**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. 08.04.2025 4](#_Toc198017739)

[Étapes réalisées 4](#_Toc198017740)

[Problèmes rencontrés 4](#_Toc198017741)

[Étapes suivantes 4](#_Toc198017742)

[2. 09.04.2025 5](#_Toc198017743)

[Étapes réalisées 5](#_Toc198017744)

[Problèmes rencontrés 5](#_Toc198017745)

[Étapes suivantes 5](#_Toc198017746)

[3. 11.04.2025 6](#_Toc198017747)

[Étapes réalisées 6](#_Toc198017748)

[Problèmes rencontrés 7](#_Toc198017749)

[Étapes suivantes 7](#_Toc198017750)

[4. 28.04.2025 8](#_Toc198017751)

[Étapes réalisées 8](#_Toc198017752)

[Problèmes rencontrés 8](#_Toc198017753)

[Étapes suivantes 8](#_Toc198017754)

[5. 29.04.2025 9](#_Toc198017755)

[Étapes réalisées 9](#_Toc198017756)

[Problèmes rencontrés 9](#_Toc198017757)

[Étapes suivantes 9](#_Toc198017758)

[6. 02.05.2025 10](#_Toc198017759)

[Étapes réalisées 10](#_Toc198017760)

[Problèmes rencontrés 10](#_Toc198017761)

[Étapes suivantes 10](#_Toc198017762)

[7. 05.05.2025 10](#_Toc198017763)

[Étapes réalisées 10](#_Toc198017764)

[Problèmes rencontrés 10](#_Toc198017765)

[Étapes suivantes 10](#_Toc198017766)

[8. 06.05.2025 10](#_Toc198017767)

[Étapes réalisées 10](#_Toc198017768)

[Problèmes rencontrés 10](#_Toc198017769)

[Étapes suivantes 10](#_Toc198017770)

[9. 09.05.2025 10](#_Toc198017771)

[Étapes réalisées 10](#_Toc198017772)

[Problèmes rencontrés 10](#_Toc198017773)

[Étapes suivantes 10](#_Toc198017774)

[10. 10.05.2025 10](#_Toc198017775)

[Étapes réalisées 10](#_Toc198017776)

[Problèmes rencontrés 10](#_Toc198017777)

[Étapes suivantes 10](#_Toc198017778)

# 08.04.2025

## Étapes réalisées

Lors de la réception et après 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.

## Étapes suivantes

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

# 09.04.2025

## 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, le temps a été pris pour terminer en ajoutant les tâches supplémentaires.

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#

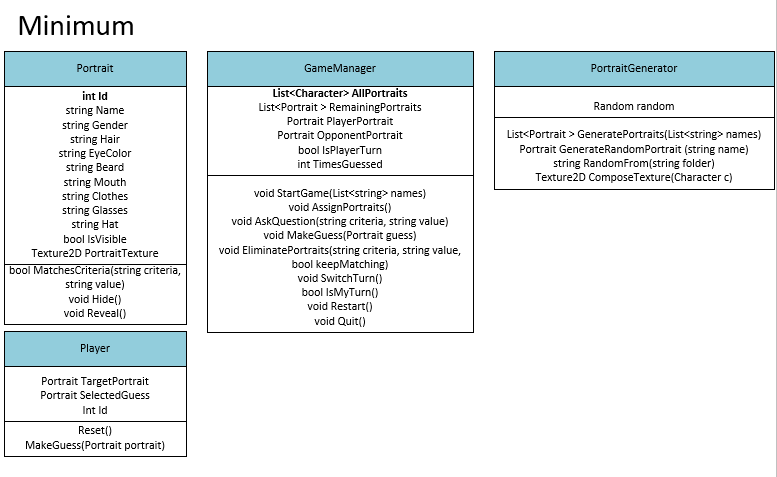
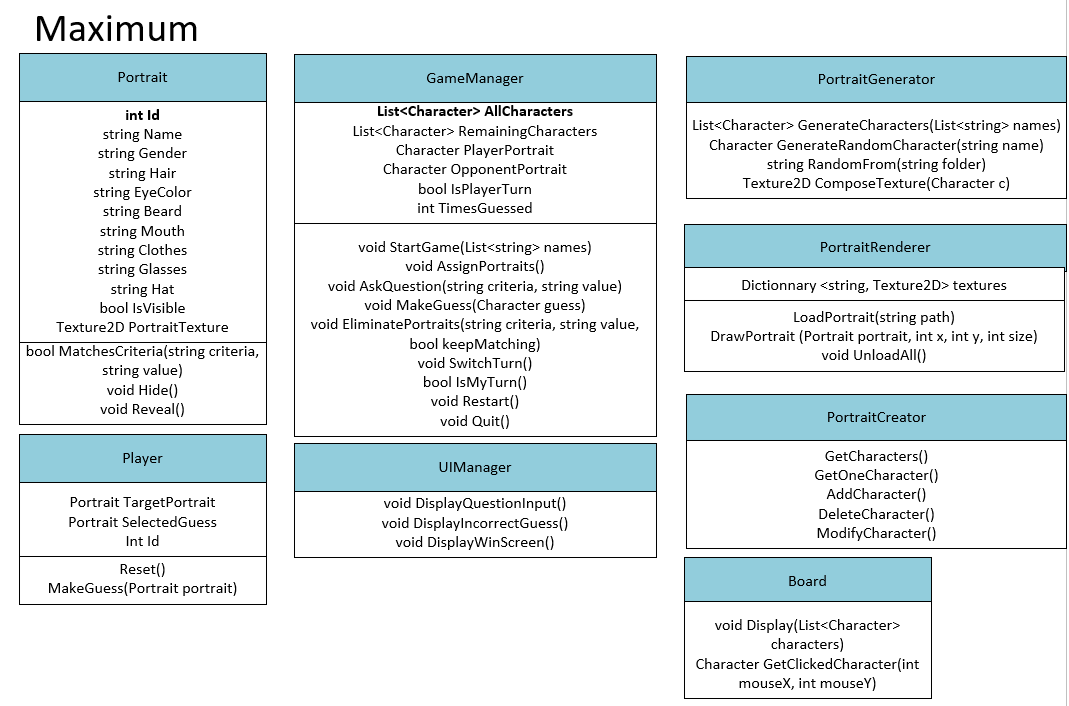
# 11.04.2025

## 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.

Mise à jour du planning et du JT.

