**Rapport TPI**

**Rapport réalisé par : Sacha Leone**

Classe : 3IND-3TPMa

Commencé le : 08.04.2025

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

Table des matières

[Introduction 2](#_Toc198561860)

[Description du projet 2](#_Toc198561861)

[Hypothèse 2](#_Toc198561862)

[Outils et méthodes 3](#_Toc198561863)

[Outils/logiciels utilisés 3](#_Toc198561864)

[Sources utiles 3](#_Toc198561865)

[Méthodes 3](#_Toc198561866)

[Contexte 6](#_Toc198561867)

[Réalisation 7](#_Toc198561868)

[Les classes 7](#_Toc198561869)

[Les interfaces 21](#_Toc198561870)

[Les textures 29](#_Toc198561871)

[Les sons et les musiques 31](#_Toc198561872)

[Représentation graphique du chemin du code 31](#_Toc198561873)

[Résultats (Critères) 31](#_Toc198561874)

[Conclusion 31](#_Toc198561875)

[Problèmes rencontrés 31](#_Toc198561876)

[Améliorations possibles 31](#_Toc198561877)

[Futur de l’application 31](#_Toc198561878)

[Avis personnel 31](#_Toc198561879)

[Annexes 31](#_Toc198561880)

# Introduction

Dans le cadre du TPI, il a fallu réaliser un projet informatique sous 110 périodes. Ce projet a été supervisé par M. Schenk qui, lorsque cela s'est avéré nécessaire, a été consulté pour recevoir divers conseils et indications. Le TPI est l’examen final à réaliser en vue de l’obtention du CFC comme informaticien orienté développement.

Les critères propres à mon projet à respecter sont les suivants :

* C02 - Développer des modèles de données
* C03 - Implémenter un modèle de données
* C09 - Analyser, identifier et vérifier la validité des données
* C10 - Préparer, présenter et évaluer les données
* G01 - Documentation des exigences métier et techniques
* G08 - Elaboration d'un concept de réalisation
* G11 - Elaboration de concepts de tests et définition des cas de tests
* G12 - Exécution et évaluation des tests

Les différents éléments à rendre pour l’évaluation sont les suivants **: un rapport, un journal de travail, un websummary et le travail réalisé**.   
Ce travail une fois terminé, doit être synthétisé puis exposé à l’aide d’un support de présentation, devant deux experts : M. Del Torchio et M. Jeanmaire.

## Description du projet

**Problématique**  
Dans le jeu "Qui est-ce ?" classique, les personnages sont toujours identiques et disposés aux mêmes emplacements, ce qui nuit à la re-jouabilité.

**Solution proposée**  
Pour pallier cette limitation, un système de génération aléatoire de portraits sera mis en place. Chaque partie proposera une nouvelle combinaison de personnages uniques, générés sans doublons (ni sosies), et placés aléatoirement sur la grille.

**Implémentation**  
Un algorithme de génération de portraits combinera aléatoirement différents attributs visuels (cheveux, yeux, accessoires, etc.) pour créer des visages uniques. Ces portraits seront ensuite distribués de manière aléatoire sur le plateau de jeu à chaque nouvelle partie.

**Objectif final**  
Produire un rendu visuel cohérent et dynamique de ces données, assurant une expérience de jeu renouvelée à chaque session.

## Hypothèse

Dans le cas où tous les objectifs minimums sont atteints (voir planification), il est envisageable de développer plus librement. Par exemple, en ajoutant des fonctionnalités, des effets visuels/sonores, des mécaniques supplémentaires, qui ne sont pas spécifiées dans le cahier des charges.   
Ce développement libre doit, tout de même, respecter les conditions définies initialement et ne pas nuire au fonctionnement principal.

