des mouvements	44
Les coups spéciaux	44
Suites possibles	44

Améliorer l'interface utilisateur.....	44
Ajouter un moteur d'échecs.....	46
Ajouter un mode en réseau	47
Ajouter un mode de jeu puzzle	47
Annexes.....	48
Résumé du rapport du TPI	48
Planification finale	49
Journal de travail.....	54
Manuel d'Installation.....	58
Manuel d'Utilisation.....	59
Sources	60
Sites internet.....	60
Archives du projet	62

Analyse préliminaire

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.

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
- Prévion 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

Planification initiale

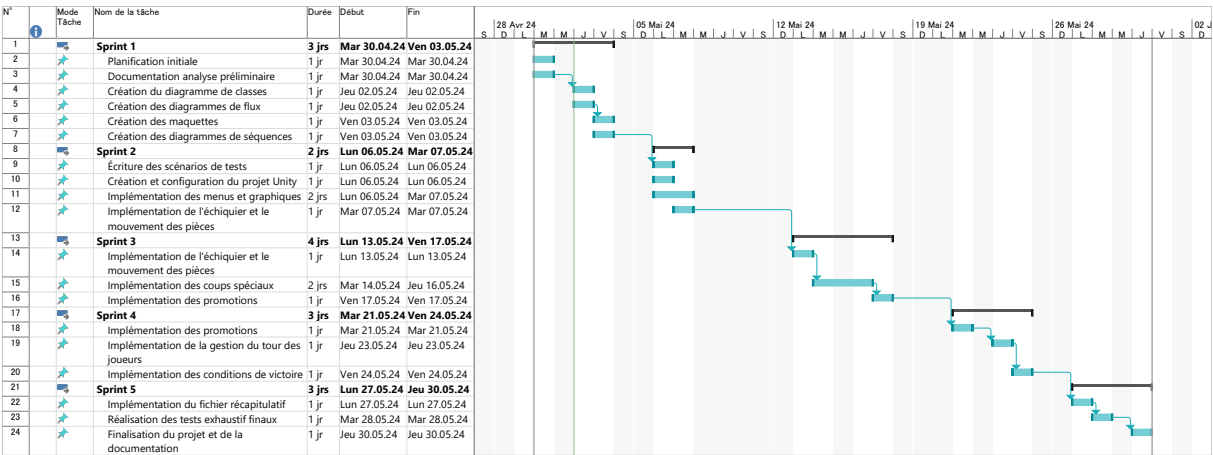


Figure 1: Planification initiale

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.

Analyse / Conception

Concept

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.

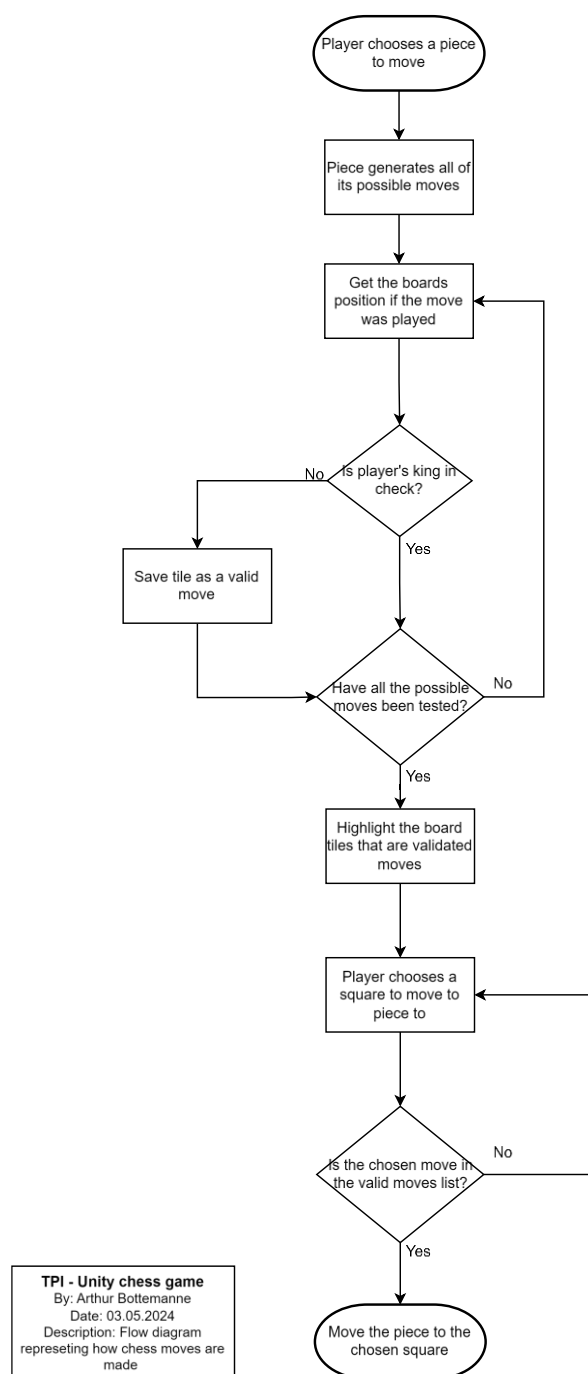


Figure 2: Diagramme de flux de la gestion du mouvement des pièces

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

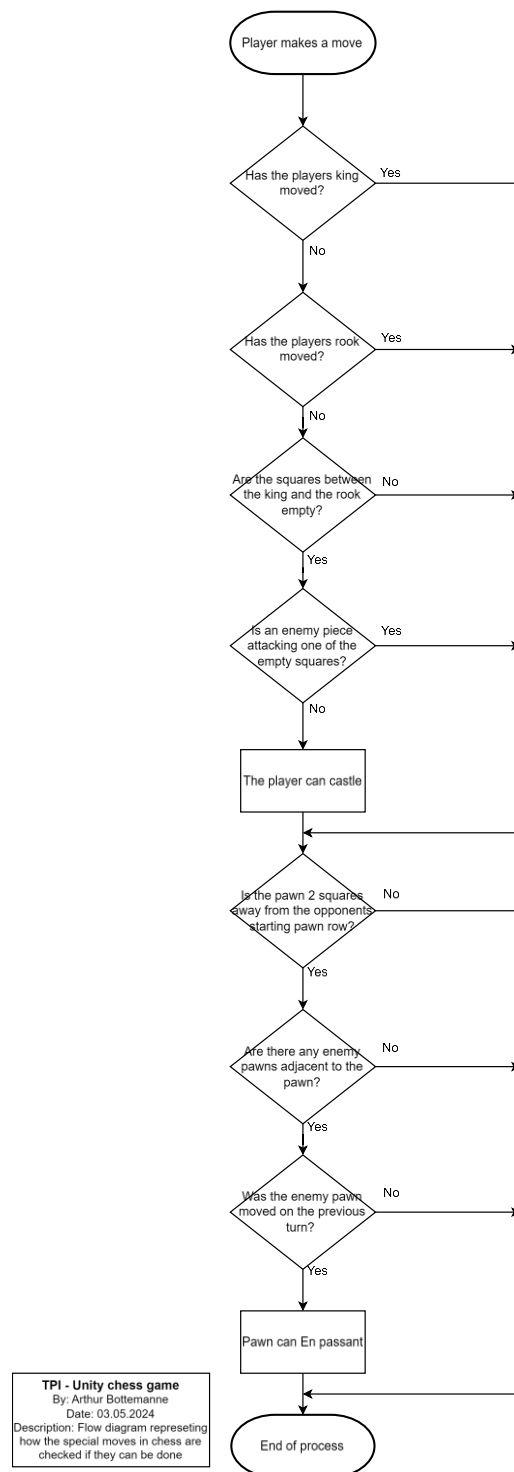


Figure 3: Diagramme de flux de la gestion des mouvements spéciaux

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

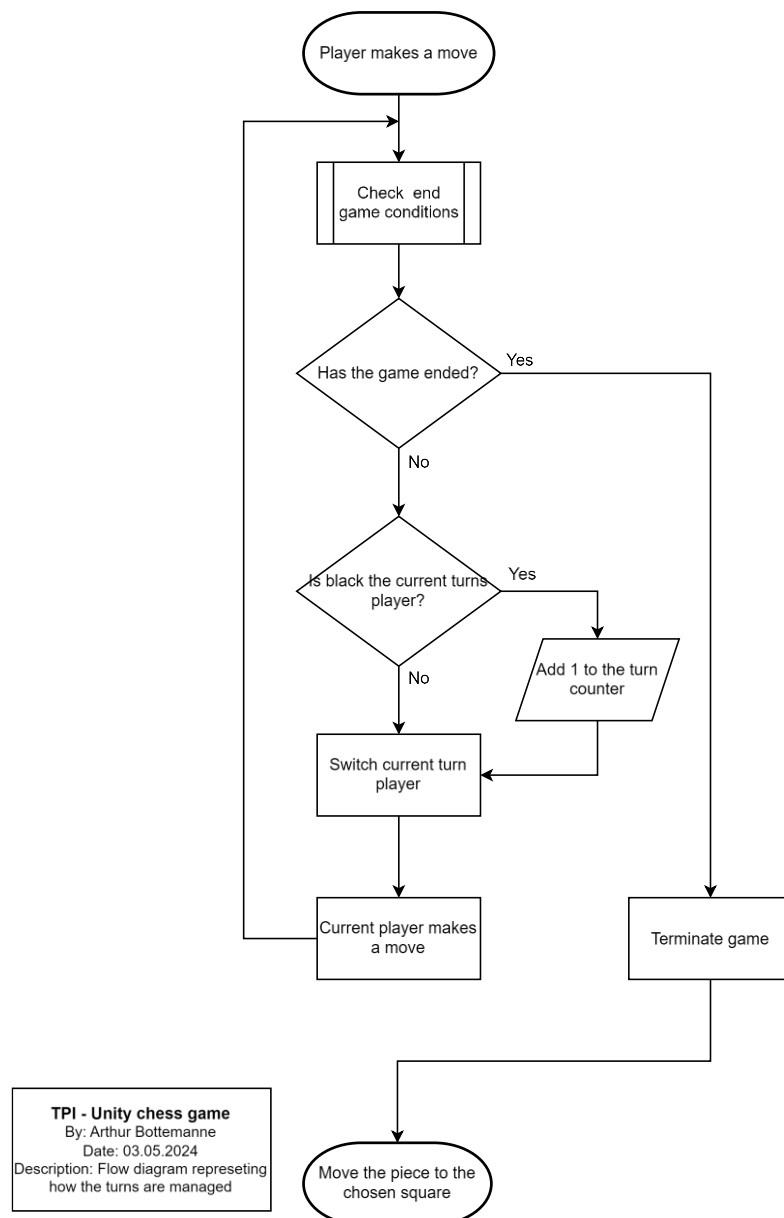


Figure 4: Diagramme de flux de la gestion du tour des joueurs

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

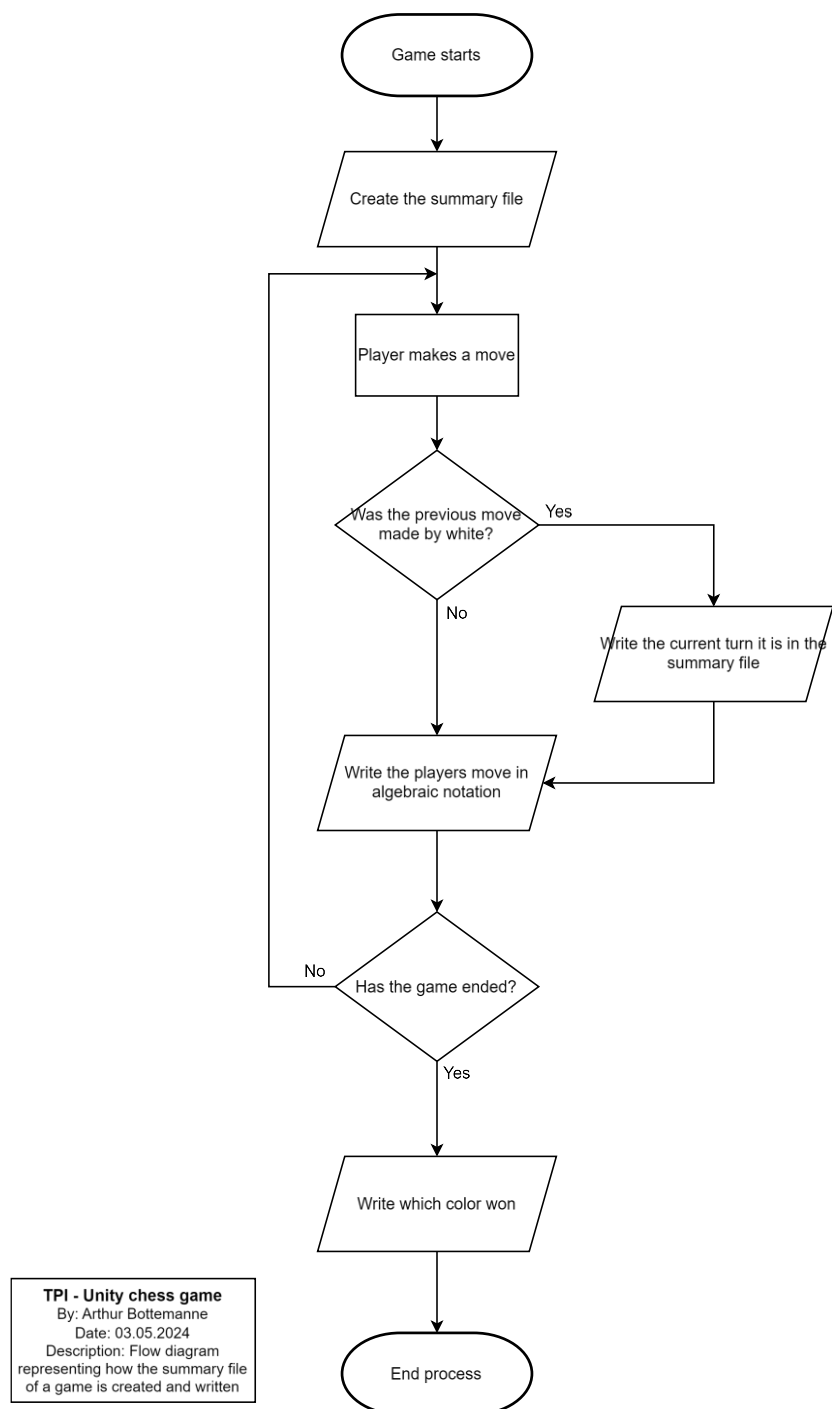


Figure 5: Diagramme de flux de la gestion du fichier récapitulatif

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

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 :

