**Journal de travail TPI**

**Journal réalisé par : Sacha Leone**

Classe : 3IND-3TPMa

Commencé le : 08.04.2025

Projet : Jeu de société « Qui est-ce » digital

## Table des matières

[1. Mardi 08.04.2025 4](#_Toc198906973)

[2. Jeudi 10.04.2025 5](#_Toc198906974)

[3. Vendredi 11.04.2025 6](#_Toc198906975)

[4. Lundi 28.04.2025 8](#_Toc198906976)

[5. Mardi 29.04.2025 9](#_Toc198906977)

[6. Vendredi 02.05.2025 10](#_Toc198906978)

[7. Lundi 05.05.2025 11](#_Toc198906979)

[8. Mardi 06.05.2025 12](#_Toc198906980)

[9. Jeudi 08.05.2025 13](#_Toc198906981)

[10. Vendredi 09.05.2025 14](#_Toc198906982)

[11. Lundi 12.05.2025 15](#_Toc198906983)

[12. Mardi 13.05.2025 16](#_Toc198906984)

[13. Jeudi 15.05.2025 17](#_Toc198906985)

[14. Vendredi 16.05.2025 18](#_Toc198906986)

[15. Lundi 19.05.2025 19](#_Toc198906987)

[16. Mardi 20.05.2025 20](#_Toc198906988)

[17. Jeudi 22.05.2025 21](#_Toc198906989)

[18. Vendredi 23.05.2025 22](#_Toc198906990)

Après chaque journée de travail, la documentation comme : la planification et le journal de travail, ont été mis à jour. Je n’en ferai pas mention ultérieurement, puisque c’est une étape évidente.

# Mardi 08.04.2025

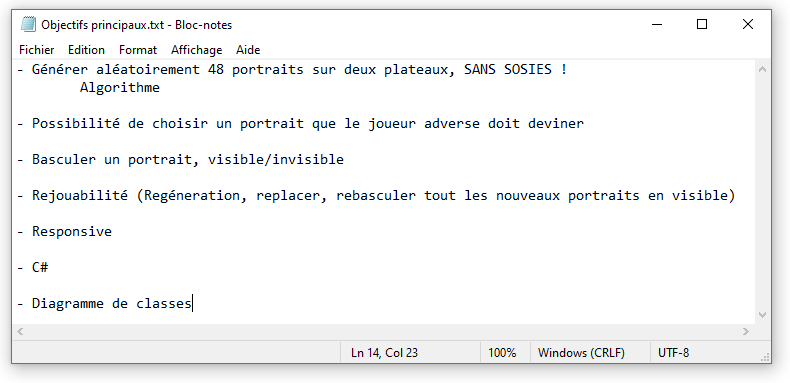
## Étapes réalisées

Après la réception et la prise de connaissance du cahier des charges, la première étape a été d’organiser mon projet comme suit :

* Préparation d’un repos GitHub pour le projet à venir.
* Préparation des différents répertoires après « git clone ».
* Préparation des différents documents (Websummary, JT, Rapport, etc…).
* Préparation de la planification.

Étant déjà en possession de canevas de documentation (Ex. : Rapport, JT, Planification) prêt à l’emploi ces étapes ont rapidement été accomplies.

Ensuite, j’ai résumé les objectifs principaux demandé, qui sont le minimum requis pour finir officiellement le produit.



Puis, j’ai listé les différents chemins que j’aurai pu emprunter pour réaliser le projet. Incluant donc les idées et leur faisabilité, les contraintes, le temps à disposition, les capacités réels.

## Problèmes rencontrés

Aucun problème rencontré. (Trop d’idées)

## Étapes suivantes

* Faire des recherches sur le sujet. (Randomisation de portrait, éviter les sosies, algorithme, etc…)

# Jeudi 10.04.2025

## Étapes réalisées

* Ayant dû réaliser ma soutenance TIP (Travail de maturité interdisciplinaire) je n’ai pas avancé sur le projet entre 8h10-9h00.
* La planification étant encore incomplète, les dernières tâches ont été rajoutées.
* En faisant des recherches plus approfondies, j’ai trouvé un projet GitHub open-source appelé « Avataaaar » qui crée des portraits aléatoirement grâce à des images svg superposées.
* Voici le site [getavataaars.com](https://getavataaars.com/) qui présente le fonctionnement.
* Ce repos GitHub serait peut-être d’aide puisqu’il contient déjà une bonne partie du fonctionnement recherché. Il faut tout de même comprendre son fonctionnement et mettre en place l’algorithme de création de portrait en C#, ce repos contient le code en TypeScript et HTML qui sont très différents du langage choisi.