# Outils et méthodes

## Outils/logiciels utilisés

* Ordinateur de l’école (Windows 10)
* [Visual Studio 2022](https://visualstudio.microsoft.com/fr/vs/) (éditeur de code) avec la [librairie Raylib](https://www.raylib.com/) en **C# Application console**.
* [GitHub](https://github.com/) (Déposer le code et nommer les étapes avec les commit)
* [Krita](https://krita.org/fr/) (Logiciel de dessin, montage)

INSERER LOGO ET AUTRES APPLIS

## Sources utiles

* [Raylib (cheatsheet)](https://www.raylib.com/cheatsheet/cheatsheet.html) (Utilisation de la librairie)
* Recherches sur Internet et YouTube
* [ChatGPT](https://chatgpt.com/) (Commentaires, debug et corrections)

## Méthodes

Premièrement, la prise de connaissance du cahier des charges est la priorité avant d’entamer le projet. Après la lecture les objectifs principaux ont été extraits et synthétisés dans une note simple :

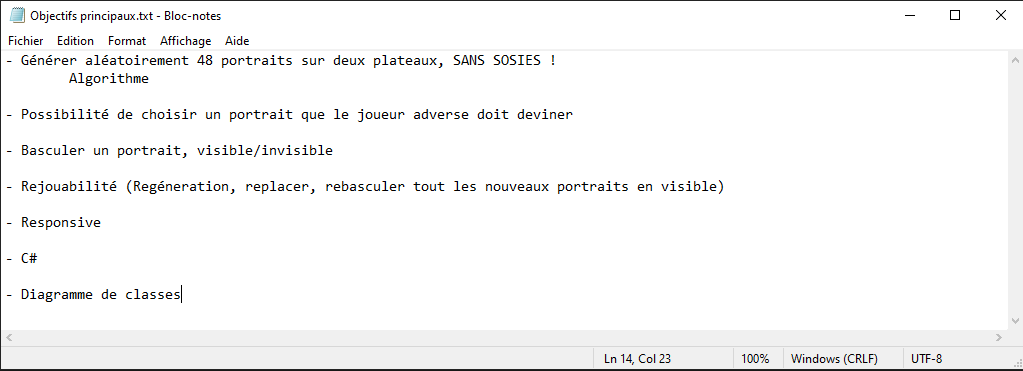


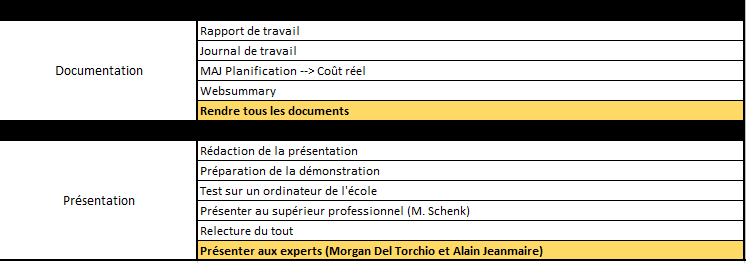
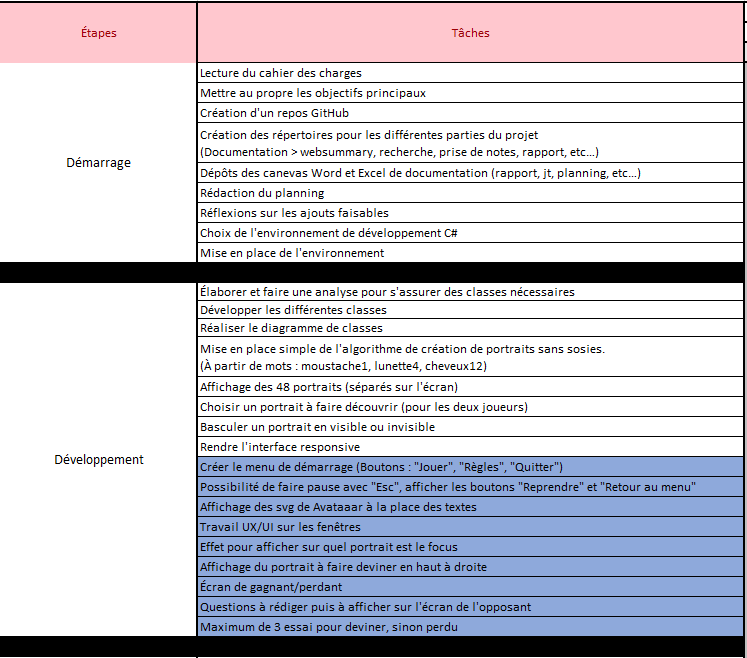
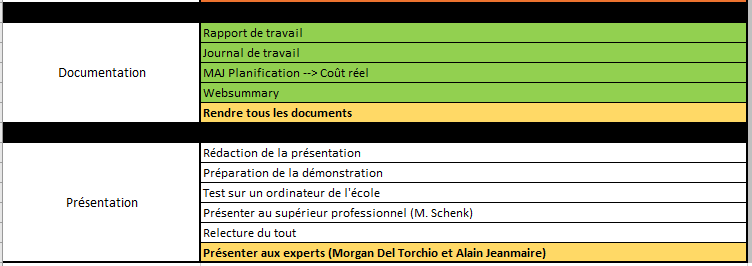
Figure 1 : Screenshot de notes rapides, objectifs