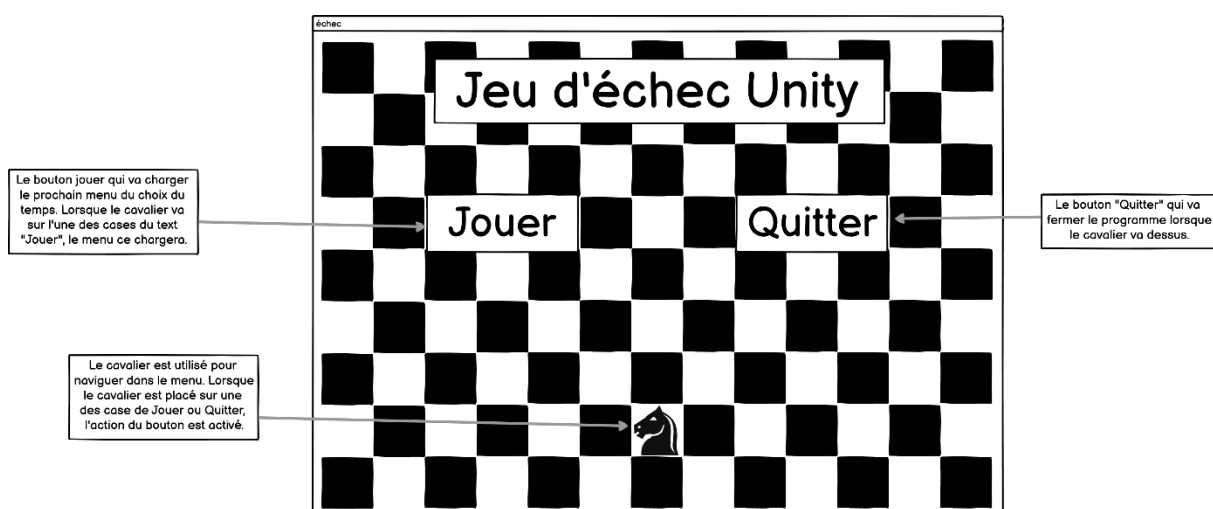


Figure 6: Maquette du menu principale

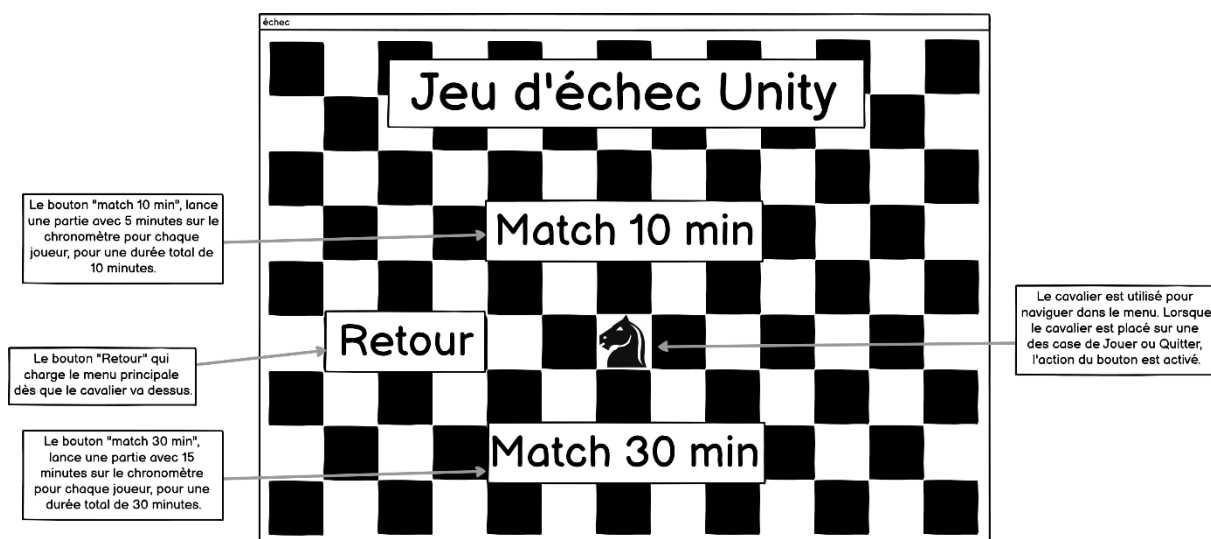


Figure 7: Maquette du menu du temps d'une partie

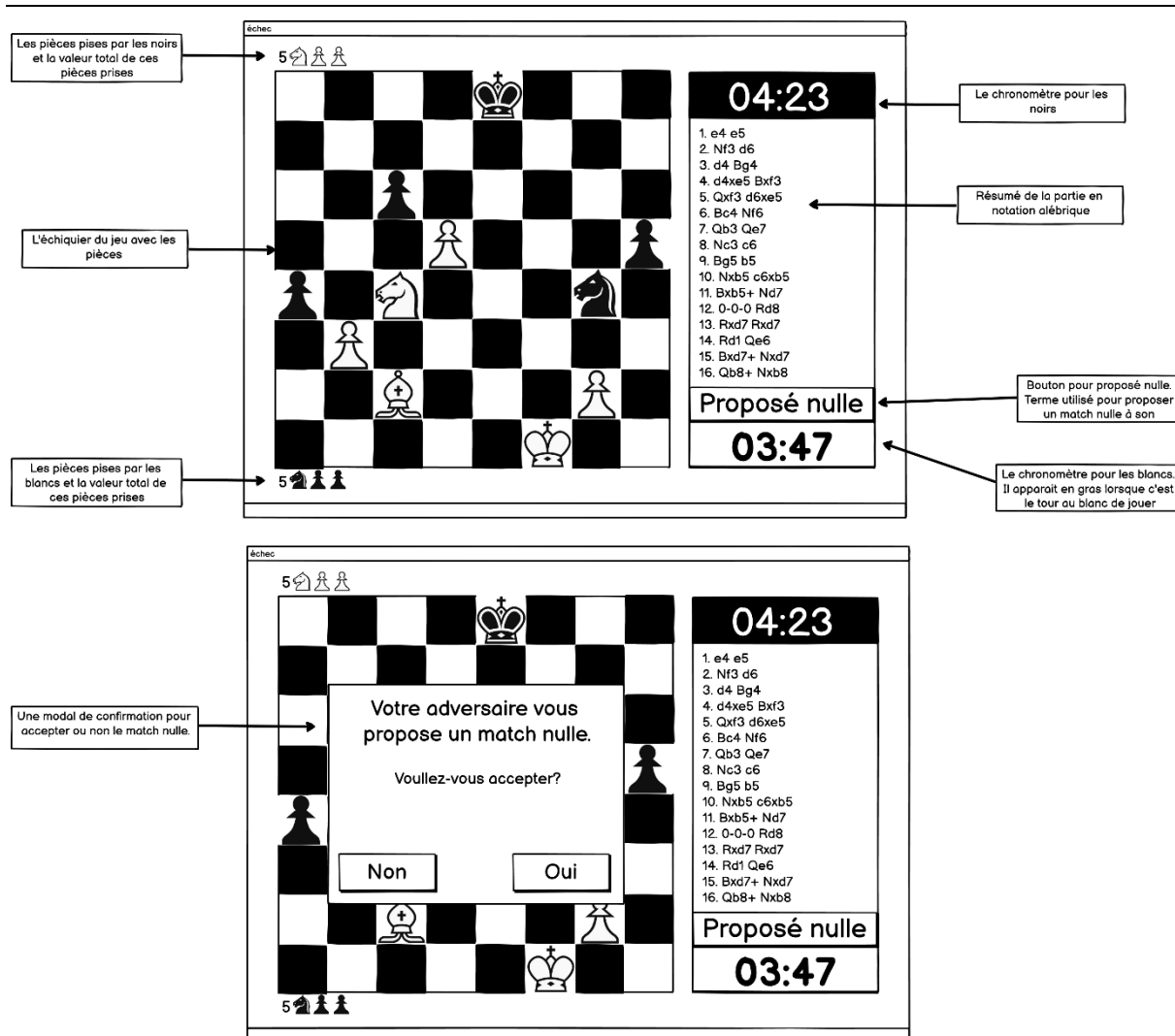


Figure 9: Maquette d'une partie en cours

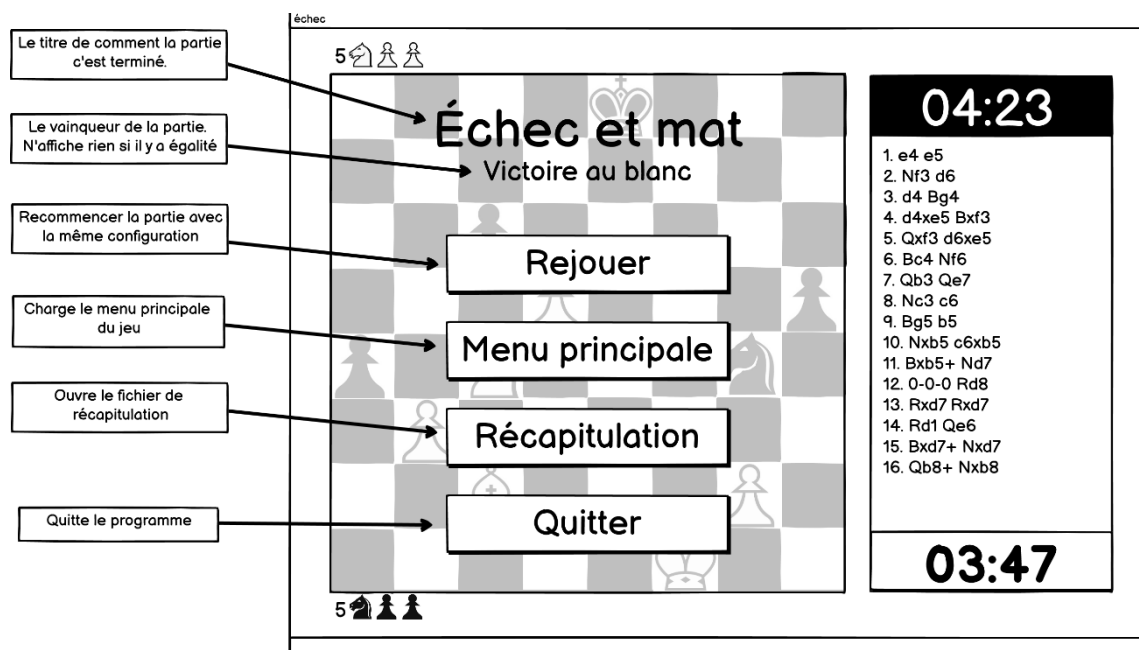


Figure 8: Maquette de la fin d'une partie

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.

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.

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.

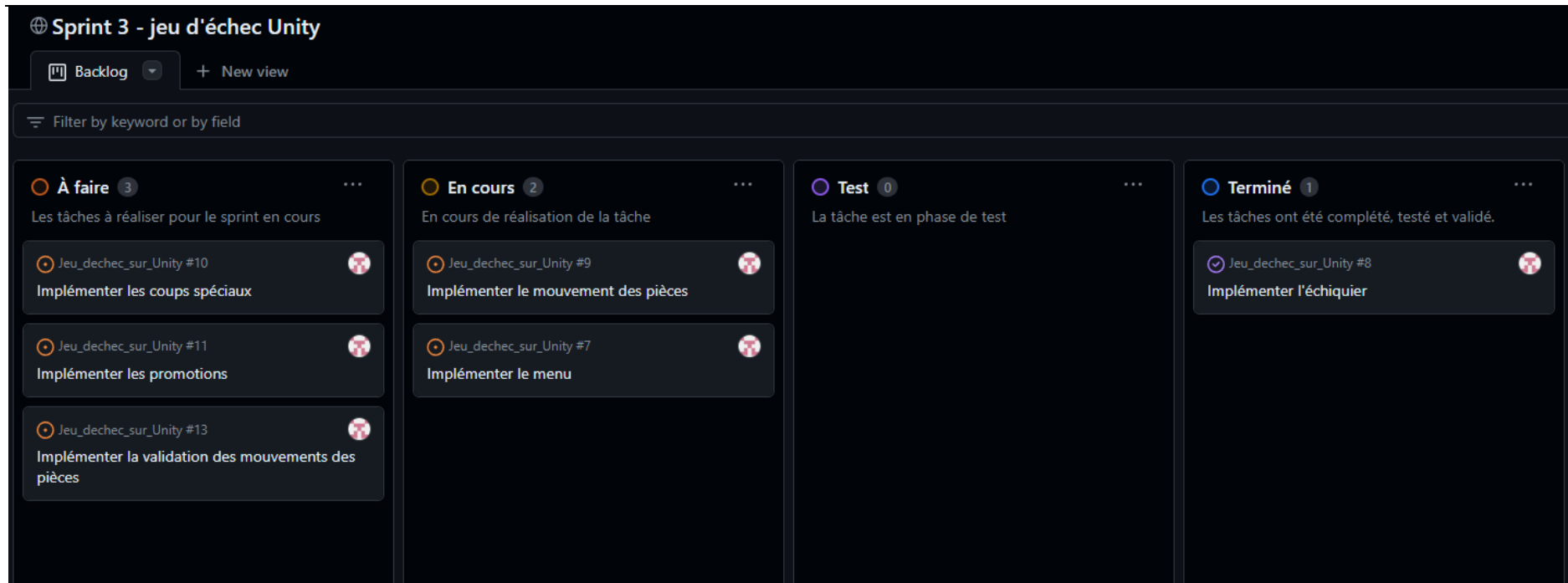


Figure 10: Exemple d'un sprint et de ces issues sur GitHub

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.