## Problèmes rencontrés

Il est très compliqué de trouver un moyen de récupérer chaque partie de l’image SVG séparément. Par exemple uniquement : les yeux, la bouche, les cheveux dans leurs fichiers svg respectifs.

## Étapes suivantes

* Algorithme de création de portraits sans sosies.

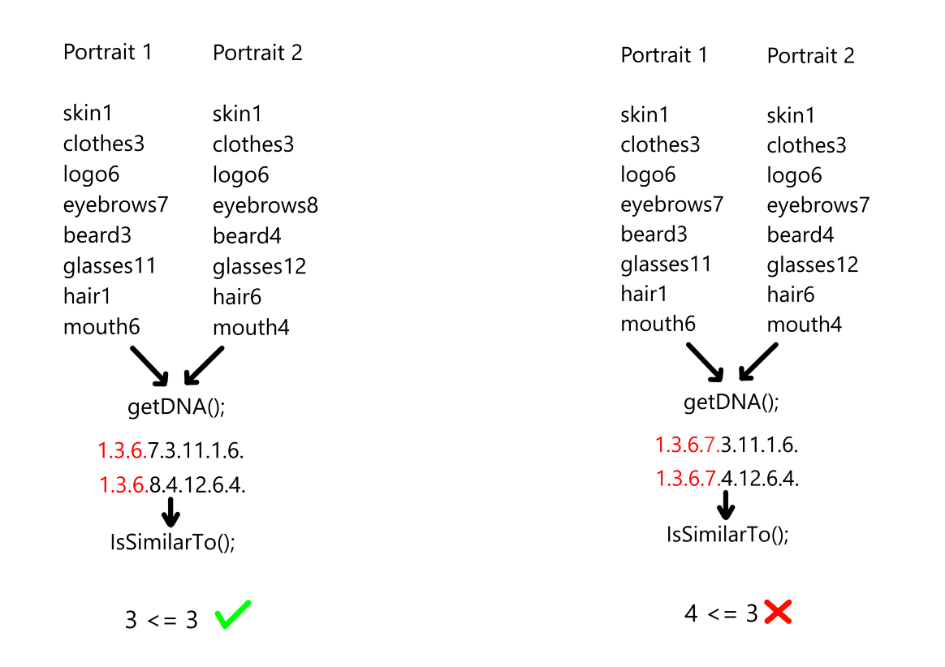
# 28.04.2025

## 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

* Affichage des 48 portrait. 24 pour chaque joueur.
* Choisir un portrait à faire deviner.

# 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**

**UIManager** s’occupera de dessiner toutes les interfaces du jeu. Cette classe orchestre change d’affichage dynamiquement en fonction de l’état du jeu (menu, partie, génération, création, options, victoire) et incarne le visage de la communication avec l’utilisateur.

La première interface créée était celle du menu principal qui propose des boutons qui déclencheront des transitions d’état via le **GameManager**.

## 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.

## Étapes suivantes

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

# 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**

* Mettre en place un **squelette fonctionnel** de la machine à états du jeu via une énumération GameState. (InGame, Menu, Options, etc…)
* Définir les **variables de contrôle** essentielles (état du jeu, tours de joueurs, sélection de portrait).
* Connecter les composants principaux : interface utilisateur (UIManager), rendu des portraits (PortraitRenderer) et les deux joueurs (Player).

Ce qui a été fait :

* Implémentation de l’énumération GameState : permet de structurer les différentes phases du jeu — du menu principal jusqu'à l'écran de victoire. Cela servira de fondation pour un cycle de jeu clair et maintenable.
* Initialisation des composants : les méthodes Initialize() et InitializeCreator() ont été définies pour préparer les managers nécessaires selon le contexte (jeu ou création de portraits).
* Gestion du son : une logique conditionnelle a été mise en place pour activer ou désactiver la musique selon la variable isMusicMuted, assurant une flexibilité immédiate pour l’expérience utilisateur.
* Première version de Update() : l’ossature de cette méthode a été construite pour gérer les transitions entre les états Menu, InGame, Options, Creating et Generation. Chaque cas appelle les modules correspondants (UI, son, génération de portraits).
* Séparation des responsabilités : des méthodes comme Generate(), GenerateExample() ou Reset() ont été introduites pour découpler les actions majeures du jeu, facilitant ainsi la lisibilité du code et la maintenance future.

## Problèmes rencontrés

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

## Étapes suivantes

* Finir le GameManager

# 05.05.2025

## Étapes réalisées

## Problèmes rencontrés

## Étapes suivantes

# 06.05.2025

## Étapes réalisées

## Problèmes rencontrés

## Étapes suivantes

# 09.05.2025

## Étapes réalisées

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

## Problèmes rencontrés

## Étapes suivantes

# 10.05.2025

## Étapes réalisées

## Problèmes rencontrés

## Étapes suivantes