## Problèmes rencontrés

Aucun problème rencontré.

## Étapes suivantes

* Réflexions sur les ajouts faisables
* Choix de l'environnement de développement C#

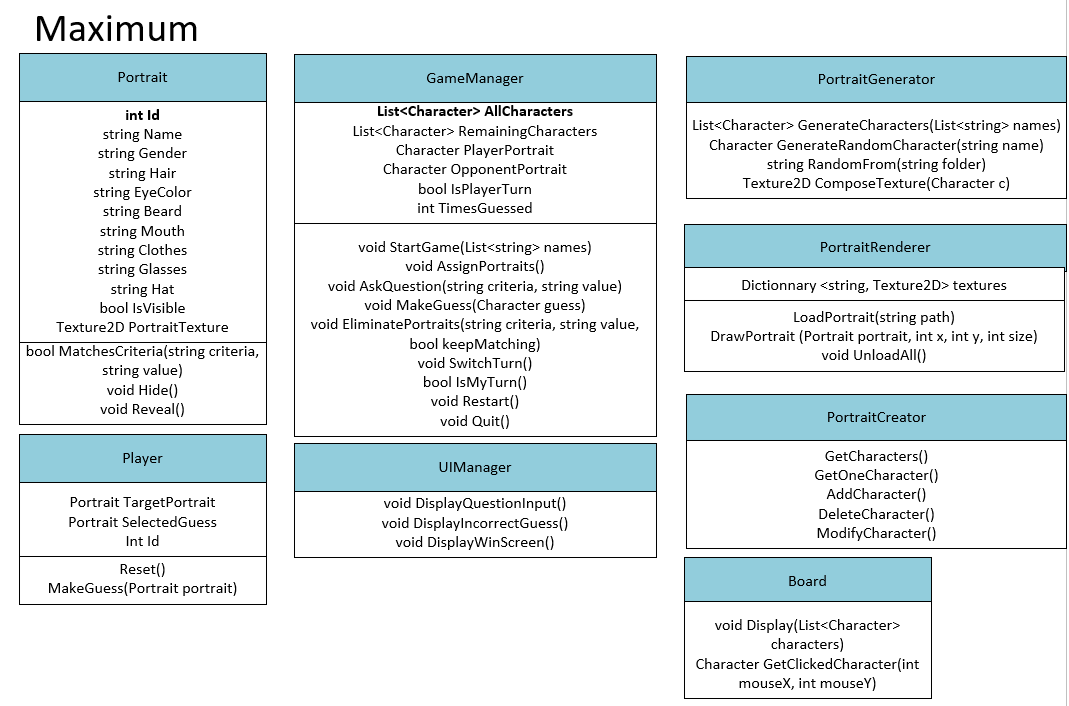
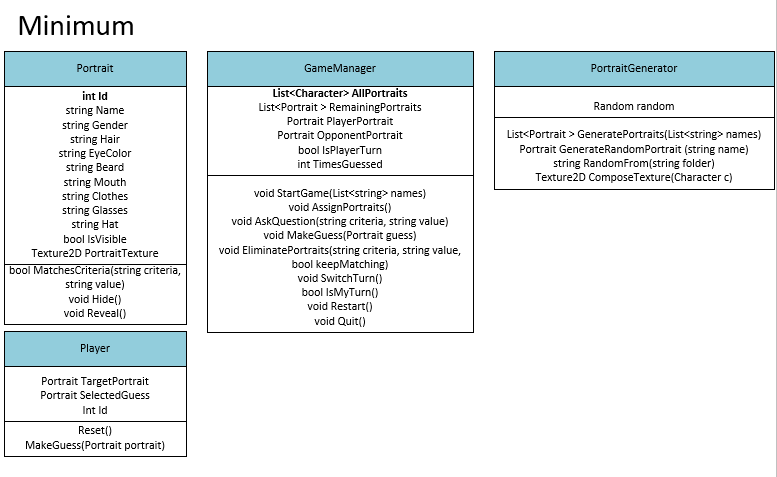
# Vendredi 11.04.2025