Une planification propre, sous forme de GANTT, listant toutes les tâches définies et leur durée est rédigée dès le début. (Coûts estimés)  
Celle-ci est ensuite quotidiennement mise à jour. (Coûts réels)

INSERER FICHIER GANTT

Figure 2 : Screenshot des tâches (Coûts réel)

Figure 3 : Screenshot des tâches (Coûts estimés)



Chaque jourde travail il est également demandé de remplir un journal de travail (JT), afin d’avoir un suivi sur l’avancement du projet et sur les problèmes rencontrés.  
*(La rédaction du JT, est soutenue par le code présent dans les commits sur GitHub, les notes prisent pendant le développement et la planification.)*

Pour avoir un avancement méthodique et efficace, la création d’un repos Git a été cruciale, sur le repos se trouve l’entièreté du projet, autant la documentation que le code.

De cette manière non seulement les compartiments du projet seront proprement séparés, mais ils seront aussi atteignables sur tous les ordinateurs du CPNE et également sur mon ordinateur personnel. La mise en place de se repos contribue à un projet : en sécurité, atteignable et facilement mis à jour.

Voici l’arborescence du repos qui a permis de maintenir le bon ordre des fichiers :

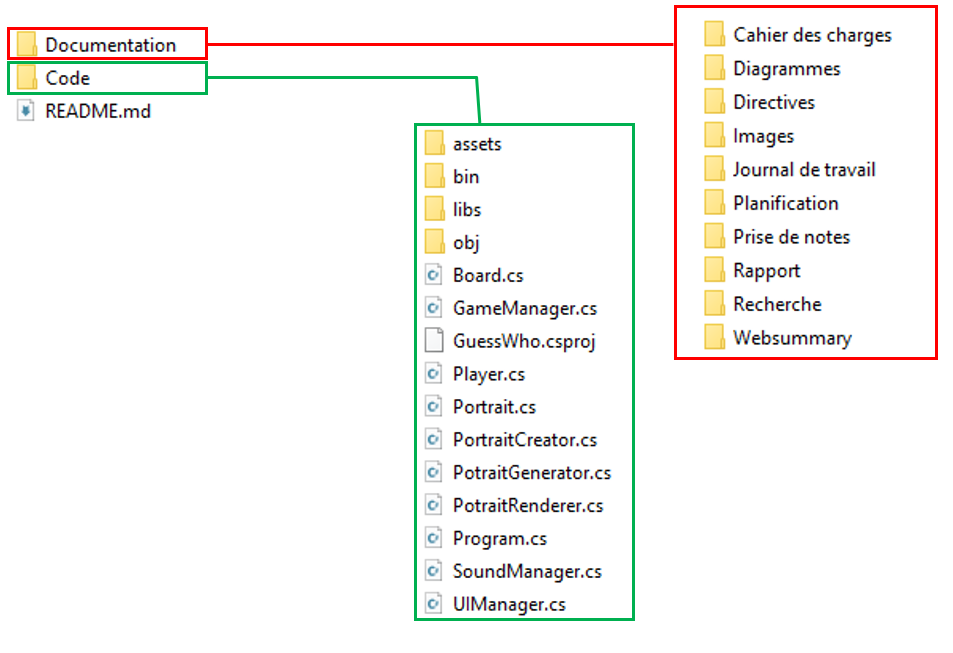


Figure 4 : Screenshot de l'arborescence

Pour résumer, une journée de travail méthodique suit les étapes suivantes :