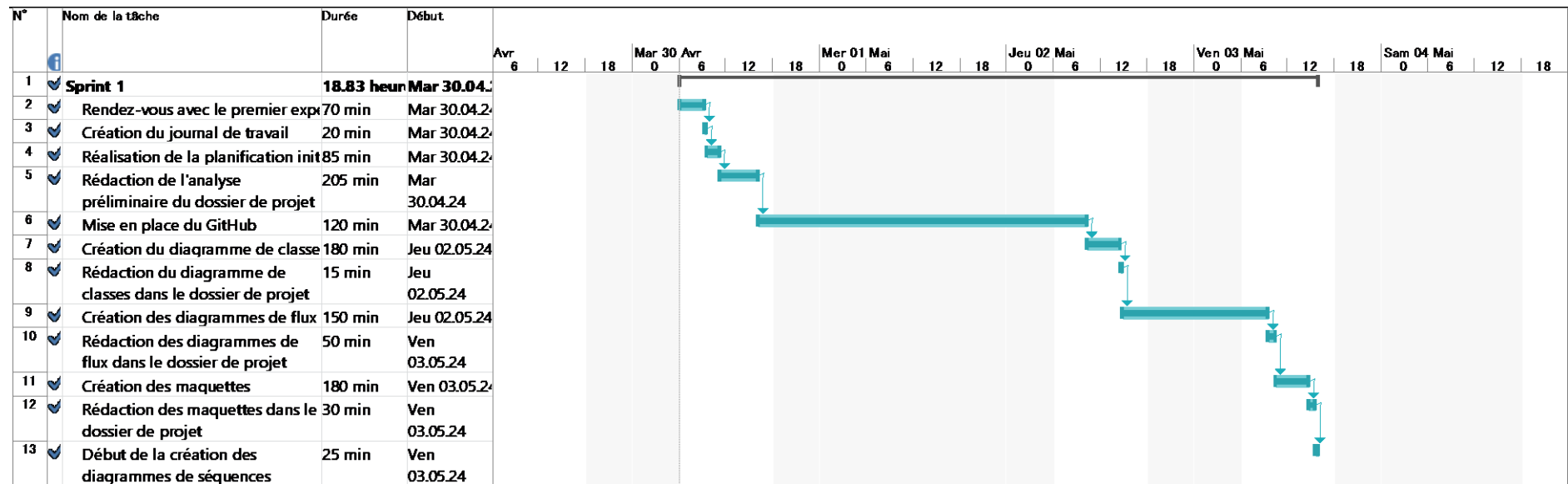


Figure 11: Sprint 1 de la planification détaillée

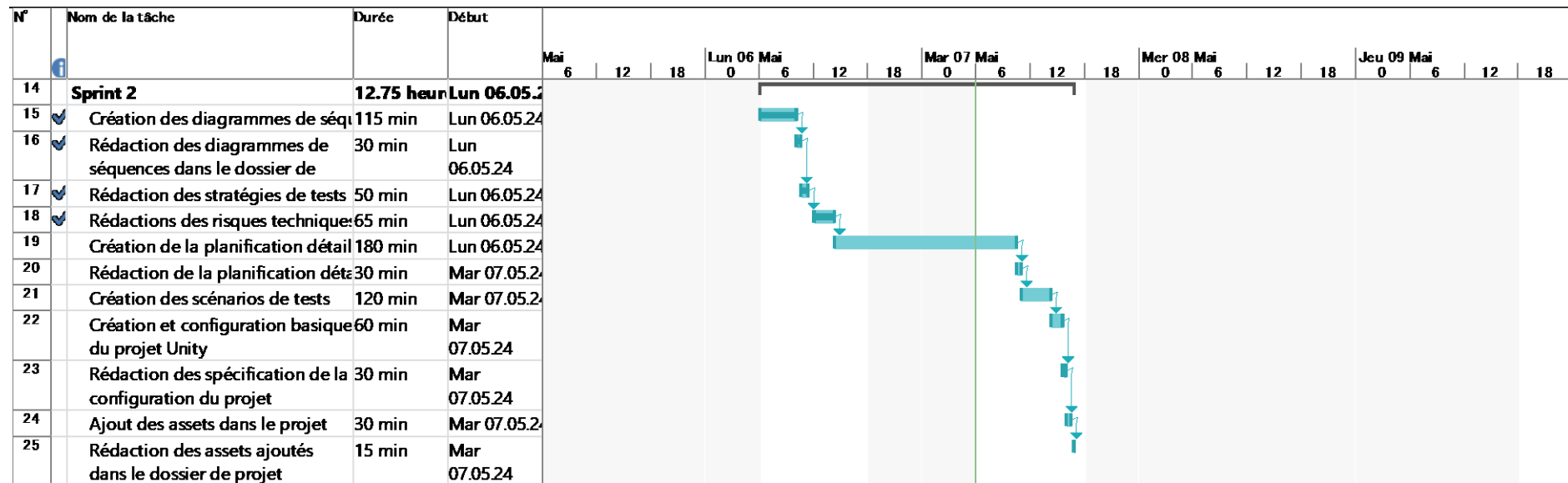


Figure 12: Sprint 2 de la planification détaillée

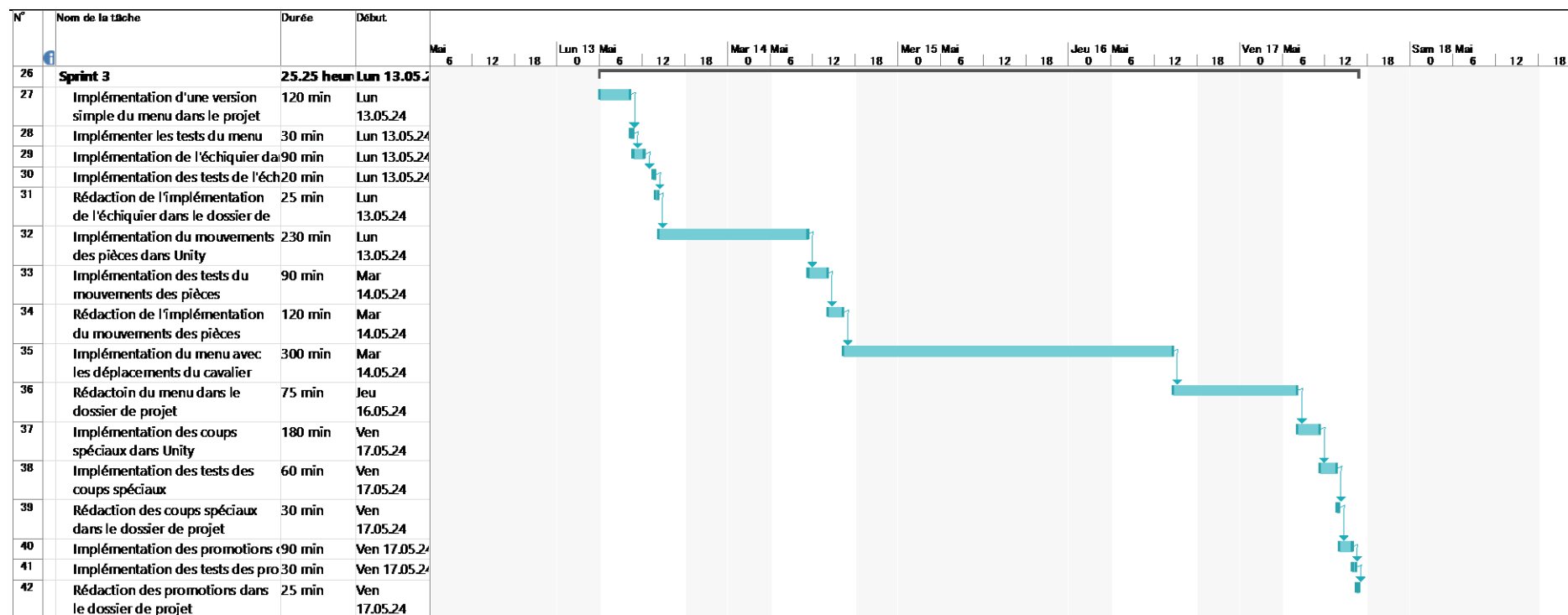


Figure 13: Sprint 3 de la planification détaillée

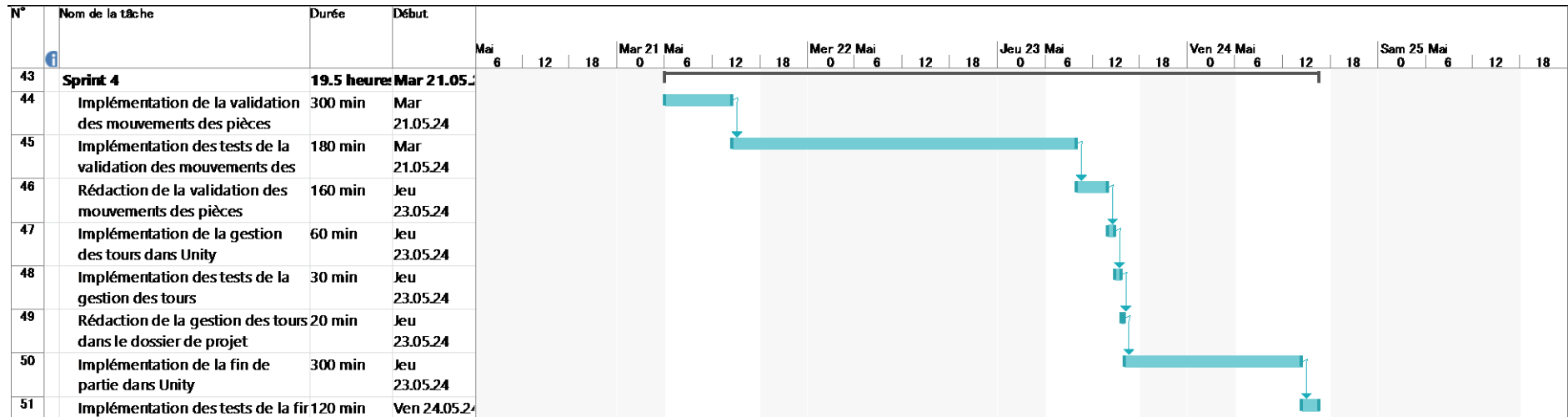


Figure 14: Sprint 4 de la planification détaillée

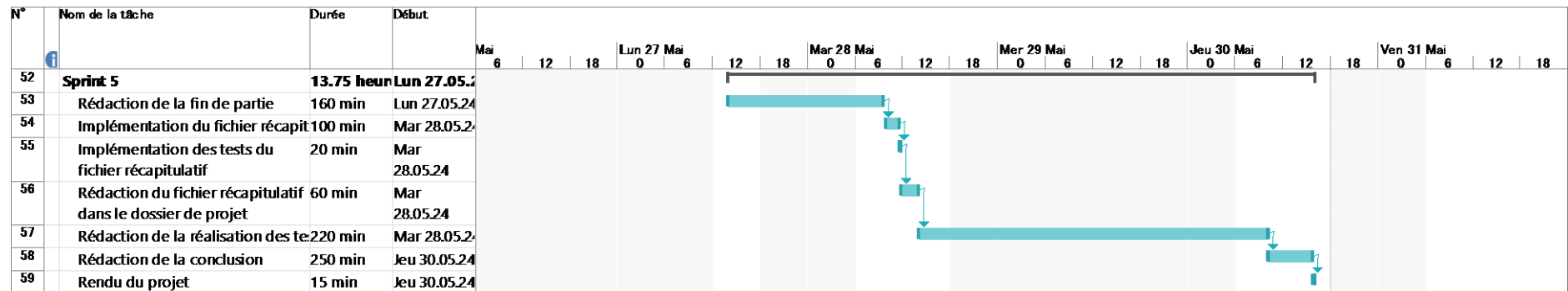


Figure 15: Sprint 5 de la planification détaillée

Dossier de conception

Diagramme de classes

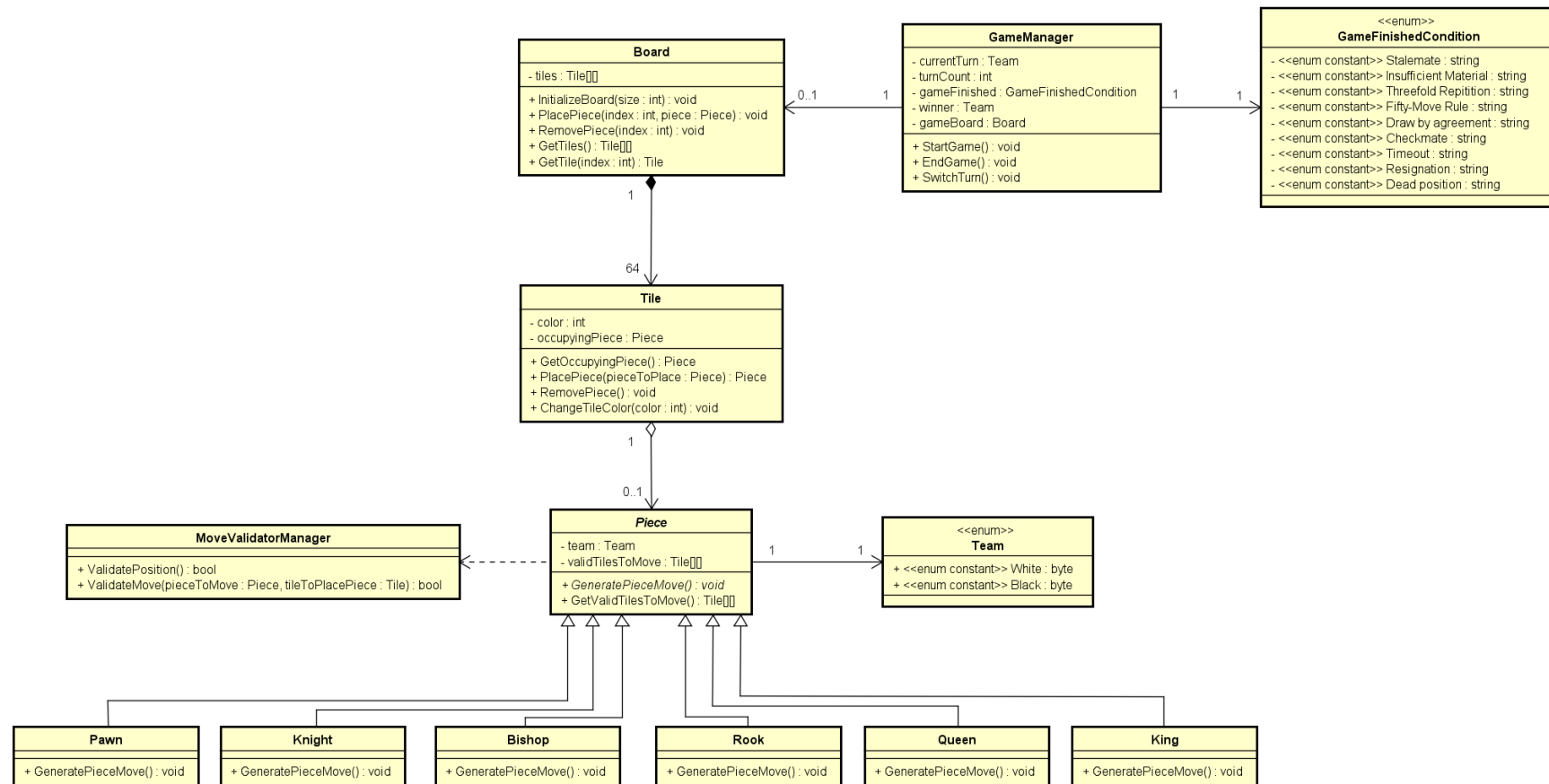


Figure 16: Diagramme de classes du projet

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.

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.

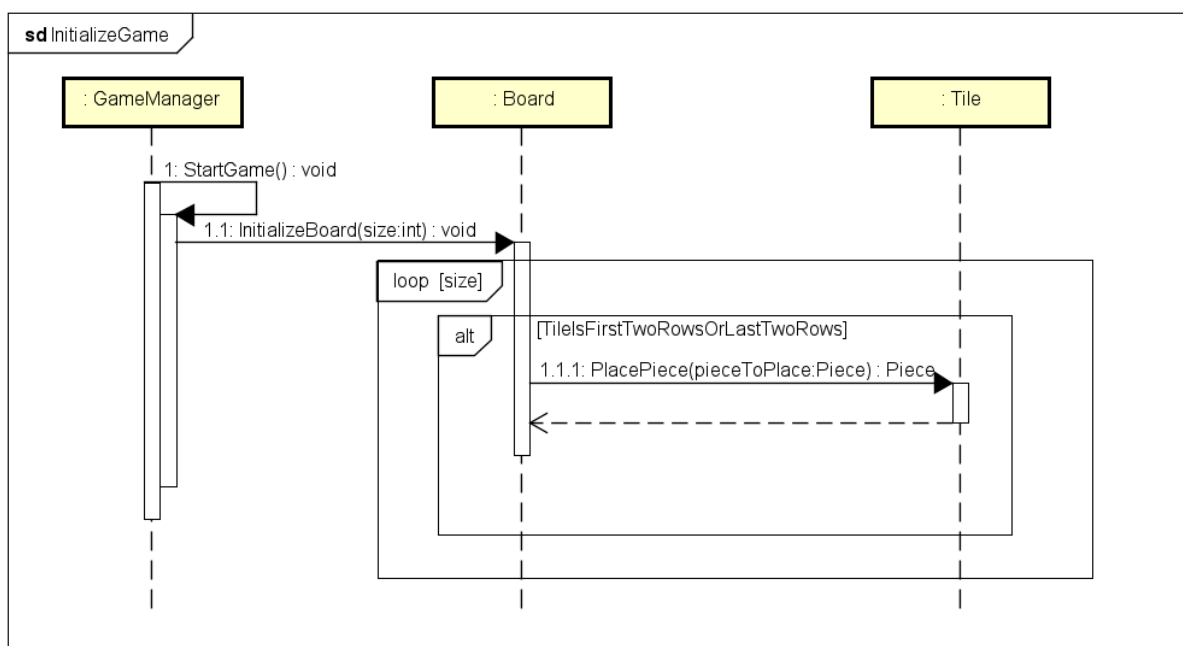


Figure 17: Diagramme de séquence du début d'une partie

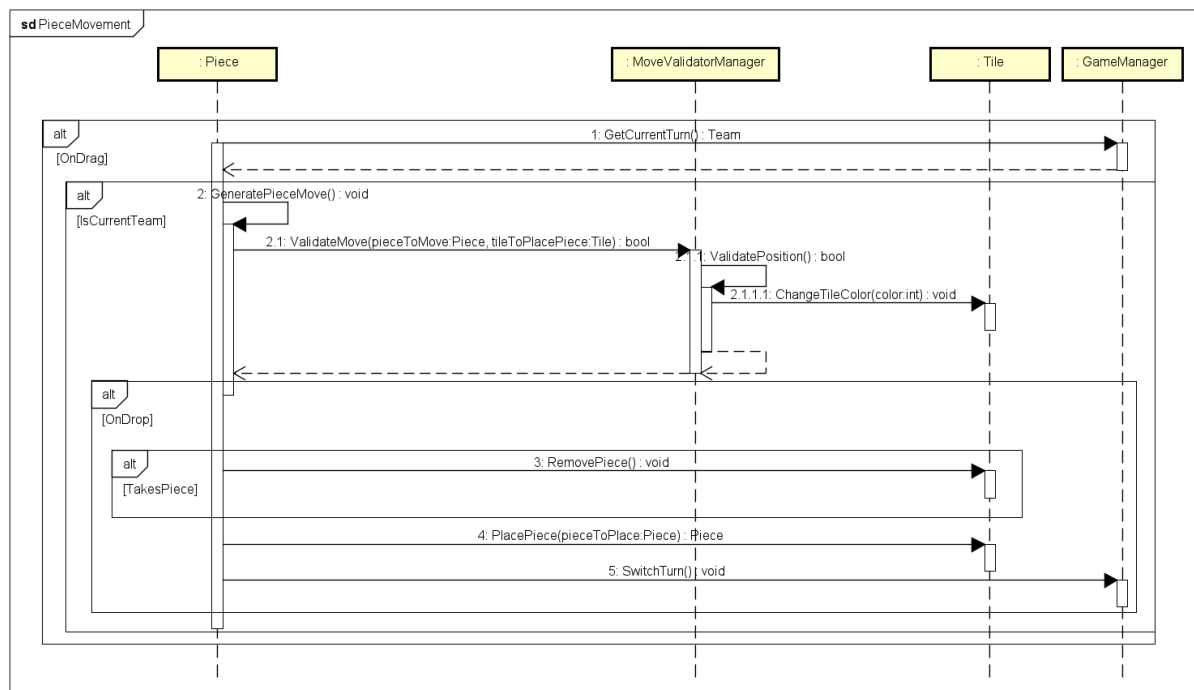


Figure 19: Diagramme de séquence de la fin d'une partie

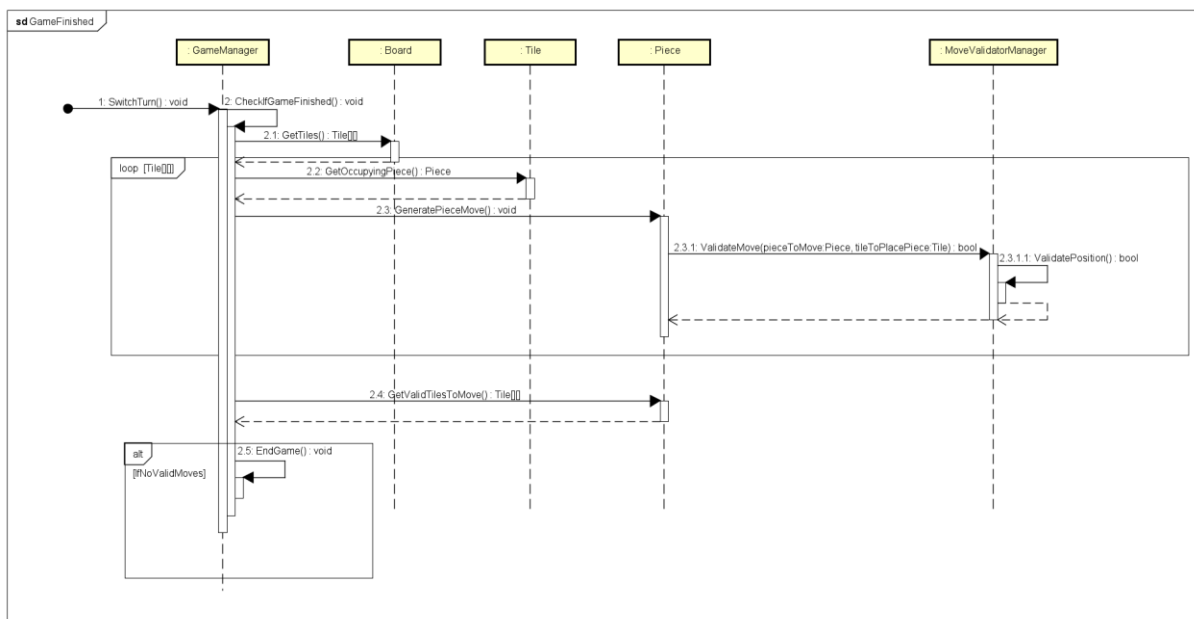


Figure 18: Diagramme de séquence du mouvement des pièces

Réalisation

Dossier de réalisation

Mise en place

Organisation des résultats du travail

Un répertoire GitHub a été créé pour assurer le stockage pour tous les fichiers (fichiers de documentation et fichiers du projet Unity).