## Étapes réalisées

J’ai choisi de réaliser un projet avec Raylib en C# :

Raylib en C# (via Raylib-cs) permet de créer des jeux et applications 2D/3D simples avec gestion graphique, audio et entrées (clavier/souris/manette).

* J’ai également créé deux documents Visio, pour définir les classes que je dois créer au minimum pour que mon « Qui est-ce ? » fonctionne. De la même manière, il a été imaginé les classes qu’il aurait été possible d’ajouter si le projet se déroulait dans les temps. Voici les classes :



Il faut donc développer quatre classes au minimum, dans ma solution Raylib C# en application console, pour que le projet fonctionne. Les paquets suivants ont été importés pour une possible utilisation des svg fournis par le repos « Avataaar » :

* SkiaSharp
* Svg.Skia

Ces paquets serviront pour convertir les images SVG en PNG et ensuite les faire afficher en texture par Raylib.

## Problèmes rencontrés

* La récupération des fichiers SVG à partir du site Avataaars s’est révélée complexe. Extraire les textures directement depuis la page web a présenté trop d’obstacles techniques, notamment en raison du rendu dynamique et de la structure éclatée du code. Le dépôt GitHub, quant à lui, contenait un grand nombre de fichiers, rendant la localisation du code SVG difficile et chronophage.

Le code étant majoritairement écrit en TypeScript avec une architecture HTML dense et peu lisible, il n’a pas été jugé pertinent de pousser l’exploration plus loin. Une décision a donc été prise : plutôt que de s'attarder sur ce fonctionnement interne, il a semblé plus efficace de concevoir une solution autonome en reconstituant l’algorithme de génération, pour gagner en flexibilité et en compréhension globale du processus.

J’ai donc supprimé les paquets SkiaSharp et Svg.Skia mentionnés plus haut et abandonnée le repos de « Avataaars » bien qu’il fût exactement ce que je cherchais.

## Étapes suivantes

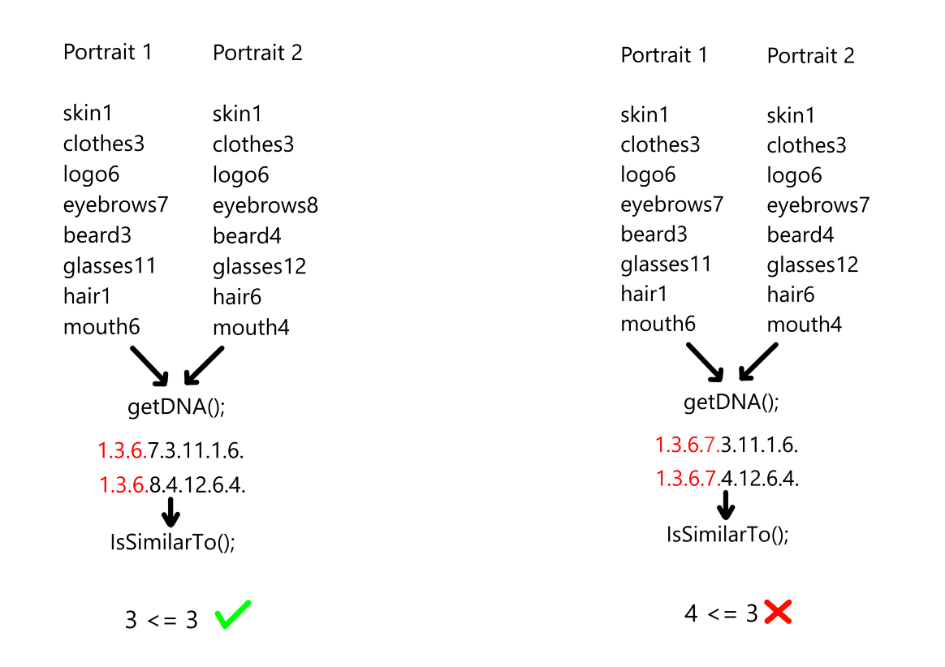
* Algorithme de création de portraits sans sosies.

