**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](#_Toc198036875)

[2. Jeudi 10.04.2025 5](#_Toc198036876)

[3. Vendredi 11.04.2025 6](#_Toc198036877)

[4. Lundi 28.04.2025 8](#_Toc198036878)

[5. Mardi 29.04.2025 9](#_Toc198036879)

[6. Vendredi 02.05.2025 10](#_Toc198036880)

[7. Lundi 05.05.2025 11](#_Toc198036881)

[8. Mardi 06.05.2025 11](#_Toc198036882)

[9. Jeudi 08.05.2025 11](#_Toc198036883)

[10. Vendredi 09.05.2025 11](#_Toc198036884)

[11. Lundi 12.05.2025 11](#_Toc198036885)

[12. Mardi 13.05.2025 11](#_Toc198036886)

Après chaque journée de travail, la documentation comme : la planification ou 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

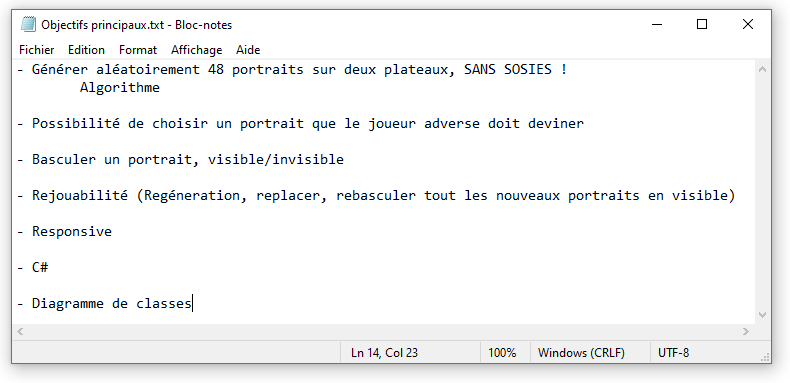
## Étapes réalisées

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#

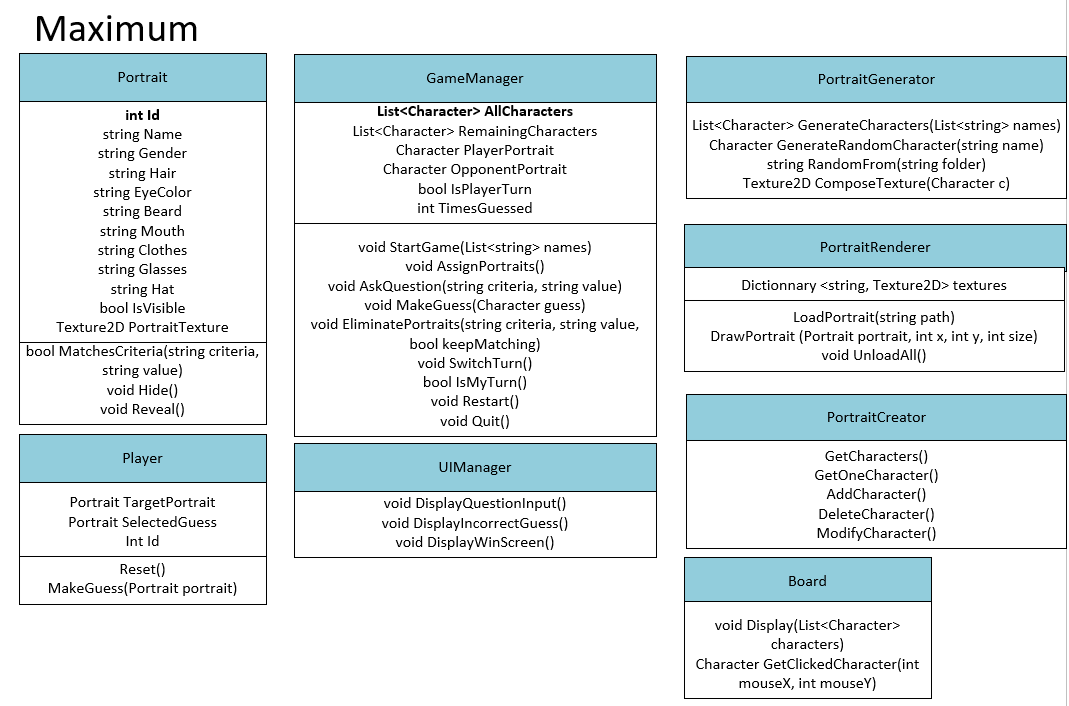
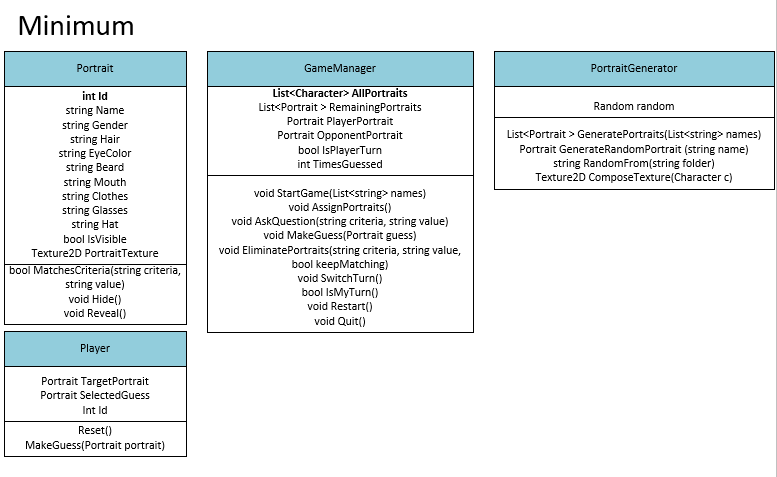
# Vendredi 11.04.2025