* Clonage du projet depuis GitHub
* Consultation de la planification
* Développement (prise de notes en cas de problème)
* Rédaction du journal de travail
* Mise à jour de la planification
* Push du/des commit(s) sur le repos

Ainsi de suite jusqu’à l’échéance.

# Contexte

Langage, spécifications, facilité, connaissances personnelles

Raylib, pourquoi l’avoir chois

# Réalisation

## Les classes

Afin d’assurer un MVP (Produit minimum viable) il a été crucial d’établir des limites aux nombres de classes et fonctionnalités qui auraient été créés. **Suivent deux schémas**, l’un listant le minimum de classes nécessaires au fonctionnement du jeu. Puis, l’autre représentant le maximum de classes dans le cas où le minimum est accompli :

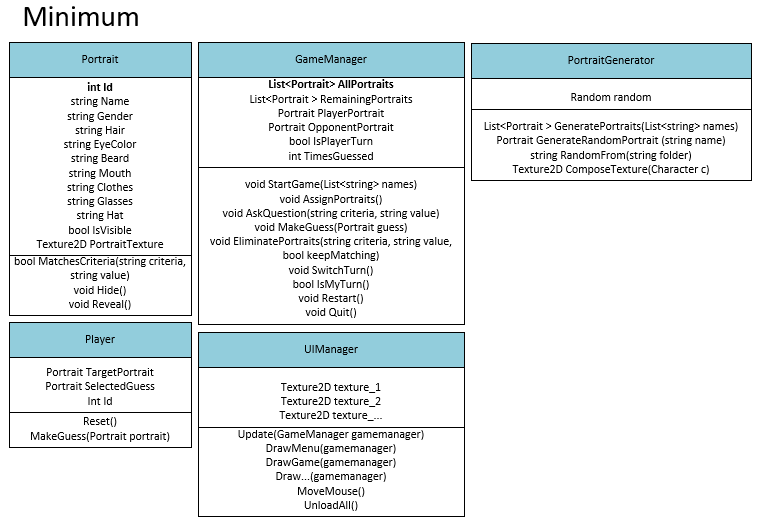
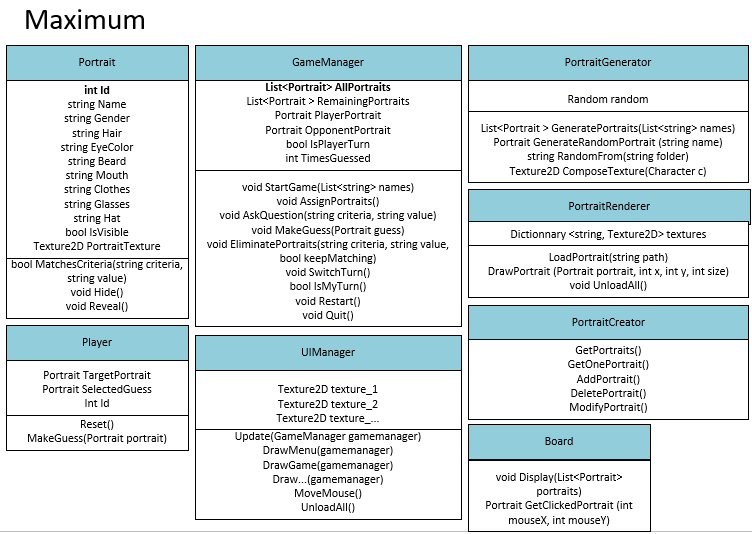
Évidemment, lors de la programmation, certaines retouches ont été réalisées sur les classes pour atteindre les fonctionnements souhaités, ces schémas ne représentent pas le résultat final des classes.