# Lundi 28.04.2025

## Étapes réalisées

**Portrait**

* La mise en place de l’algorithme de randomisation de portrait a été commencée et terminée, va suivre un aperçu de son fonctionnement.
* Un portrait est défini par les propriétés suivantes :   
  Id, Nom, Skin, Clothes, Logo, Eyebrows, Eyes, Beard, Glasses, Hair et Mouth.
* Lorsqu’une texture est aléatoirement choisie, dans un répertoire pour créer un portrait, il est vérifié que le portrait final n’a pas plus de 3 propriétés en commun avec un autre portrait déjà créé. Pour récupérer les similitudes une méthode « **GetDNA** » qui renvoi l’ADN ou la combinaison de chaque portrait a dû être ajoutée. **GetDNA** est appelé dans une méthode « isSimilarTo » préexistante.

Ces deux méthodes sont utilisées lors de la génération aléatoire, si un portrait a plus de 3 gènes en commun on regénère un portrait. Ci-dessous, un schéma qui illustre tout ce cheminement :

Afficher des textes comme sur le schéma au lieu de textures, a été plus pertinent pour obtenir d’abord l’algorithme souhaité.

## Problèmes rencontrés

Aucun problème rencontré.

## Étapes suivantes

* Créer toutes les classes du schéma « minimum »

# Mardi 29.04.2025

## Étapes réalisées

**GameManager**

* La classe GameManager est l’élément central de la logique du jeu. Il a d’abord fallu définir une structure claire pour gérer les différents états du jeu.   
  Pour cela, une énumération GameState a été introduite, regroupant tous les écrans nécessaires : Menu, Options, Generation, Creating, InGame, Guessing et Victory.
* Plusieurs variables de contrôle ont ensuite été ajoutées, notamment pour suivre la progression du jeu. Par exemple, pour récupérer à qui est le tour actuel « currentPlayerTurn ».
* La méthode « Update() » dans le GameManager détecte tout changement d’état et appelle toutes les classes nécessaires, par exemple l’UIManager qui dessinera la bonne interface au bon moment ou encore la méthode « Generate() » pour générer les portraits de la partie.

**UIManager[[1]](#footnote-1)**

* **UIManager** s’occupe de dessiner toutes les interfaces, les textures et les boutons du jeu. Cette classe orchestre l’affichage dynamiquement en fonction de l’état du jeu (menu, partie, génération, création, options, victoire).
* La première interface créée était celle du menu principal qui propose des boutons qui déclencheront des transitions d’état via le **GameManager**. La seconde interface était celle du jeu présentant les deux plateaux des deux joueurs et les portraits générés.
* Intégration de l’*UIManager* pour permettre l’élimination des portraits via la détection des clics utilisateur.
* Liaison entre l’interface graphique et la logique du *Board*, rendant les interactions en jeu plus intuitives et réactives.

## Problèmes rencontrés

* Oubli de passer les fichiers images en « Toujours copier » dans Visual Studio 2022  
  pour tester en Debug.
* Chemin d’accès différents, « \\ » au lieu de « / » à cause des fonctions de Raylib.
* Changement de l’état GameState qui pose un problème, textures clandestines qui remplacent le fond de la fenêtre.
* Perte de mémoire entre les états.
* Les textures sont très pixelisées.

## Étapes suivantes

* Classe Player
* Classe PortraitGenerator
* Choisir un portrait à faire deviner.