## 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 était exactement ce que je cherchais.

## Étapes suivantes

* Algorithme de création de portraits sans sosies.

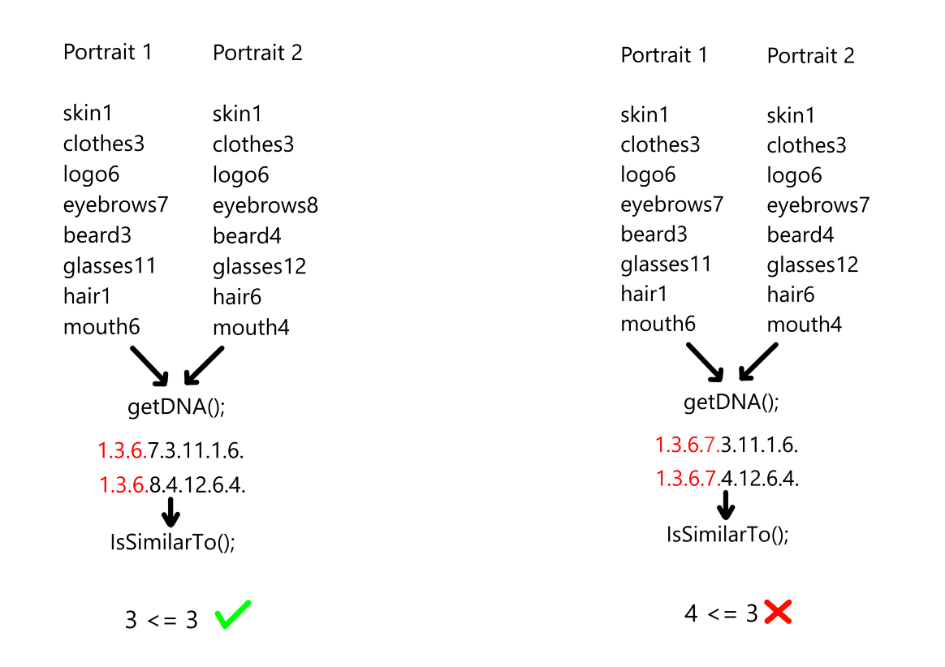
# Lundi 28.04.2025

## Étapes réalisées

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

**PortraitGenerator**

Il a été mis à disposition l’algorithme de création de portraits aléatoires sans sosies à un nouvelle classe « **PortraitGenerator** » qui contient les méthodes :

* **GeneratePortraits**
* **CreateRandomPortrait**
* **GetRandomAsset**

« **GeneratePortraits** » appelle « **CreateRandomPortrait** » 24 fois pour les 24 portraits différents.

Si la génération respecte le maximum de gènes similaires alors on l’ajoute à la liste de portrait qui sera affichée plus tard.

« **CreateRandomPortrait** » lui, s’occupe de la récupération aléatoire d’un fichier par catégorie en appelant **GetRandomAsset.**   
Lorsque tous ces fichiers sont récupérés ils sont inscrits dans un nouveau portrait.

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

**UIManager** s’occupera 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.

## 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 GameManager et Player
* Choisir un portrait à faire deviner.

# Vendredi 02.05.2025

## 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.

**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.

## Problèmes rencontrés

Conflits avec ***Raylib\_cs.Rectangle.***

## Étapes suivantes

* Finir le GameManager
* Travail d’interface UX/UI

# Lundi 05.05.2025

## Étapes réalisées

## Problèmes rencontrés

## Étapes suivantes

# Mardi 06.05.2025

## Étapes réalisées

Diagramme de classe

## Problèmes rencontrés

## Étapes suivantes

# Jeudi 08.05.2025

## Étapes réalisées

## Problèmes rencontrés

## Étapes suivantes

# Vendredi 09.05.2025

## Étapes réalisées

Ajout de sons interactifs, placement de la souris selon le tour, changement de chemin pour la génération de personnage, menu options avec volume de musique et d’effets sonores, boutons pour muter la musique.

## Problèmes rencontrés

## Étapes suivantes

# 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. 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

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)