On peut retrouver l'historique des versions du projet grâce aux commits GitHub et au minimum 1 commit a été fait chaque jour de travail sauf pour le 13 mai.

L'ergonomie de la place de travail physique est fournie par le CPNV et met à disposition un seul écran. Pour l'ergonomie informatique, une structure de fichier organisée en lien avec le TPI a été mise en place, comme c'est visible sur GitHub.

Projet Unity

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

La configuration Unity est laissée par défaut, sauf pour la résolution qui est définie avec un écran fenêtré de 1000x750 pixels.

Les plugins par défaut ont été installés par Unity, et le plugin « Input System » a été installé pour une implémentation plus simple du clic de la souris.

Limitations

Moteur de Jeu Unity 2D

Unity est un moteur de jeu puissant ; cependant, il se peut que pour des anciens appareils, la performance nécessaire de Unity soit trop coûteuse. Toutefois, dans le cadre de ce projet de TPI, cela ne devrait pas être un problème.

Certains plugins de Unity peuvent aussi ne pas être compatibles avec la version choisie. Mais, en raison de sa simplicité, ce projet utilise peu de plugins.

Dans certains cas de figure, la taille de l'écran et celle de la fenêtre du jeu peuvent poser des problèmes comme la disparition de certains graphiques.

Le build du jeu sera seulement fait pour le système d'exploitation Windows. Pour Linux et Mac, aucune précaution n'est prise pour que le programme soit fonctionnel sur ces machines en raison des exigences du cahier des charges.

L'échiquier

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 cases 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 existantes. S'il est divisible par deux, la case sera claire, dans le cas inverse, la case sera foncée.

```
/// <summary>
/// Initialize the tiles of the board
/// </summary>
1 reference | ArthurCPNV, 20 hours ago | 1 author, 9 changes
public void InitializeBoard()
{
    // cache the size of the tile prefab
    RectTransform tilePrefabRectTransform = tilePrefab.GetComponent<RectTransform>();
    float tileWidth = tilePrefabRectTransform.sizeDelta.x;
    float tileHeight = tilePrefabRectTransform.sizeDelta.y;

    for (int rank = 0; rank < boardSize; rank++)
    {
        for (int file = 0; file < boardSize; file++)
        {
            // Get the position in which the tile should be placed
            Vector3 tilePosition = new Vector3(XOffset + (tileWidth * rank), YOffset + (tileHeight * file), 0);

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

            // Change the color of the tile alternating from dark to light squares
            Image tileImage = tileObject.GetComponent<Image>();
            tileImage.color = ((file + rank) % 2 == 0) ? darkSquareColor : lightSquareColor;

            // Add tile to the tiles array
            Tile tileScript = tileObject.GetComponent<Tile>();
            _tiles[file, rank] = tileScript;
            tileScript.DefaultColor = tileImage.color;
        }
    }

    InitializePieces();
}
```

Figure 20: L'implémentation de la méthode "InitializeBoard" dans la classe "Board"



Figure 21: Les GameObjects de l'échiquier initialisé

Voici un exemple de l'échiquier une fois initialisé :

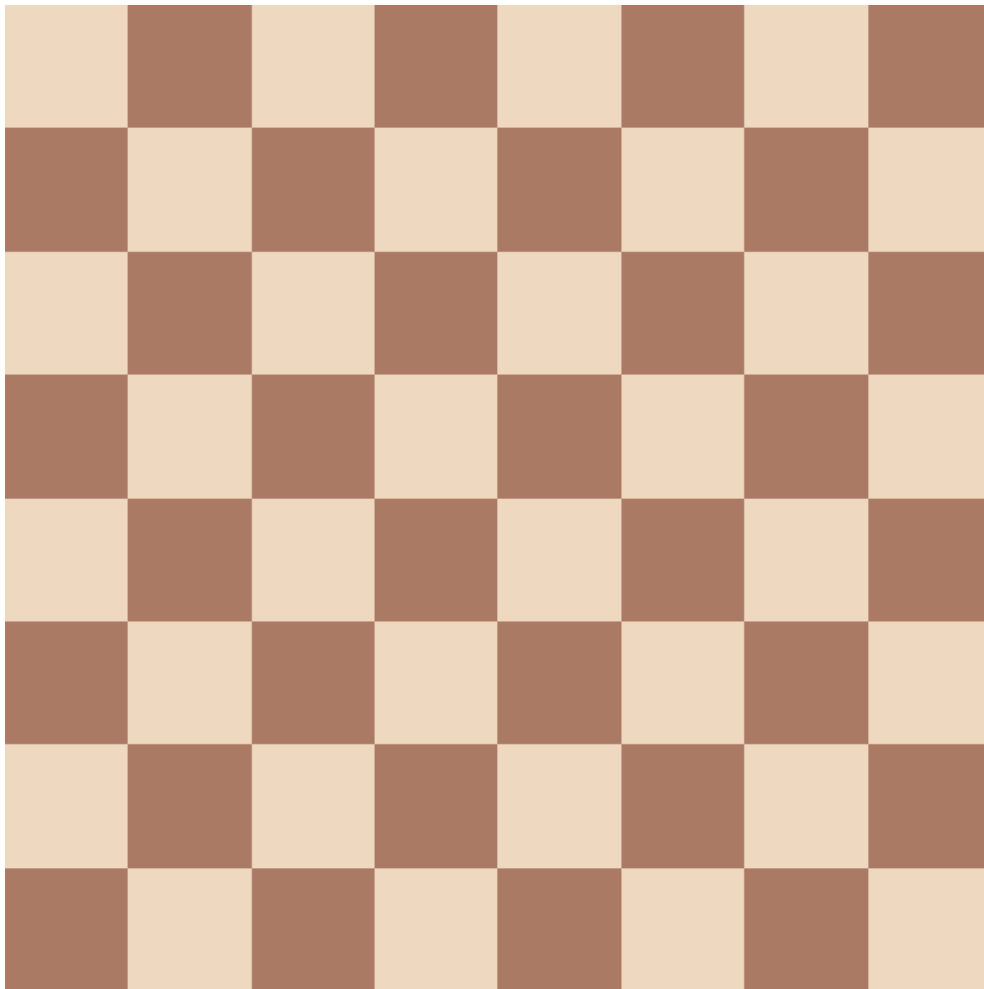


Figure 22: Le rendu graphique de l'échiquier initialisé

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}
};
```

Figure 23: Exemple de la représentation de la variable "_initialBoardPosition"

L'exemple choisi est celui d'un échiquier de 5 cases par 5, car l'image prendrait trop de place pour représenter le code réel d'un échiquier de 8 par 8.

Ce choix 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 :



Figure 24: Le rendu graphique de l'échiquier et des pièces initialisé

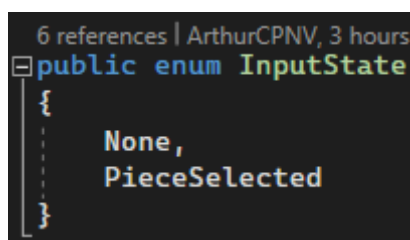
Le mouvement des pièces

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, annuler notre sélection et bouger la pièce.

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 25: Les états possibles de la saisie

La génération des mouvements des pièces

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

Il y a deux catégories de pièces : les pièces dites glissantes et celles qui ne le sont pas.

La génération des mouvements est généralement effectuée en prenant en compte :

- Les directions dans lesquelles la pièce peut se déplacer
- En implémentant la logique pour une pièce glissante ou non glissante
- En vérifiant les cases pour voir si elles sont vides ou occupées par une autre pièce sur le carré

Pendant la génération des coups possibles d'une pièce, nous partons de la case de la pièce et nous vérifions les cases suivantes dans les directions définies par la pièce. Pour cela, on prend en compte les éléments suivants :

- Si la case est vide, elle est ajoutée comme une case vers laquelle la pièce peut se déplacer. La vérification continue avec la prochaine case dans la même direction si la pièce est glissante.
- Si la case possède une pièce adverse (sauf pour le pion), 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.

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 cavalier :

```
for (int i = 0; i < _directions.Length; i++)
{
    (int, int) direction = _directions[i];

    fileToMove = _pieceFile;
    rankToMove = _pieceRank;

    fileToMove += direction.Item1;
    rankToMove += direction.Item2;

    if (Board.IsInBoardLimits(fileToMove, rankToMove))
    {
        movePositionPiece = gamePieces[fileToMove, rankToMove];
        moveTile = _gameTiles[fileToMove, rankToMove];

        if (movePositionPiece == null || movePositionPiece.Team != _team)
        {
            _validTilesToMove.Add(moveTile);
        }
    }
}
```

Figure 26: Implémentation de la génération des mouvements des pièces glissantes

La variable « `_directions` » définit les index de déplacement d'une case à une autre. Elle est initialisée dès qu'une pièce est créée.

On vérifie si le déplacement se fait dans les limites du plateau, et si la case de destination est vide ou occupée par une pièce de l'équipe adverse. Si c'est le cas, on ajoute cette case parmi les cases possibles où le cavalier peut se déplacer.

Pour les pièces glissantes, une boucle « do, while » est implémentée pour chaque direction de la pièce jusqu'à ce que la direction soit entièrement explorée.

Validation des mouvements générés

Quand une pièce a généré tous ses coups possibles, une vérification de tous les mouvements générés est ensuite effectuée.

Cette validation est effectuée en simulant la position après le coup et en vérifiant si le roi de la même couleur que la pièce se retrouve en échec. Cela garantit que tous les coups sont légaux.

Le problème avec cette implémentation, c'est qu'elle ne donne pas beaucoup de flexibilité. Cette validation fonctionne seulement pour les mouvements simples, d'une pièce qui se déplace d'une case à une autre. Les coups complexes comme le roque ou le coup en passant ont besoin de vérifications supplémentaires.

Le mouvement d'une pièce

Le mouvement des pièces est géré par la classe « Board ». Dans cette classe, il y a la méthode « MovePiece » qui accepte trois paramètres :

- « departureTile » qui décrit la case actuelle de la pièce
- « destinationTile » qui décrit la case où déplacer la pièce
- « switchTurns » qui définit si le tour du joueur doit changer après le mouvement de la pièce

Dans cette méthode, certaines vérifications pour les coups spéciaux sont faites comme :

- La détection de la possibilité d'une prise en passant
- Si le coup est une prise en passant
- Si une tour a bougé
- Si le roi peut effectuer un mouvement de roque

Ces vérifications sont nécessaires car le tableau des coups possibles pour une pièce ne décrit que la case vers laquelle la pièce peut se déplacer. Toutes les autres informations, telles que le roque ou la prise en passant, ne sont pas prises en compte. Elles doivent donc être vérifiées pendant le mouvement d'une pièce, ce qui affecte l'évolutivité, la clarté et les performances du programme.

```
// Move the piece from the departure tile to the destination tile
destinationTile.RemovePiece();
destinationTile.PlacePiece(pieceObject, isPromoting);
departureTile.RemovePiece();

// End of turn logic
pieceScript.ResetGeneratedMoves();
updatePiecesList();

if (switchTurns)
{
    _gameManager.SwitchTurn();
}
```

Figure 27: Implémentation du mouvement des pièces

Dans cette image, on peut voir comment une pièce est déplacée. La logique principale de la méthode consiste à retirer la pièce de la case cible puis à placer la pièce que l'on souhaite bouger à cet emplacement. Ensuite, le « GameObject » de la pièce qui reste sur la case de départ est supprimé.

Après cette logique, les variables sont mises à jour et le tour passe au joueur adverse dépendant du paramètre « switchTurns ».

La raison pour laquelle le paramètre « switchTurns » est utilisé est que le roque exécute cette méthode deux fois, pour déplacer d'abord le roi, puis la tour. Sans ce paramètre, le joueur peut encore jouer après avoir effectué le roque.

Améliorations possibles

Si j'avais plus de temps, j'envisagerais de changer la façon dont les coups sont définis. Au lieu de stocker un tableau avec les cases possibles où la pièce peut se déplacer, je créerais un tableau qui accepte des objets. Chaque objet contiendrait la case où la pièce peut se déplacer, ainsi que deux autres paramètres optionnels indiquant, d'une part, les pièces à retirer et, d'autre part, le déplacement d'une autre pièce.

Cette implémentation permet d'ajouter facilement la prise en passant et le roque, et rend le programme évolutif si l'on souhaite ajouter d'autres règles.

Les coups spéciaux

Le roque

Le roque est géré par la classe « King » et nécessite les attributs suivants pour garder en mémoire la possibilité de roque :

```
private bool _canCastleQueenSide = true;
private bool _canCastleKingSide = true;

private Tile _kingSideRook = null;
private Tile _queenSideRook = null;
```

Figure 28: les attributs de la classe du roi pour le roque

Ces attributs définissent les tours (on parle ici des pièces) pour le roque et si le roi peut roque dans une direction ou l'autre.

La possibilité du roque est vérifiée à chaque génération de mouvement du roi. La méthode « CheckCastleMoves » vérifie d'abord si le roi a le droit de roque d'un côté ou de l'autre. Ensuite, le programme vérifie qu'il n'y ait aucune pièce entre le roi et la tour. Si c'est le cas, la case à deux index de décalage par rapport au roi est ajoutée comme coup possible.

Lorsque le roi bouge, la méthode « CheckIfMoveIsCastling » vérifie si le coup joué est un roque du roi. Si c'est le cas, la tour correspondante est également déplacée à côté du roi.

Si le roi est déplacé, les droits de roque des deux côtés sont perdus. Si l'une des tours bouge, seul le droit de roque du côté de la tour est perdu pour le roi.

Prise en passant

La prise en passant est gérée par la classe « Pawn ».

Lorsqu'un pion avance de deux cases, les cases adjacentes au pion sont vérifiées pour voir si elles comportent un pion de l'équipe adverse. Si c'est le cas, le premier pion indique la case où il peut se déplacer pour la prise en passant, ainsi que la pièce adverse à capturer en cas de prise en passant.