# Vendredi 02.05.2025

## Étapes réalisées

**PortraitGenerator**

* La classe *PortraitGenerator* permet de créer automatiquement des portraits uniques en combinant plusieurs attributs (cheveux, vêtements, barbe, etc.) choisis aléatoirement. Chaque attribut est classé par rareté (*common*, *uncommon*, *rare*, *legendary*) avec une pondération spécifique (ex. : 50 % pour *common*, 1 % pour *legendary*), assurant diversité et rareté visuelle.
* La méthode *CreateRandomPortrait()* assemble un portrait complet tout en respectant certaines règles : par exemple, pas de logo si le vêtement est rare/légendaire, pas de barbe pour les femmes, et cheveux sélectionnés depuis des dossiers distincts selon le genre. Les noms sont choisis via des fichiers spécifiques (masculin/féminin), garantissant unicité et cohérence.
* La méthode *GeneratePortraits()* génère un ensemble de portraits différents avec contrôle de similarité, puis les duplique pour les deux joueurs. Ce système robuste offre une grande rejouabilité tout en maintenant une logique esthétique et identitaire forte.

**PortraitRenderer**

* La classe *PortraitRenderer* est responsable de la gestion graphique des portraits dans le jeu. Elle centralise le chargement, le rendu et la libération des textures afin d’optimiser les performances et éviter les doublons.  
  Un dictionnaire privé a été mis en place pour stocker les textures déjà chargées. Cela permet d’éviter des appels redondants à la mémoire GPU.
* La méthode *LoadPortraitTextures()* se charge de préparer un portrait complet en mémoire avant son affichage. Chaque couche graphique (peau, vêtements, yeux, cheveux, etc.) est passée à une méthode privée *LoadValidTexture()*, qui vérifie la validité du chemin et évite tout rechargement inutile.
* La méthode *DrawPortrait()* s’occupe de l’affichage à l’écran. Elle superpose les différentes textures dans l’ordre prévu, à une position et taille données. En cas d’élimination du personnage, un filtre noir est appliqué au rendu pour en indiquer visuellement l’état.
* Enfin, la méthode *UnloadAll()* permet de libérer proprement toutes les textures de la mémoire GPU à la fin d’une partie ou lors d’un changement d’état.  
  Cette classe assure ainsi une gestion fine des ressources graphiques, essentielle pour maintenir une fluidité constante pendant le jeu.

## Problèmes rencontrés

* L’ordre des textures devait être défini avec la méthode DrawPortrait, PortraitRenderer permet d’alléger le travail de UIManager

## Étapes suivantes

* Classe Player
* Travail d’interface UX/UI

# Lundi 05.05.2025

## Étapes réalisées

**Player**

* La classe **Player** a été développée pour représenter un joueur. Cette classe incarne à la fois **l’état** et les **actions** possibles d’un joueur au cours d’une partie.
* **Board** : Chaque joueur possède son propre plateau de jeu, initialisé via un tableau de portraits transmis au constructeur.
* **TargetPortrait et SelectedGuess** : Ces propriétés représentent respectivement le portrait à deviner et le portrait actuellement sélectionné comme devinette.
* **Méthode MakeGuess** : Elle compare la sélection du joueur avec le portrait cible et retourne un booléen indiquant si la devinette est correcte.
* **Zone** : Une propriété graphique a été ajoutée pour permettre le positionnement personnalisé du joueur sur l’interface, avec une précision importante sur le type (Raylib\_cs.Rectangle), afin d’éviter toute ambiguïté avec le type System.Drawing.Rectangle.
* Un point notable a été la gestion du conflit entre deux bibliothèques de rectangles. La décision de spécifier explicitement le type ***Raylib\_cs.Rectangle*** s’est imposée afin d’éviter les collisions de noms et de garantir la cohérence graphique avec l’environnement Raylib.
* Quelques ajustements d’interfaces ont été fait afin de permettre la navigation entre plusieurs « pages ».