```
/// <summary>
/// Define the pawns variables for en passant.
/// </summary>
/// <param name="canTakeTile">The tile which the pawn can move to take en passant.</param>
/// <param name="pieceToTake">The piece to take if the pawn takes en passant.</param>
2 references | ArthurCPNV, 5 days ago | 1 author, 1 change
public void SetEnPassant(Tile canTakeTile, Piece pieceToTake)
{
    _canTakeEnPassantTile = canTakeTile;
    _pieceToTakeEnPassant = pieceToTake;
}
```

Figure 29: Implémentation de la définition des variables en passant

Ici, les variables définissent d'abord la case où le pion peut effectuer la prise en passant, puis la pièce à retirer lorsque le pion réalise cette prise en passant.

Après chaque coup joué, les variables pour la prise en passant sont réinitialisées à null pour chaque pion dans la position.

Promotion

La promotion est gérée par la classe « Pawn »

Lorsque le pion arrive à la fin du plateau, une nouvelle reine est créée sur la case du pion et le remplace.

Il n'est pas nécessaire de vérifier la fin du plateau par rapport à l'équipe du pion, car le pion ne peut pas se déplacer en arrière. Si un pion blanc pouvait reculer d'une case dès le début de la partie, il réaliserait sa promotion.

Dans l'implémentation de la promotion, il n'est pas possible de choisir en quelle pièce le pion sera promu. Cela est dû à un changement de priorités et à un manque de temps.

Améliorations possibles

Les problèmes de l'implémentation des coups spéciaux sont liés avec le problème de la validation des mouvements. Puisque le tableau de la génération des mouvements permet seulement l'information des cases possibles, cela rend l'implémentation des coups spéciaux plus complexes.

S'il était possible de décrire plus d'informations pour un coup, au-delà de simplement la case de destination de la pièce, la lisibilité et la compréhension du code seraient grandement améliorées.

Pour les promotions, l'amélioration est évidente, elle consiste à implémenter la possibilité de promouvoir le pion en différentes pièces. La manière dont cela pourrait être fait est décrit dans la partie des objectifs non atteints.

Description des tests effectués

Le temps limité mis à disposition n'a pas permis de réaliser les tests unitaires. Pour cette raison, seuls les tests manuels de bout en bout ont été effectués.

Ces tests ont été réalisés dans le même environnement : sur un ordinateur Windows 10 64 bits. Ces ordinateurs sont ceux du poste de travail du CPNV, ainsi que des ordinateurs personnels divers.

Erreurs restantes

La prise en passant

La validation des mouvements n'a pas été implémentée en prenant en compte la possibilité des mouvements comme la prise en passant où l'interaction est plus complexe.

De la manière dont les mouvements sont définis, seules la case de la pièce et la case de destination sont prises en compte. Si une pièce se trouve sur la case de destination, elle est simplement capturée.

Le problème avec la prise en passant est qu'elle capture effectivement une pièce, mais pas sur la case de destination. La validation de mouvement ne prend en compte que le déplacement d'une pièce et non les pièces capturées.

Cela pose donc un problème dans un cas précis : si les deux pions sont cloués au roi, alors la prise en passant n'est pas possible.

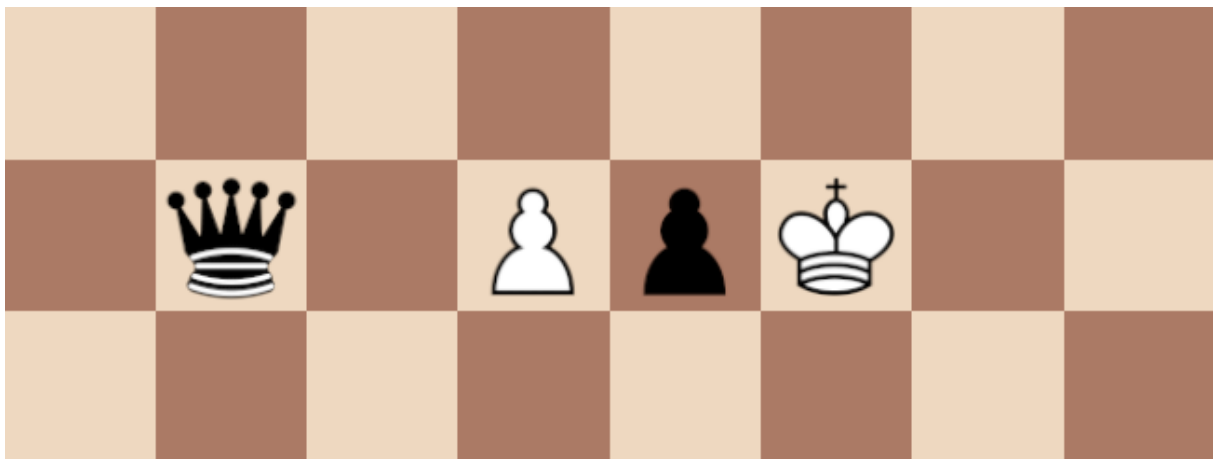


Figure 30: Exemple d'une position où les deux pions sont cloué pour la prise en passant

Dans cet exemple, le joueur noir vient juste de déplacer son pion de e7 à e5. C'est maintenant au tour des blancs de jouer. Le pion en d5 peut donc capturer le pion en e5 en passant. Cependant, s'il le fait, Le roi blanc serait en échec, ce qui est illégal selon les règles des échecs.

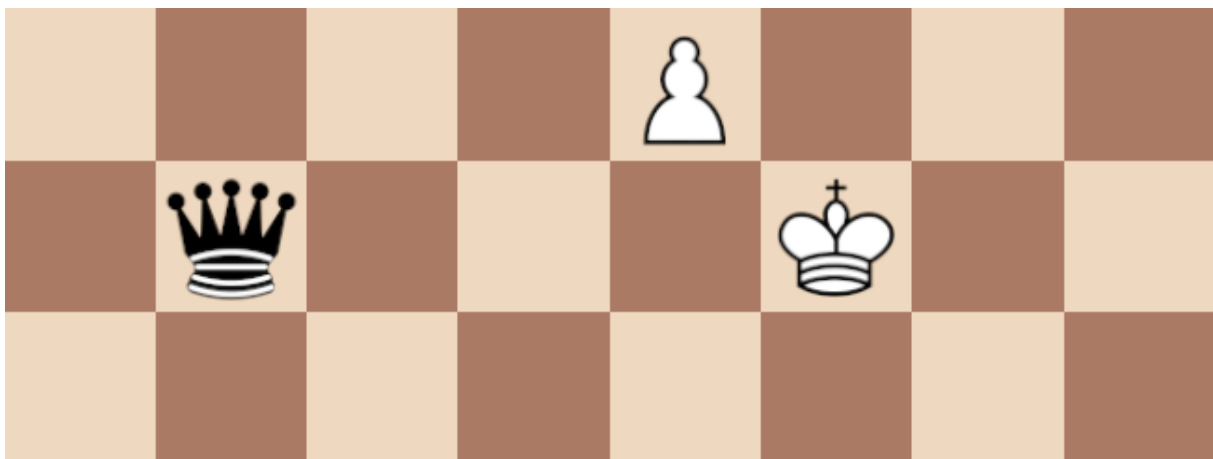


Figure 31: Exemple de la position si le pion prend en passant

Le roque

Il existe un problème similaire pour le coup du roque, car la validation des mouvements ne considère que le déplacement d'une seule pièce, d'une case à une autre. Cependant, lors du roque, le roi doit également vérifier s'il se déplace à travers une case attaquée et s'il est en échec sur sa case de destination.

Prenons l'exemple suivant : dans cette position, le roi peut effectuer le petit roque, mais le fou bloque le chemin en attaquant les cases entre les deux pièces. Ainsi, dans cette configuration, le roi ne devrait pas pouvoir roquer.

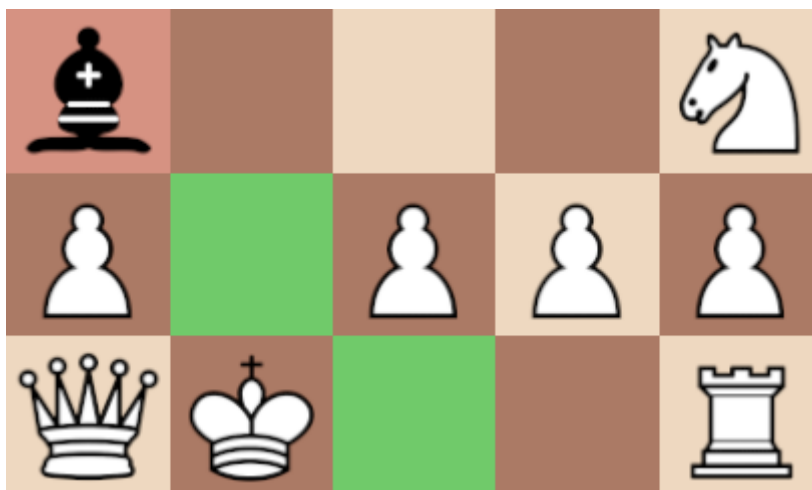


Figure 32: Exemple d'une position ou le fou adverse bloque le roi de roquer

Cependant, on peut voir que la case est toujours disponible comme mouvement lorsque l'on sélectionne le roi.

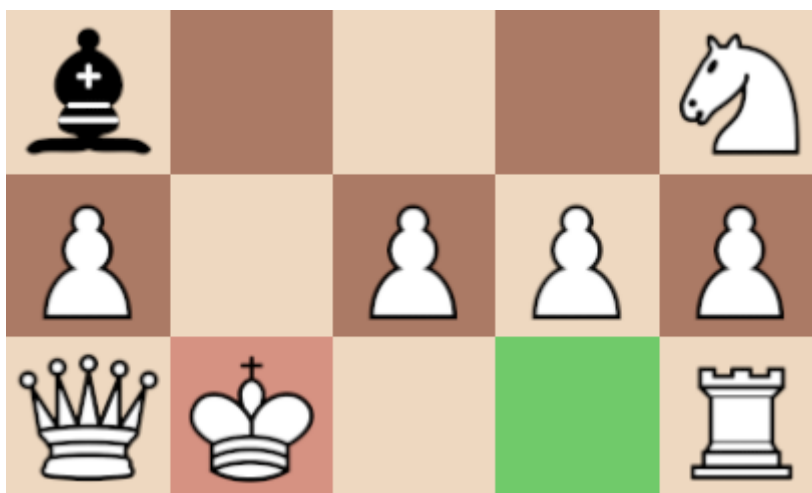


Figure 33: Exemple ou la case mis en évidence montre que le roque est possible

Dans cette situation, on peut roquer comme ci-dessous :

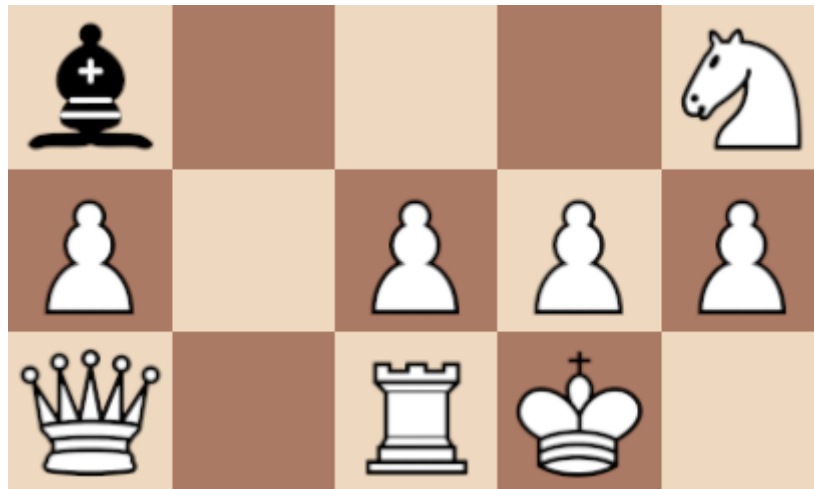


Figure 34: Exemple de la position si l'on décide de roquer

Liste des documents fournis

Les documents à fournir pour un client sont les suivants :

- Le manuel d'installation
- Le manuel d'utilisation

Ces documents peuvent être retrouvé dans le répertoire GitHub du projet.

Conclusions

Objectifs non atteints

Les objectifs non atteints du projet sont les suivants :

La promotion

Les promotions sont fonctionnelles mais ne laissent pas la possibilité aux joueurs de choisir la pièce à remplacer par le pion. Lors d'une promotion, le pion est strictement remplacé par une reine.

Pour implémenter cette fonctionnalité, des composants différents devraient interagir entre eux.

Pour l'affichage des pièces de promotion, ce rendu graphique sera géré par le « MenuManager ».

Durant la promotion, les pièces ne devraient pas être sélectionnables. Pour ce faire, l'implémentation d'un 3^e état de saisie de « InputManager » devrait être défini, ce qui rendrait la sélection des pièces sur l'échiquier impossible mais laisserait la sélection de la pièce de promotion.

Enfin, la modification de la méthode « Promote » du pion devrait être modifiée pour accepter un paramètre définissant la pièce à laquelle le pion doit être promu.

Le chronomètre

Pour implémenter les chronomètres, j'envisage de créer une nouvelle classe nommée « TimerManager » qui s'occupera de toute la logique des chronomètres.

Une mise en place graphique dans la scène de la partie devrait être rajoutée ainsi qu'une implémentation logique de la mise à jour du chronomètre dans la méthode MonoBehavior « Update », qui est appelé à chaque frame du jeu, avec l'affichage de cette méthode.

Les conditions de fin de partie

Pour les conditions de fin de partie, la classe « GameManager » s'occupe de tous les aspects généraux d'une partie comme : le nombre de tours, à quelle équipe c'est le tour de jouer et si la partie est terminée.

Pour implémenter les conditions de fin de partie, différentes conditions doivent être vérifiées à chaque mouvement effectué.

Pour le pat, il suffit de générer tous les mouvements possibles du joueur. Si le joueur n'a aucun coup possible, il y a match nul et le jeu est arrêté.

Pour l'échec et mat, comme pour le pat, tous les mouvements possibles sont générés. Mais, en plus de cela, il faut générer les mouvements adverses pour vérifier si le roi est attaqué par une pièce adverse. Si c'est le cas, il y a échec et mat et le jeu s'arrête.

Le fichier de récapitulation

Pour implémenter le fichier de récapitulation, la classe « SummaryManager » devrait être créée pour garder en mémoire les coups joués.

Une méthode pour traduire le mouvement d'une pièce et des cases devrait être implémentée. Une logique supplémentaire pour les coups spéciaux devrait également être envisagée (comme le roque, qui est noté comme « O-O » ou « O-O-O » selon la direction).

Points positifs

Ce TPI m'a permis d'identifier mes lacunes en informatique, ce qui me donne une idée claire des domaines dans lesquels je dois m'améliorer en tant qu'informaticien.

Ce projet m'a également permis d'accroître mes compétences en Unity et en résolution de problèmes, car j'ai dû faire face à des types d'erreurs que je n'avais jamais rencontrées jusqu'à présent.

Points négatifs

Dans la phase de pré-TPI, je me suis basé sur le jeu des dames or, le projet final porte sur le jeu d'échecs qui est plus complexe. Mon analyse de départ était donc un peu trop simple et a négligé les règles plus élaborées du jeu d'échecs.

J'ai commis l'erreur de démarrer mon analyse avec l'élément le plus complexe, à savoir le diagramme de classes. Or, en traitant d'abord les diagrammes de flux, les maquettes et même les diagrammes de séquence, j'aurai eu une meilleure idée de ce qui était nécessaire d'implémenter dans mes classes.

Difficultés rencontrées

Baisse de régime

Durant le 6 et le 7 mai, pour des raisons personnelles, j'ai subi ce que j'appellerai un « coup de mou » où je n'ai malheureusement pas été suffisamment productif. Ceci a donc engendré un retard important dans les tâches à réaliser.

Pour m'y remettre, j'ai décidé de réorganiser mes priorités et de planifier mentalement les tâches à réaliser.

Certaines tâches moins importantes, comme l'implémentation des tests unitaires, ont été repoussées à la fin de l'implémentation complète si le temps le permettait.

Cela m'a permis de me sentir moins submergé par le travail à accomplir et de retrouver des idées claires.

J'ai ensuite éliminé les distractions de mon environnement de travail, comme mon téléphone, que je laissais dans mon sac.

Enfin, lorsque je me sentais déconcentré ou moins performant, je prenais une brève pause. Cela m'a aidé mentalement à clarifier mes idées et à me sentir plus positif pour terminer.

Dès lors, j'ai su garder un bon rythme de travail et ma productivité s'est considérablement améliorée.

Validation des mouvements

La validation des mouvements a entraîné des erreurs complexes à déboguer, notamment dues à la difficulté d'afficher les valeurs des variables.

La première erreur à résoudre était une erreur « stack overflow » dû à un appel récursif d'une méthode.

À la fin de la génération des mouvements, la méthode de validation était appelée. Celle-ci invoquait ensuite la méthode de vérification de la position, qui à son tour déclenchait à nouveau la génération des mouvements, engendrant ainsi une boucle infinie.

Pour remédier à cette erreur, le paramètre « validateMoves » à la méthode de la génération des coups a été rajouté ce qui définit si les coups générés doivent être validés ou non.

La deuxième erreur rendait la génération des mouvements possibles partiellement fonctionnelle. Car pour valider les mouvements d'une pièce, la méthode devait simuler la position des pièces après le mouvement.

Cependant, cette simulation n'était pas réinitialisée. Cela signifiait que si le premier coup d'une pièce était légal, tous les autres l'étaient également.

Les coups spéciaux

En raison du manque de temps, la génération des mouvements possibles d'une pièce était implémentée de manière non évolutive. Ceci a rendu l'implémentation des différents coups spéciaux bien plus complexe que nécessaire.

L'amélioration possible de la génération des mouvements a déjà été abordée dans le chapitre de la réalisation.

Suites possibles

En partant du principe que les erreurs restantes ont été résolues et que les objectifs non atteints ont été réalisés, voici quelques suites possibles de ce projet :

Améliorer l'interface utilisateur

Certaines améliorations au niveau graphique seront bien appréciées :

- Pour l'échiquier, il n'y a pas de notation de la lettre du rang ou du numéro de la file.
- Pour la prévisualisation des cases, des différences de couleurs des cases seraient les bienvenues.

Voici un exemple de la prévisualisation des cases.

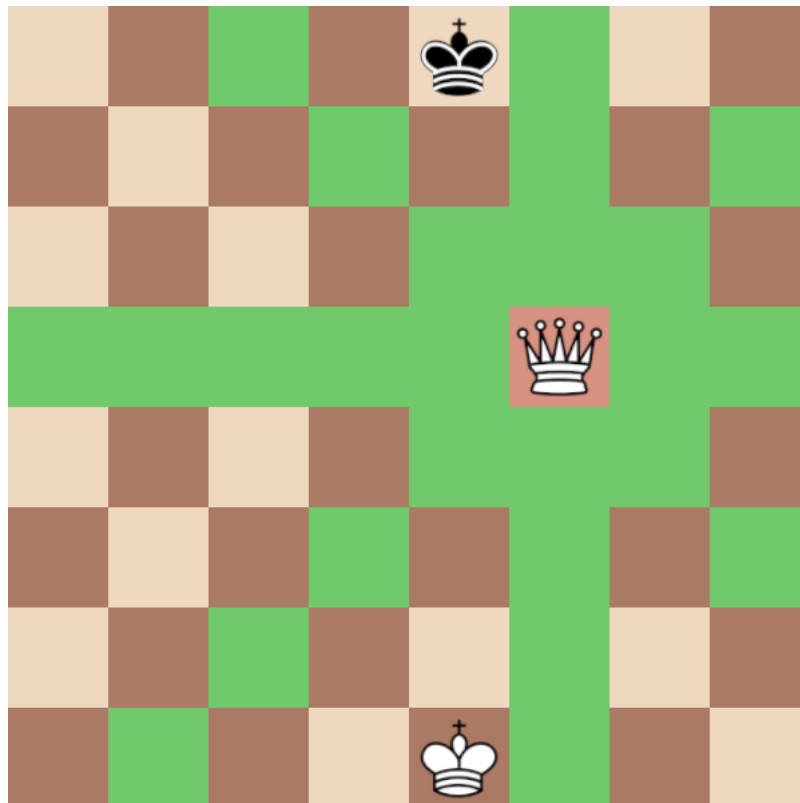


Figure 35: Exemple de la mise en évidence des coups possibles d'une pièce

Comme on peut le voir sur cette image, le fait que la prévisualisation des cases soit d'une couleur unie, qui ne varie pas en fonction de la couleur de la case, peut prêter à confusion concernant la délimitation des carrés.

La solution la plus évidente serait de changer la prévisualisation des coups légaux en points.

Une autre solution serait de conserver la couleur des cases, mais de modifier leur luminosité en fonction de la couleur de la case.

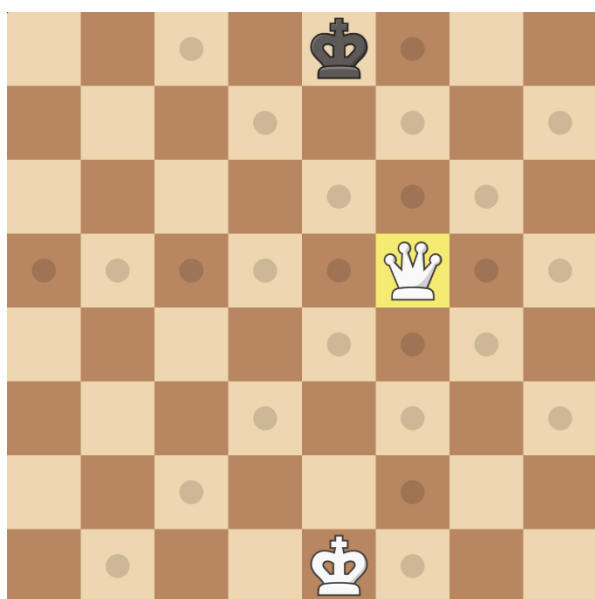


Figure 36: Exemple d'une solution de la mise en évidence des coups possible d'une pièce de chess.com

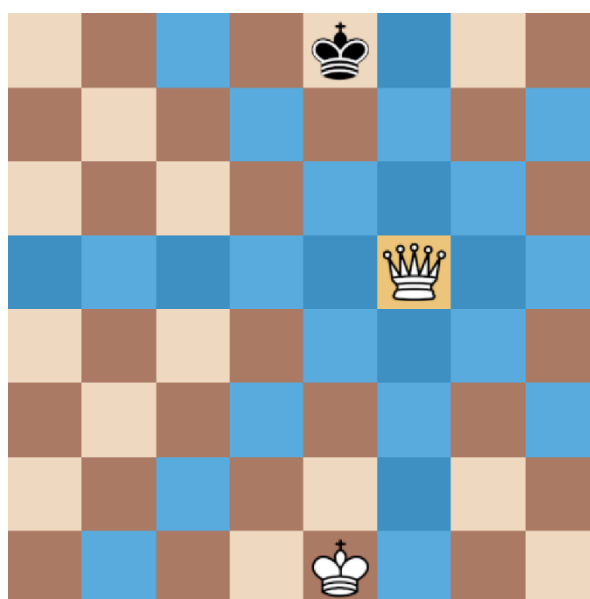


Figure 37: Exemple d'une solution de la mise en évidence des coups possibles d'une pièce

Ajouter un moteur d'échecs

Une idée courante pour les programmes d'échecs est d'implémenter une intelligence artificielle, autrement dit un moteur d'échecs.

On peut soit intégrer un moteur d'échecs déjà existant, comme « Stockfish », ou en implémenter un par soi-même.

Pour intégrer un moteur d'échecs dans le programme, des modifications doivent être faites pour accepter des protocoles dépendants du moteur, comme le protocole UCI ou Winboard.

Implémenter son propre moteur d'échecs peut être une tâche assez complexe qui nécessitera une forte modification de l'état actuel du projet, d'une part, pour le fonctionnement du moteur et, d'autre part, pour la performance.

Pour plus de renseignements sur le sujet, il existe le CPW (Chess Programming WIKI » qui explique avec précision les différentes techniques d'implémentation d'un moteur d'échecs.

Ajouter un mode en réseau

Une autre possibilité est de laisser la possibilité aux joueurs de jouer contre un adversaire en réseau. Cela nécessitera soit la création d'un serveur pour gérer la communication entre les joueurs ou une connexion « peer-to-peer » qui est moins sécurisée, mais plus simple à implémenter.

Ceci pourrait être implémenté avec les frameworks offerts par Unity, Mirror et Photon.

Ajouter un mode de jeu puzzle

Dans les programmes d'échecs, on trouve souvent une fonctionnalité de « puzzle », où l'on donne au joueur une position et il doit trouver le ou les coups gagnants.

On peut envisager de mettre en place ce mode de jeu en utilisant une liste de positions initiales, associée à chaque position initiale une liste de coups pour résoudre le problème.

Annexes

Résumé du rapport du TPI

Ce travail de projet individuel, réalisé dans le cadre du CPNV avec 90 heures allouées, consiste à développer un jeu d'échecs en 2D en utilisant Unity comme moteur de jeu. Les règles officielles des échecs seront implémentées pour permettre à deux joueurs de s'affronter.

La réalisation du projet a commencé plus tard que prévu en raison d'un coup de mou durant la phase d'analyse. Cependant, les fonctionnalités clés du programme ont pu être implémentées. La génération des pièces et de l'échiquier au début de l'implémentation n'a pas posé de problème. Pour les cases de l'échiquier, un préfab a été créé pour faciliter la génération, et les pièces sont définies par le composant ajouté au « GameObject » enfant de la case. Les pièces ont été implémentées avec le polymorphisme en tête, ce qui permet une implémentation évolutive et plus simple. La génération des mouvements des pièces n'a pas posé de problème significatif, mais la vérification des mouvements générés et les coups spéciaux comme la prise en passant et le roque ont été plus complexes à implémenter. Cela est dû à la manière dont les mouvements sont représentés dans le programme, limitant ainsi la flexibilité. Finalement, des fonctionnalités comme la fin de partie et le fichier récapitulatif n'ont pas pu être implémentées par manque de temps, principalement en raison des retards au démarrage du projet et de la mauvaise implémentation de la représentation des coups dans le programme.

Dans l'ensemble, bien qu'il y ait eu quelques soucis, ils n'ont pas été un obstacle insurmontable pour le projet et ont pu être surmontés. Dans le programme final, quelques bugs sont toujours présents et des améliorations sont possibles, mais la manière de résoudre les problèmes rencontrés est plus importante que le résultat final.

Planification finale

Pour la planification finale, les tâches en vert représentent la réalisation fait durant le projet, et les sections bleu est la planification détaillée du projet.

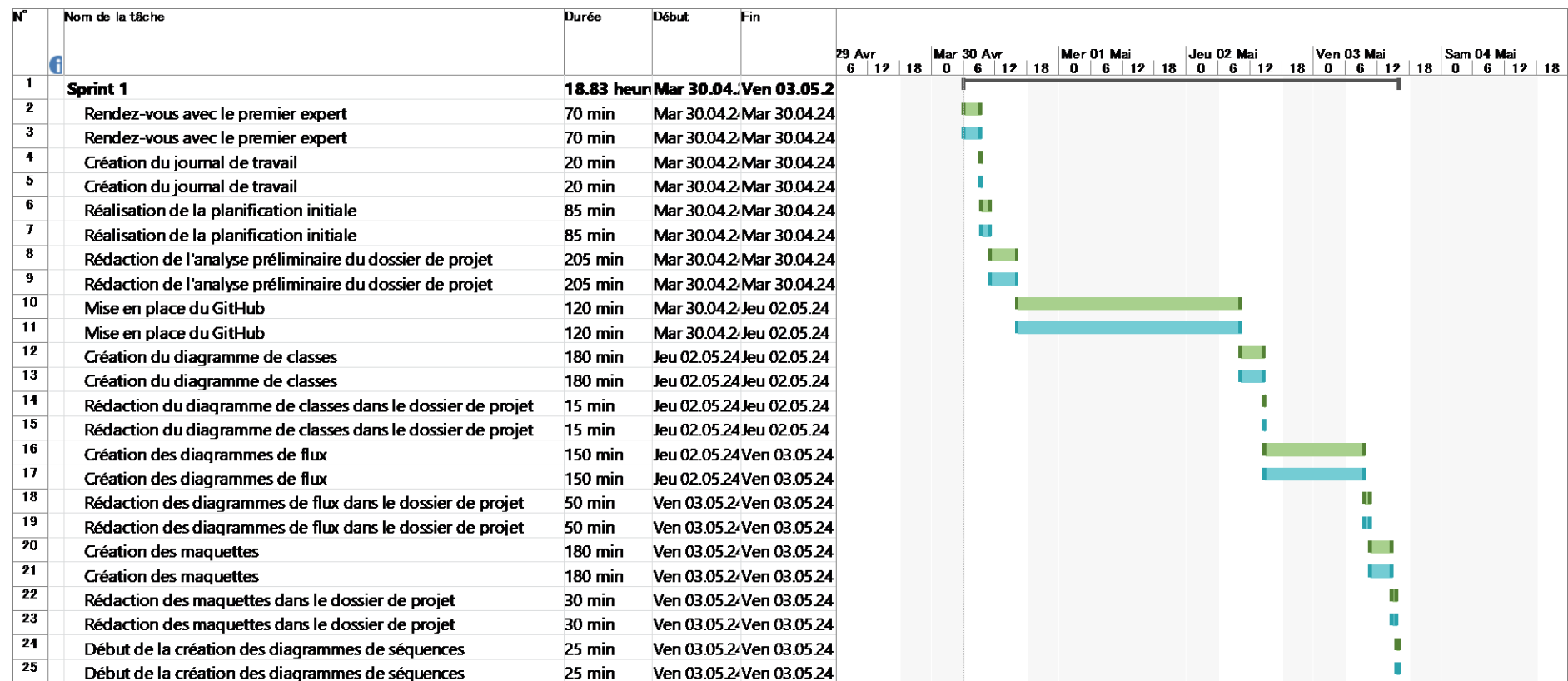


Figure 38: Sprint 1 de la planification finale

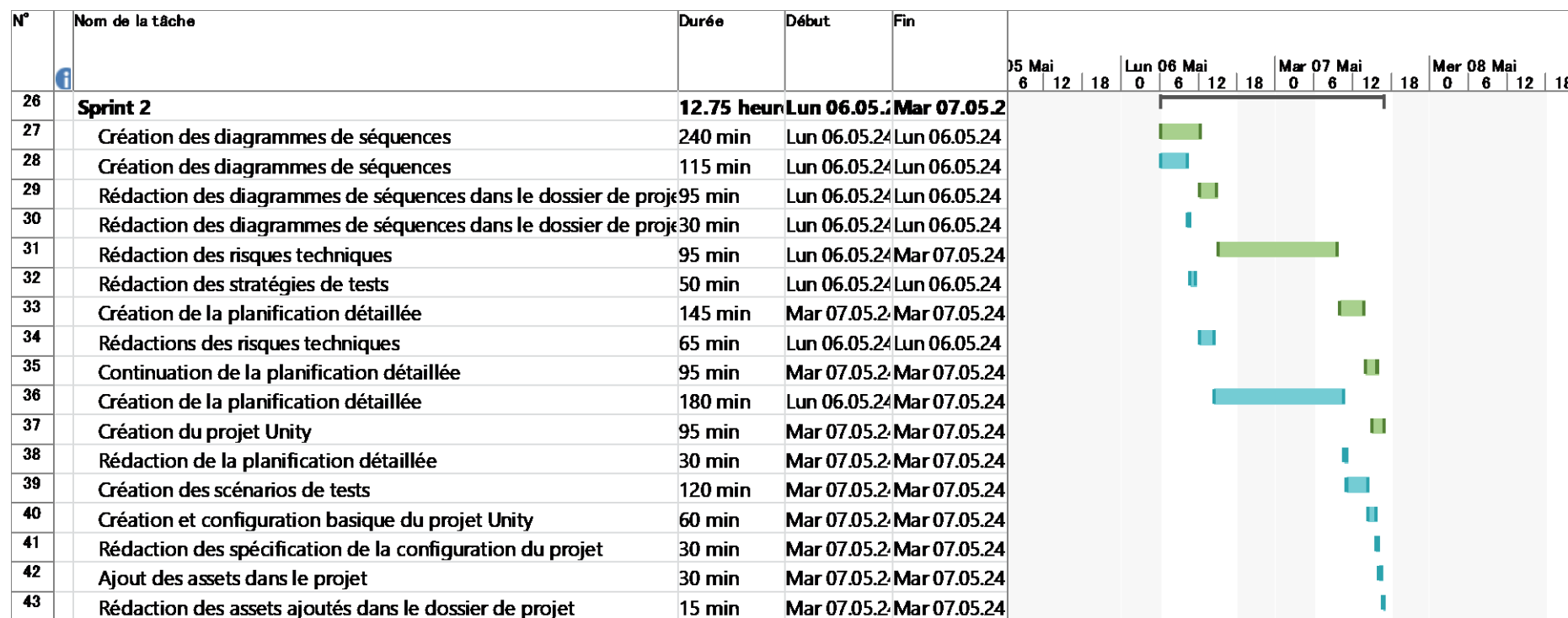


Figure 39: Sprint 2 de la planification finale

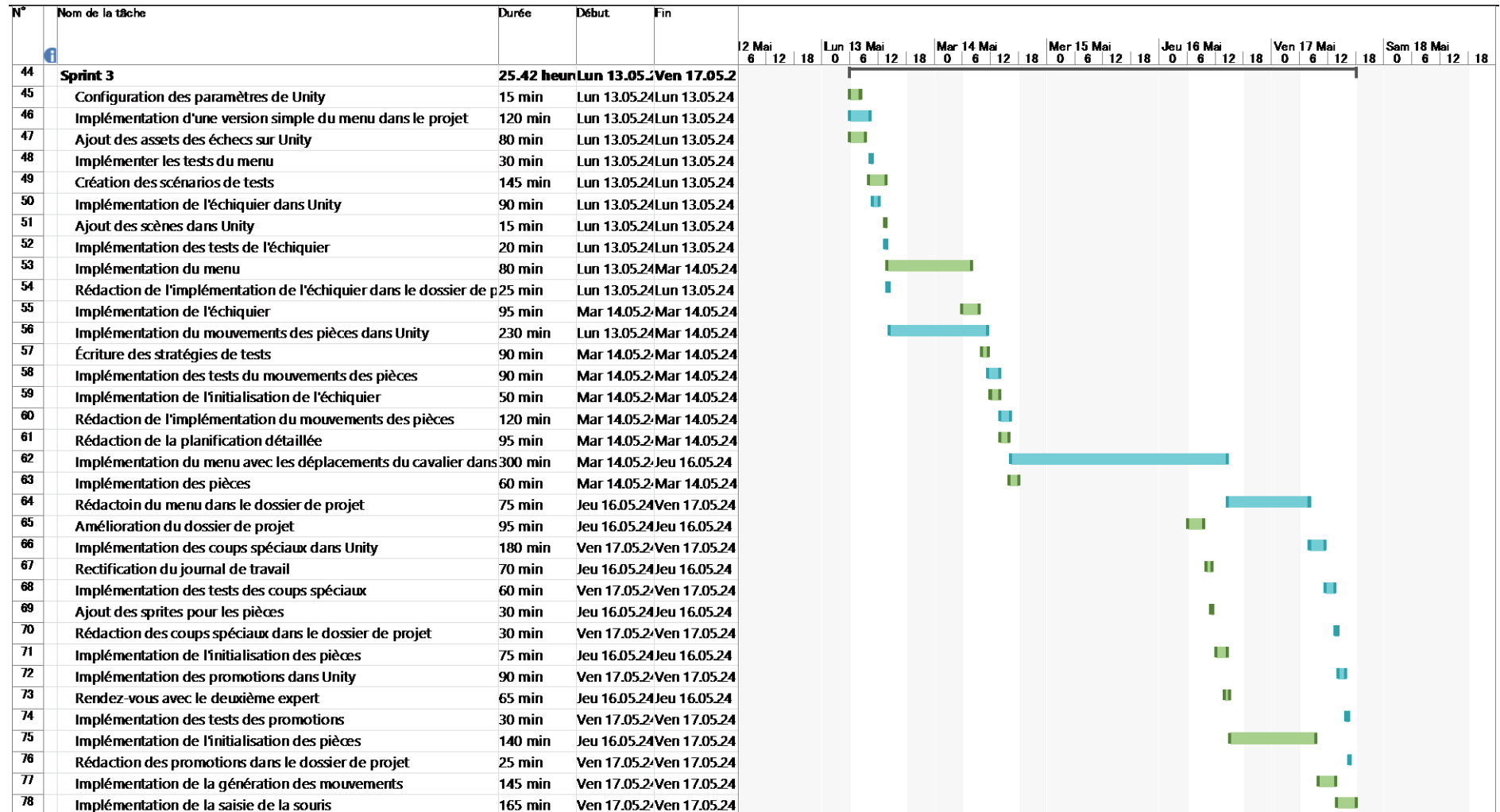


Figure 40: Sprint 3 de la planification finale

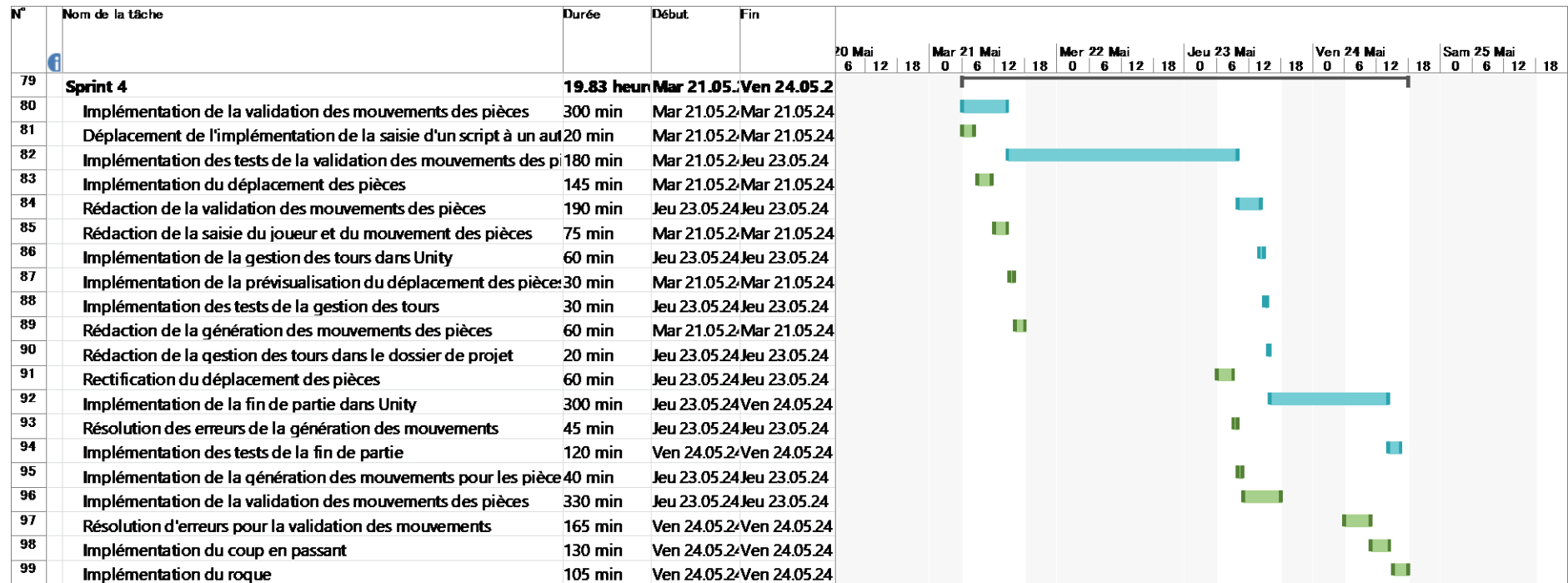


Figure 41: Sprint 4 de la planification finale

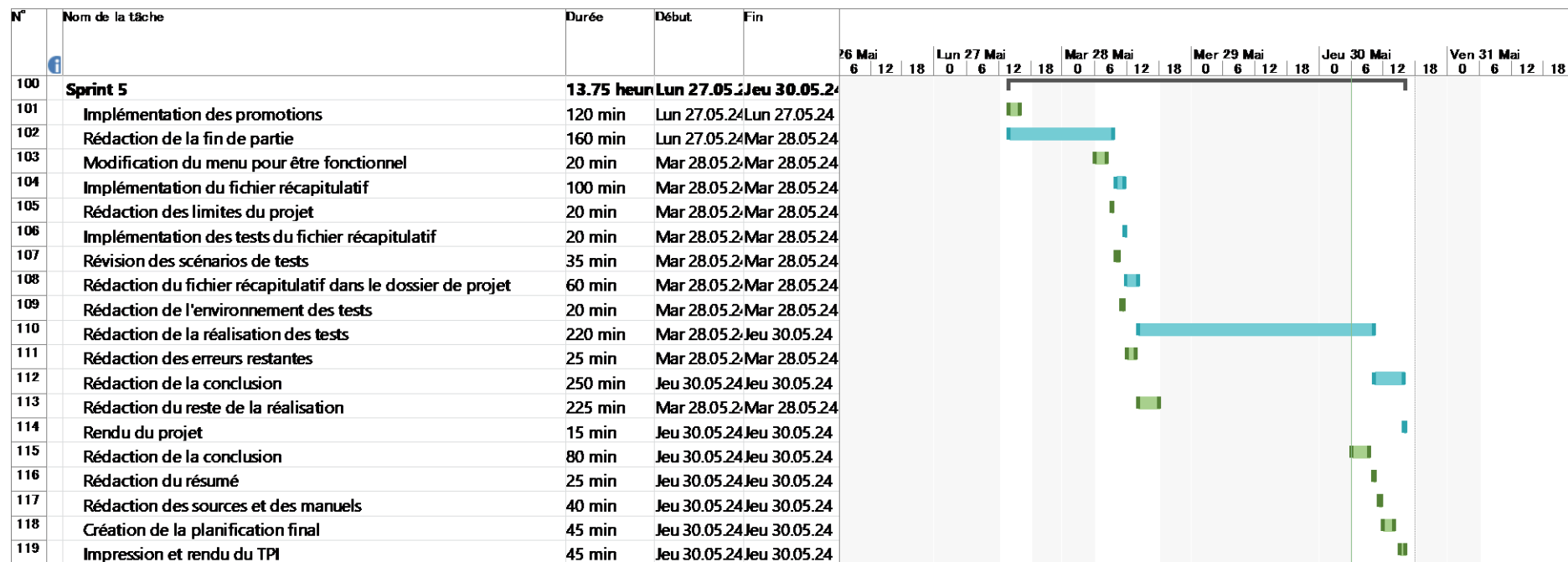


Figure 42: Sprint 5 de la planification finale

Journal de travail

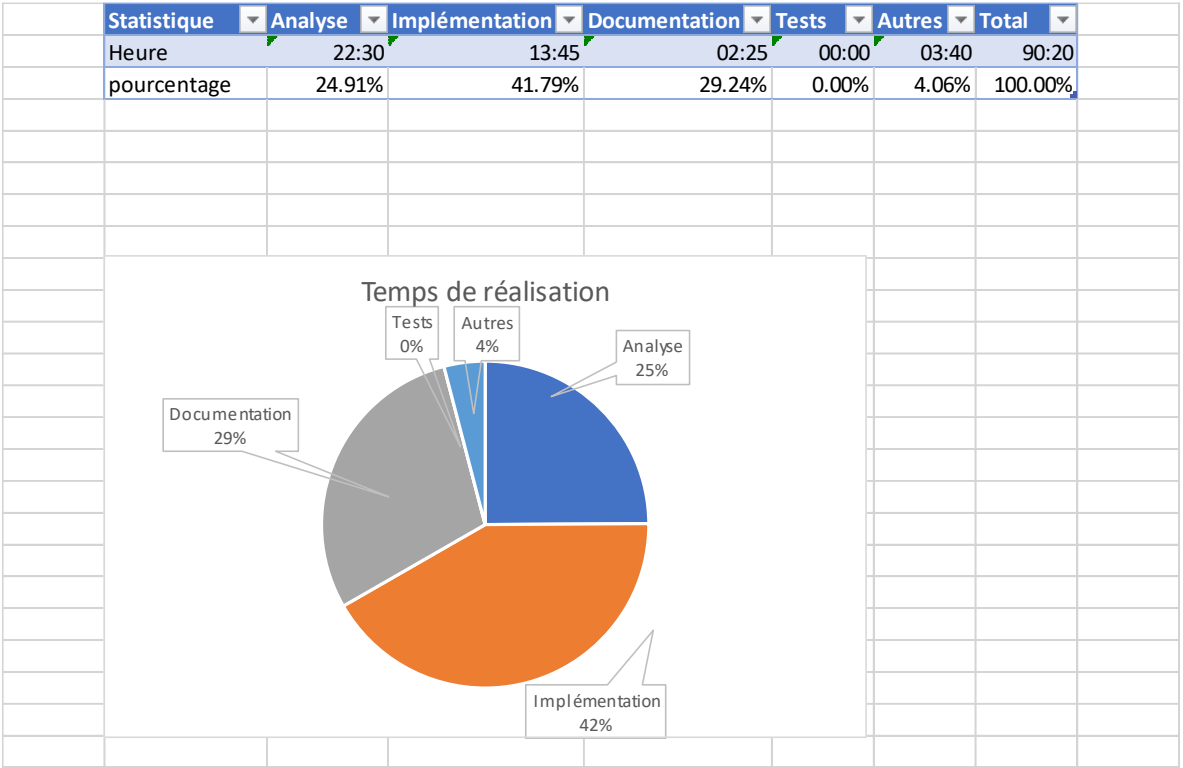
Journal de travail - TPI Jeu d'échec sur Unity							
Date	Début	Fin	Durée	Sujet	Description	Remarque	Source
30.04.2024	08:05	09:15	01:10	Autres	Meeting avec le premier expert, reçu et signature du cahier des charges, explication du déroulement du TPI		
30.04.2024	09:15	09:35	00:20	Documentation	Création du journal de travail		
30.04.2024	09:50	10:05	00:15	Autres	Communication avec le deuxième expert pour valider la méthode de communication		
30.04.2024	10:05	11:30	01:25	Analyse	Mise en place de la planification initiale.	à vérifier avec mon chef de projet	
30.04.2024	11:30	12:15	00:45	Documentation	Commencement de l'écriture de l'analyse préliminaire du dossier de projet	Manque de compléter les chapitres objectifs et planification initiale	Mahé Lavaud pour des conseils de style sur la page de garde
30.04.2024	13:30	14:30	01:00	Documentation	Écriture de l'analyse préliminaire du dossier de projet	Style à jour, première version de la page de garde faite et analyse préliminaire terminé	
30.04.2024	14:30	15:05	00:35	Documentation	Amélioration du style pour le dossier de projet et le journal de travail		
30.04.2024	15:20	15:45	00:25	Documentation	Ajout d'un graphique dans le journal de travail pour mieux représenter la répartition des tâches		
30.04.2024	15:45	16:00	00:15	Autres	Création du répertoire GitHub et premier commit des fichiers de documentation		
30.04.2024	16:00	16:40	00:40	Documentation	Écriture de la planification initiale dans le dossier de projet		
30.04.2024	16:40	16:55	00:15	Autres	Écriture de l'email pour l'envoi de la planification initiale	Je me sens confiant pour la suite du projet	
02.05.2024	08:50	09:35	00:45	Analyse	Création des milestones, des tags et du projet KanBan sur GitHub		
02.05.2024	09:50	10:50	01:00	Analyse	Création des tâches sur GitHub		
02.05.2024	10:50	12:15	01:25	Analyse	Création du diagramme de classes		Comment utiliser astah: https://astah.net/support/astah-pro/user-guide/class-diagrams/
02.05.2024	13:30	14:15	00:45	Analyse	Réalisation du diagramme de classes		Guide pour les relations: https://cpnv-es-ngv.gitbook.io/uml-backlog/class-diagram/standards/les-relations
02.05.2024	14:15	14:30	00:15	Analyse	Validation du diagramme de classes		Validation faite avec Monsieur Viret
02.05.2024	14:30	15:05	00:35	Analyse	Rectification du diagramme de classes		