## Problèmes rencontrés

* Conflits avec ***Raylib\_cs.Rectangle. (Zone du joueur)***

## Étapes suivantes

* Création de la classe Board

# Mardi 06.05.2025

## Étapes réalisées

* **Board**  
  La classe *Board* a été conçue pour représenter le plateau de jeu d’un joueur, c’est-à-dire l’ensemble des portraits visibles qu’il peut analyser et éliminer. Cette classe permet d’alléger considérablement la logique de la classe *Player* en encapsulant la gestion des portraits dans un objet distinct. Plutôt que d’attribuer directement un tableau de portraits au joueur, on lui fournit désormais un *Board* autonome, capable de gérer les opérations courantes de manière centralisée.
* Le constructeur prend un tableau de portraits en paramètre, avec une vérification de nullité pour garantir la robustesse du système.  
  La méthode *EliminatePortraitsByQuestion()* applique une logique d’élimination basée sur un attribut donné (par exemple « Hair ») et une valeur spécifique (ex. : « blonde.png »). Elle utilise un *switch expression* pour déterminer si chaque portrait correspond ou non à la condition posée. Ceux qui ne correspondent pas sont marqués comme éliminés.
* La méthode *Reset()* permet de réinitialiser l’état de tous les portraits, utile pour recommencer une partie ou revenir à un état neutre du plateau.
* Le joueur interagit désormais avec un plateau unique, sans avoir à gérer les détails internes des portraits. Cela prépare le terrain pour une meilleure extensibilité du système à l’avenir.
* Changement des chemins d’accès pour la génération de portrait afin de centraliser les ressources graphiques dans un répertoire dédié.
* Adaptation du chargement des textures dans *PortraitRenderer* pour refléter la nouvelle structure de fichiers, assurant une meilleure organisation du projet.
* Pour améliorer l’interaction, j’ai ajouté un effet visuel sur les portraits : lorsqu’un portrait est survolé avec la souris, il s’élève légèrement. Cette animation fluide est gérée dans le UIManager, à l’endroit même où les clics sont détectés. Cela permet une navigation plus intuitive et une meilleure lisibilité en jeu.
* Le portrait à deviner est affiché en haut à droite grâce à un Clone() du portrait cible. En affichant une copie indépendante, le joueur garde une vision constante de sa cible, même si elle est masquée en jeu.

## Problèmes rencontrés

* Le portrait à deviner est affiché en haut à droite. Au départ, j’utilisais directement le portrait original, ce qui le rendait invisible en haut de l’écran lorsqu’il était masqué sur le plateau, car les deux affichages étaient liés. J’ai corrigé ce problème en utilisant la méthode Clone() pour afficher une copie indépendante du portrait cible.

## Étapes suivantes

* Création de la classe SoundManager

# Jeudi 08.05.2025

## Étapes réalisées

**SoundManager**

* Mise en place d’un système audio interactif visant à renforcer l’immersion du joueur à travers des sons contextuels, tels que les clics, validations ou transitions entre écrans. Chaque action clé bénéficie désormais d’un retour sonore adapté.
* Développement d’un menu *Options* complet, permettant un contrôle granulaire de l’ambiance sonore du jeu. L’utilisateur peut ajuster séparément le volume de la musique de fond et celui des effets sonores. Des boutons de type *mute/unmute* ont également été ajoutés pour permettre un contrôle rapide, notamment la possibilité de couper la musique sans affecter les effets sonores.

(Ajout de l’état Option au GameManager)

* Refonte complète de l’interface utilisateur pour la rendre responsive, avec une adaptation dynamique à différentes tailles et configurations d’écran. Deux modes ont été implémentés : double écran et écran unique, offrant à l’utilisateur une expérience flexible selon son matériel.
* Cette évolution a nécessité une restructuration en profondeur de l’UIManager afin de gérer l’affichage conditionnel des éléments. Chaque fenêtre s’adapte désormais automatiquement au mode sélectionné, et l’ensemble de l’UI a été repensé pour garantir une lisibilité optimale, que ce soit sur un seul écran ou répartie sur deux moniteurs.

## Problèmes rencontrés

* Bien que les effets sonores étaient mutés dans la page d’option, lorsque l’on quittait la page d’option l’état revenait à zéro. Il a suffi de changer l’ordre d’appels des méthodes dans le GameManager lorsque l’état revenait au Menu. (Appeler la méthode UpdateSFX() en premier, j’ai créé une méthode « LoadUIAndSounds », qui s’occupait de charger les sons et textures de l’UI)

## Étapes suivantes

* Ajout de l’écran de fin (Après avoir gagné la partie)
* Fonctionnalités supplémentaires (Déplacement de souris, logique de portraits)