02.05.2024	15:20	15:35	00:15	Documentation	Écriture de l'analyse du diagramme de classes		
02.05.2024	15:35	16:05	00:30	Analyse	Création du diagramme de flux de la gestion du mouvement des pièces		https://support.microsoft.com/fr-fr/topic/cr%C3%A9er-un-diagramme-de-flux-simple-dans-visio-e207d975-4a51-4bfa-a356-eeec314bd276

03.05.2024	08:00	08:40	00:40	Analyse	Création du diagramme de flux de la condition de fin de partie		
03.05.2024	08:40	09:05	00:25	Analyse	Création du diagramme de flux de la gestion des mouvements spéciaux		
03.05.2024	09:05	09:35	00:30	Analyse	Création du diagramme de flux pour la gestion du tour des joueurs		
03.05.2024	09:50	10:15	00:25	Analyse	Création du diagramme de flux pour la gestion du fichier récapitulatif		https://en.wikipedia.org/wiki/Algebraic_notation_(chess)
03.05.2024	10:15	10:40	00:25	Autres	Modification des projets sur GitHub	Changement de un projet pour le GitHub à un projet pour chaque sprint du GitHub	
03.05.2024	10:40	11:30	00:50	Documentation	Écriture des diagrammes de flux dans le dossier de projet		
03.05.2024	11:30	12:15	00:45	Analyse	Création de la maquette du menu priciaple		
03.05.2024	13:30	13:50	00:20	Analyse	Réalisation de la maquette du menu principale		
03.05.2024	13:50	14:20	00:30	Analyse	Création de la maquette du menu du chronomètre		
03.05.2024	14:20	15:05	00:45	Analyse	Création de la maquette d'une partie		
03.05.2024	15:20	16:00	00:40	Analyse	Création de la maquette pour la fin d'une partie		
03.05.2024	16:00	16:30	00:30	Documentation	Écriture des maquette dans le dossier de projet		
03.05.2024	16:30	16:55	00:25	Analyse	Création du diagramme de séquence du début de la partie		https://astah.net/support/astah-pro/user-guide/sequence-diagram/
06.05.2024	08:00	09:35	01:35	Analyse	Création du diagramme de séquence du déplacement d'une pièce	Coup de mou pour le 06 et le 07	
06.05.2024	09:50	11:05	01:15	Analyse	Création du diagramme de séquence du fin de jeu	Coup de mou pour le 06 et le 07	
06.05.2024	11:05	12:15	01:10	Analyse	Réalisation du diagramme de séquence du fin de jeu	Coup de mou pour le 06 et le 07	
06.05.2024	13:30	15:05	01:35	Documentation	Écriture des diagrammes de séquences dans le dossier de projet	Coup de mou pour le 06 et le 07	
07.05.2024	08:00	09:35	01:35	Documentation	Écriture des risques techniques	Coup de mou pour le 06 et le 07	
07.05.2024	09:50	12:15	02:25	Analyse	Création de la planification détaillée	Coup de mou pour le 06 et le 07	
07.05.2024	13:30	15:05	01:35	Analyse	Réalisation de la planification détaillée	Coup de mou pour le 06 et le 07	
07.05.2024	15:20	16:55	01:35	Implémentation	Création du projet Unity	Coup de mou pour le 06 et le 07	

13.05.2024	08:00	08:15	00:15	Implémentation	Configuration des paramètres de Unity		
13.05.2024	08:15	09:35	01:20	Implémentation	Ajout des assets des échecs sur Unity		
13.05.2024	09:50	10:40	00:50	Analyse	Création des scénarios de tests		
13.05.2024	10:40	12:15	01:35	Analyse	Réalisation des scénarios de tests		
13.05.2024	13:30	13:45	00:15	Implémentation	Ajout des scènes dans Unity		
13.05.2024	13:45	15:05	01:20	Implémentation	Ajout d'un menu temporaire du jeu		
14.05.2024	08:00	09:35	01:35	Implémentation	Création de la classe de l'échiquier et mise en place de la scène		
14.05.2024	09:50	11:25	01:35	Documentation	Écriture des stratégies de tests		https://docs.unity3d.com/Manual/InstantiatingPrefabs.html
14.05.2024	11:25	12:15	00:50	Implémentation	Implémentation de l'initialisation de l'échiquier		
14.05.2024	13:30	15:05	01:35	Documentation	Rédaction de la planification		
14.05.2024	15:20	16:55	01:35	Implémentation	Création des scripts pour les pièces		
16.05.2024	08:00	09:35	01:35	Documentation	Amélioratoin du dossier de projet		
16.05.2024	09:50	11:00	01:10	Documentation	Rectification du journal de travail pour le 06, 07 et le 13 mai		
16.05.2024	11:00	11:30	00:30	Implémentation	Ajout des sprites pour les pièces		
16.05.2024	11:30	12:15	00:45	Implémentation	Implémentation de l'initialisation des pièces		
16.05.2024	13:30	14:00	00:30	Implémentation	Continuation de l'implémentation de l'initialisation des pièces		
16.05.2024	14:00	15:05	01:05	Autres	Rendez-vous avec le deuxième expert		
16.05.2024	15:20	16:05	00:45	Implémentation	Continuation de l'implémentation de l'initialisation des pièces		https://docs.unity3d.com/Manual/sprite-automatic-slicing.html
17.05.2024	08:00	09:35	01:35	Implémentation	Finalisation et refactor de l'implémentation de l'initialisation des pièces		https://learn.microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2?view=net-8.0
17.05.2024	09:50	12:15	02:25	Implémentation	Implémentation de la génération du mouvement des pièces		
17.05.2024	13:30	15:05	01:35	Implémentation	Implémentation de la saisie de la souris		https://docs.unity3d.com/Packages/com.unity.inputsystem@1.8/manual/Installation.html

17.05.2024	15:20	16:55	01:35	Implémentation	Résolution de problème avec la saisie de la souris		
21.05.2024	08:00	08:20	00:20	Implémentation	Déplacement de la fonctionnalité de la saisie dans un script apart		
21.05.2024	08:20	09:35	01:15	Implémentation	Implémentation du déplacement des pièces		
21.05.2024	09:50	11:00	01:10	Implémentation	Implémentation du déplacement des pièces		https://discussions.unity.com/t/how-delete-or-remove-a-component-of-an-gameobject/60182
21.05.2024	11:00	11:30	00:30	Implémentation	Refactorisation du code		
21.05.2024	11:30	12:15	00:45	Documentation	Rédaction de la saisie du joueur		
21.05.2024	13:30	14:00	00:30	Documentation	Rédaction du mouvement des pièces		
21.05.2024	14:00	14:30	00:30	Implémentation	Implémentation de la prévision du déplacement des pièces		
21.05.2024	14:30	15:05	00:35	Documentation	Rédaction de la génération des mouvements des pièces		
21.05.2024	15:20	15:55	00:35	Documentation	Rédaction de la génération des mouvements des pièces		
21.05.2024	15:55	16:55	01:00	Implémentation	Rectification du déplacement des pièces	Problème du script qui ne garde pas en mémoire les variable de la pièce, qui cause problème dans certain cas	
23.05.2024	08:50	09:35	00:45	Implémentation	Résolution d'erreur de la génération des mouvements des pièces		
23.05.2024	09:50	10:30	00:40	Implémentation	Implémentation de la génération du mouvement des pièces restante		
23.05.2024	10:30	11:00	00:30	Implémentation	Implémentation de la gestion de tour		
23.05.2024	11:00	12:15	01:15	Implémentation	Implémentation de la simulation d'une position pour faire la validation des mouvements d'une pièce		
23.05.2024	13:30	15:05	01:35	Implémentation	Implémentation de la validation d'une position		
23.05.2024	15:20	16:05	00:45	Implémentation	Résolution de problème de la génération des coups		
24.05.2024	08:00	09:35	01:35	Implémentation	Résolution de la validation des coups		
24.05.2024	09:50	11:00	01:10	Implémentation	Résolution de la validation des coups		
24.05.2024	11:00	12:15	01:15	Implémentation	Implémentation du coup en passant		
24.05.2024	13:30	14:25	00:55	Implémentation	Résolution d'erreur pour le coup en passant		
24.05.2024	14:25	15:05	00:40	Implémentation	Implémentation du roque		
24.05.2024	15:20	15:30	00:10	Implémentation	Implémentation du roque		
24.05.2024	15:30	16:55	01:25	Implémentation	Résolution d'erreur du roque		
27.05.2024	13:30	15:05	01:35	Implémentation	Implémentation de la promotion		https://stackoverflow.com/questions/64810646/delete-gameobject-before-the-end-of-the-frame
28.05.2024	08:00	08:20	00:20	Implémentation	Modification du menu pour le jeu et réalisation du build du jeu		https://docs.unity3d.com/ScriptReference/SceneManager.SceneManager.LoadScene.html
28.05.2024	08:20	08:40	00:20	Documentation	Rédaction des limites de l'environnement		
28.05.2024	08:40	09:15	00:35	Documentation	Révision des scénarios de tests	Certains cas de tests étaient écrits de manière incorrecte, et certains cas de tests était manquant	
28.05.2024	09:15	09:35	00:20	Documentation	Rédaction de l'environnement des tests dans la documentation		
28.05.2024	09:50	10:15	00:25	Documentation	Rédaction des erreurs restantes		
28.05.2024	10:15	10:45	00:30	Documentation	Rédaction de la mise en place		
28.05.2024	10:45	11:05	00:20	Documentation	Amélioration des images		
28.05.2024	11:05	11:40	00:35	Documentation	Rédaction de la génération des mouvements		
28.05.2024	11:40	12:15	00:35	Documentation	Rédaction de la validation des mouvements		
28.05.2024	13:30	13:50	00:20	Documentation	Correction du mouvement des pièces		
28.05.2024	13:50	14:30	00:40	Documentation	Rédaction du roque		

28.05.2024	14:30	15:05	00:35	Documentation	Rédaction de la prise en passant		
28.05.2024	15:20	15:40	00:20	Documentation	Rédaction des promotions		
28.05.2024	15:40	16:15	00:35	Documentation	Rédaction des objectifs non atteints		
28.05.2024	16:15	16:30	00:15	Documentation	Rédaction des conditions de fin de partie		
30.05.2024	08:50	09:35	00:45	Documentation	Rédaction de la conclusion		
30.05.2024	09:50	10:25	00:35	Documentation	Rédaction de la conclusion		
30.05.2024	10:25	10:50	00:25	Documentation	Rédaction du résumé		
30.05.2024	10:50	11:30	00:40	Documentation	Rédaction des sources, le manuel d'installation et d'utilisation		
30.05.2024	11:30	12:15	00:45	Documentation	Création et rédaction de la planification final		
30.05.2024	13:30	13:45	00:15	Autres	Impression et envoi du TPI		



Manuel d'Installation

Le répertoire GitHub se trouve sur le lien suivant :

https://github.com/ArthurCPNV/Jeu_dechec_sur_Unity

1. cloner le répertoire GitHub dans le dossier désiré avec la commande suivante :

```
C:\>git clone https://github.com/ArthurCPNV/Jeu_dechec_sur_Unity.git
```

Figure 43: Commande Git pour cloner le répertoire

2. Lancer le projet à l'aide de Unity version 2022.3.19f1.

3. Dans « File », ouvrez les options de compilation :

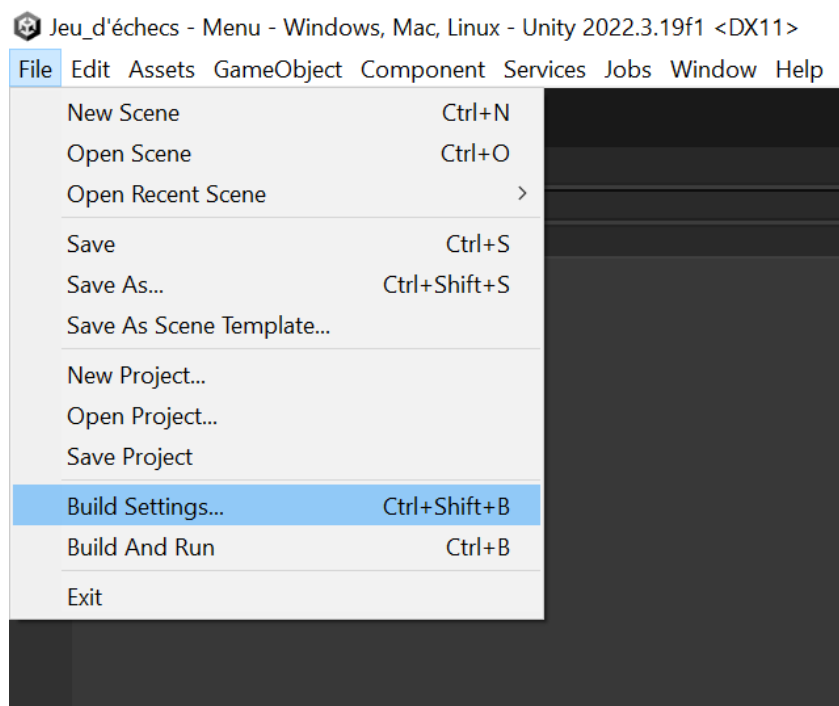


Figure 44: Le menu pour ouvrir les options de compilation

4. Laisser la configuration par défaut, et cliquer sur « Build » pour compiler le projet.

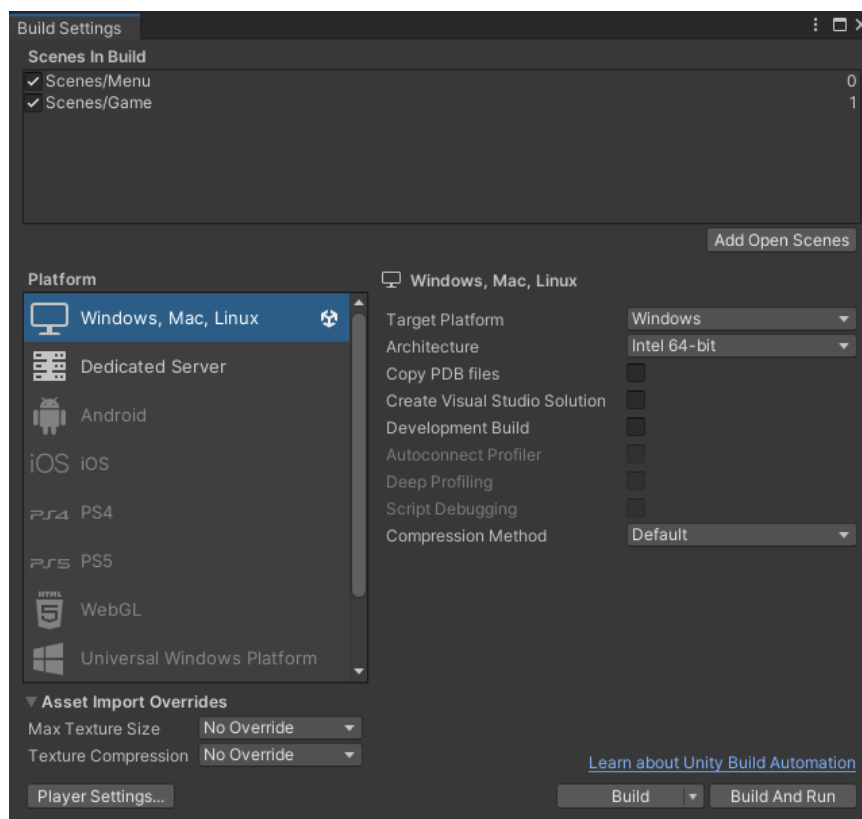


Figure 45: La fenêtre des options de compilation

Manuel d'Utilisation

Le programme suit les règles officielles des échecs, toutes les saisies des actions de l'utilisateur ce fait par la souris.

Dans le menu, le bouton cliquer commence une partie d'échecs, la partie quitter ferme le programme.



Figure 46: Le menu du programme

Lorsqu'une partie est lancée, le joueur doit cliquer sur une pièce pour la sélectionner.

Lorsqu'une pièce est sélectionnée, le joueur a plusieurs options :

- Faire un clic gauche pour désélectionner la pièce.
- Cliquer sur une case non valide pour désélectionner la pièce.
- Cliquer sur une autre pièce pour en sélectionner une nouvelle.
- Déplacer la pièce en cliquant sur une case mise en évidence.



Figure 47: L'affichage de la sélection d'un pion

Lorsque la pièce est sélectionnée, la case de la pièce est mise en évidence et l'on peut voir les coups possibles du pion par les cases vertes.

Sources

Sites internet

Diagrammes

<https://www.ibm.com/docs/en/rsm/7.5.0?topic=diagrams-relationships-in-class> - 02.05.2024

<https://astah.net/support/astah-pro/user-guide/class-diagrams/> - 02.05.2024

<https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/access-modifiers> - 02.05.2024

<https://cpnv-es-ngy.gitbook.io/uml-backlog/class-diagram/standards/les-relations> - 02.05.2024

<https://stackoverflow.com/questions/17631125/trying-to-convert-diagram-of-a-chess-game-to-java-code-abstract> - 02.05.2024

<https://support.microsoft.com/fr-fr/topic/cr%C3%A9er-un-diagramme-de-flux-simple-dans-visio-e207d975-4a51-4bfa-a356-eeec314bd276> - 03.05.2024

<https://astah.net/support/astah-pro/user-guide/sequence-diagram/> - 06.05.2024

Documentation Unity

<https://stackoverflow.com/questions/53877088/unity-whats-the-difference-between-a-playmode-unitytest-and-an-editmode-unity> - 14.05.2024

<https://discussions.unity.com/t/how-to-detect-mouse-click-on-a-gameobject/59449> - 14.05.2024

<https://docs.unity3d.com/Manual/InstantiatingPrefabs.html> - 14.05.2024

<https://docs.unity3d.com/Manual/sprite-automatic-slicing.html> - 17.05.2024

<https://learn.microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2?view=net-8.0> - 17.05.2024

<https://docs.unity3d.com/Packages/com.unity.inputsystem@1.8/manual/Installation.html> - 17.05.2024

<https://discussions.unity.com/t/how-delete-or-remove-a-component-of-an-gameobject/60182> - 21.05.2024

<https://stackoverflow.com/questions/64810646/delete-gameobject-before-the-end-of-the-frame> - 27.05.2024

<https://docs.unity3d.com/ScriptReference/SceneManagement.SceneManager.LoadScene.html> - 28.05.2024

Règles des échecs

<https://fr.wikipedia.org/wiki/%C3%89chiquier> – 14.05.2024

https://fr.wikipedia.org/wiki/R%C3%A8gles_du_jeu_d%C3%A9checs – 21.05.2024

[https://fr.wikipedia.org/wiki/En_passant_\(%C3%A9checs\)](https://fr.wikipedia.org/wiki/En_passant_(%C3%A9checs)) – 24.05.2024

[https://fr.wikipedia.org/wiki/Roque_\(%C3%A9checs\)](https://fr.wikipedia.org/wiki/Roque_(%C3%A9checs)) – 24.05.2024

[https://fr.wikipedia.org/wiki/Promotion_\(%C3%A9checs\)](https://fr.wikipedia.org/wiki/Promotion_(%C3%A9checs)) – 27.05.2024

Autres

<https://docs.github.com/en/repositories/releasing-projects-on-github/managing-releases-in-a-repository> - 28.05.2024

Archives du projet

Le répertoire GitHub du projet :

https://github.com/ArthurCPNV/Jeu_dechec_sur_Unity

Le build du projet GitHub :

https://github.com/ArthurCPNV/Jeu_dechec_sur_Unity/releases/tag/v1.0