# Vendredi 09.05.2025

## Étapes réalisées

Plusieurs ajustements majeurs ont été réalisés, sur la classe PortraitGenerator, pour renforcer l’authenticité et la lisibilité des personnages :

* Nom choisi selon le genre : la génération du nom repose désormais sur des fichiers distincts pour les prénoms masculins et féminins. Le nom est sélectionné aléatoirement en tenant compte du genre défini par l’attribut "Gender", garantissant une cohérence identitaire.
* Restrictions liées au genre : certains éléments graphiques sont désormais conditionnés par le genre. Par exemple, les portraits féminins ne peuvent plus se voir attribuer de barbe — une vérification est faite au moment de l’appel à CreateRandomPortrait(), substituant automatiquement une texture « noBeard » en cas de genre féminin.
* Dossiers séparés pour les cheveux selon le genre : les assets capillaires ont été réorganisés dans des dossiers distincts (malehair et femalehair). Cette séparation a nécessité une adaptation de la méthode GetRandomAsset() pour naviguer dynamiquement selon le genre du portrait en cours de génération.

Également pour la classe UIManager :

* **Placement de la souris selon le tour** : en fonction du joueur actif, le curseur est automatiquement repositionné pour faciliter la sélection du portrait mystère à gauche ou à droite de l’écran, selon le mode de jeu (écran unique ou double écran).
* **Écran de fin**: Lorsque le dernier portrait restant sur le plateau d’un des joueurs, correspond à la cible de l’adversaire, la partie se termine disposant le vainqueur et le perdant sur les deux parties distinctes de l’écran.

## Problèmes rencontrés

Aucun problème rencontré.

## Étapes suivantes

* Révision du projet dans sa totalité

# Lundi 12.05.2025

## Étapes réalisées

* Ajout de nouveaux commentaires plus explicites dans tout le code. Dernières retouches d’interface et derniers tests.
* La programmation touchant à sa fin, les objectifs principaux étants accomplis et le projet ayant atteint un bon niveau de développement, il est prêt à être exposé aux experts pour la présentation.
* Le protocole de tests a été rédigé.
* Ne reste plus qu’à terminer la documentation dans le rapport.

## Problèmes rencontrés

* Paquet Raylib.dll non référencé après le premier clonage du projet.

## Étapes suivantes

* Rédaction du rapport
* Rédaction du Websummary

# Mardi 13.05.2025

## Étapes réalisées

* Rédaction du Websummary pour résumer le projet sur une seule page. Faire une brève description, définir les objectifs et expliquer la réalisation.
* Rédaction du rapport.

## Problèmes rencontrés

Aucun problème rencontré.

## Étapes suivantes

* Rédaction du rapport

# Jeudi 15.05.2025

## Étapes réalisées

* Rédaction du rapport.

## Problèmes rencontrés

Aucun problème rencontré.

## Étapes suivantes

* Rédaction du rapport

# Vendredi 16.05.2025

## Étapes réalisées

* Rédaction du rapport.

## Problèmes rencontrés

Aucun problème rencontré.

## Étapes suivantes

* Rédaction du rapport

# Lundi 19.05.2025

## Étapes réalisées

* Rédaction du rapport.

## Problèmes rencontrés

Aucun problème rencontré.

## Étapes suivantes

* Rédaction du rapport

# Mardi 20.05.2025

## Étapes réalisées

* Rédaction du rapport.

## Problèmes rencontrés

Aucun problème rencontré.

## Étapes suivantes

* Rédaction du rapport

# Jeudi 22.05.2025

## Étapes réalisées

Rédaction du rapport.

## Problèmes rencontrés

Aucun problème rencontré.

## Étapes suivantes

* Rédaction du rapport

# Vendredi 23.05.2025

## Étapes réalisées

**Rendu de tous les documents et du projet pour l’évaluation.**

## Problèmes rencontrés

Aucun problème rencontré.

## Étapes suivantes

* Rédaction de la présentation

1. *Bien que cette classe ne fût définit comme une priorité, (voir schéma « Minimum ») la bonne présentation des données et l’aspect visuel d’un projet me tiens très à cœur et j’ai tout de même pris le risque d’engager le développement de cette classe secondaire.* [↑](#footnote-ref-1)