Figure 5 : Schéma des classes minimum

Figure 6 : Schéma des classes maximum

Il est temps d’expliquer en détail le rôle et les tâches que remplissent les classes qui sont aujourd’hui fonctionnelles.

NE PAS TROP VULGARISER ON PARLE A DES DEV.

### Portrait

Rôle :  
Représente un personnage avec un ensemble unique de caractéristiques visuelles et un état d'interaction dans le jeu.

Propriétés :

* **Id** : Identifiant unique du portrait.
* **Name, Skin, Clothes, Logo, Eyebrows, Eyes, Beard, Glasses, Hair, Mouth, Gender**

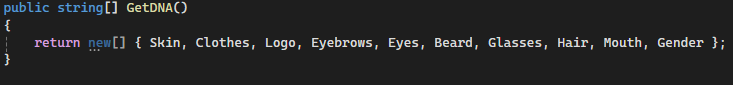
Caractéristiques visuelles constituant l’ADN du portrait, toutes requises à l’instanciation.

* **HoverOffset** : Décalage visuel appliqué lors d’un survol (effet d’animation).
* **IsEliminated** : Indique si le portrait est exclu des choix possibles.
* **isTarget** : Désigne le portrait à deviner.
* **CanAppear** : Contrôle la visibilité dans certaines situations de jeu.

Méthodes :

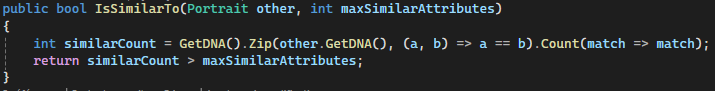
* **string[] GetDNA()**

Retourne un tableau des attributs constituant l’identité visuelle complète du portrait (Son ADN).



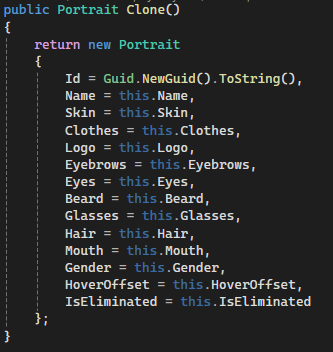
* **bool IsSimilarTo(Portrait other, int maxSimilarAttributes)**

Compare le portrait à un autre et renvoie true si le nombre d’attributs similaires dépasse le seuil autorisé.



* **Portrait Clone()**

Crée une copie indépendante du portrait avec un nouvel Id, conservant tous les attributs visuels et états sauf isTarget.



Particularités :

* Le clonage est essentiel pour dupliquer des plateaux tout en gardant des objets distincts.
* La méthode **IsSimilarTo** permet de réguler la diversité visuelle lors de la génération.

### PortraitGenerator

Rôle :Génère un ensemble de portraits aléatoires en assurant la diversité visuelle, avec une logique de rareté et de duplication maîtrisée.

Propriétés :

* **Random random** : Générateur pseudo-aléatoire pour tous les choix d’éléments visuels.

Méthodes :

* **Portrait[] GeneratePortraits(int count, int MaxSimilarAttributes)**

Génère un tableau de count × 2 portraits.  
Assure que chaque portrait original n’est pas trop similaire aux autres (selon un seuil), puis le duplique.

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.Voici un schéma explicatif :

Figure 7 : Exemple de comparaison d'ADN

* **static string GetRandomName(string gender, List<string> alreadySelectedNames)**

Sélectionne un prénom unique à partir d’un fichier genre-spécifique.  
Garantit que le nom n’a pas été attribué à un portrait précédent.

* **static List<string> LoadNames(string filePath)**

Lit un fichier de noms séparés par des virgules et renvoie une liste nettoyée et sans doublons.

* **Portrait CreateRandomPortrait(string id)**

Assemble un portrait à partir d’éléments aléatoires : peau, vêtements, accessoires, etc.  
Tient compte du genre pour adapter certains attributs (cheveux, barbe, prénoms).  
Assigne un identifiant unique basé sur le nom et l’index.

* **string GetRandomAsset(string category)**

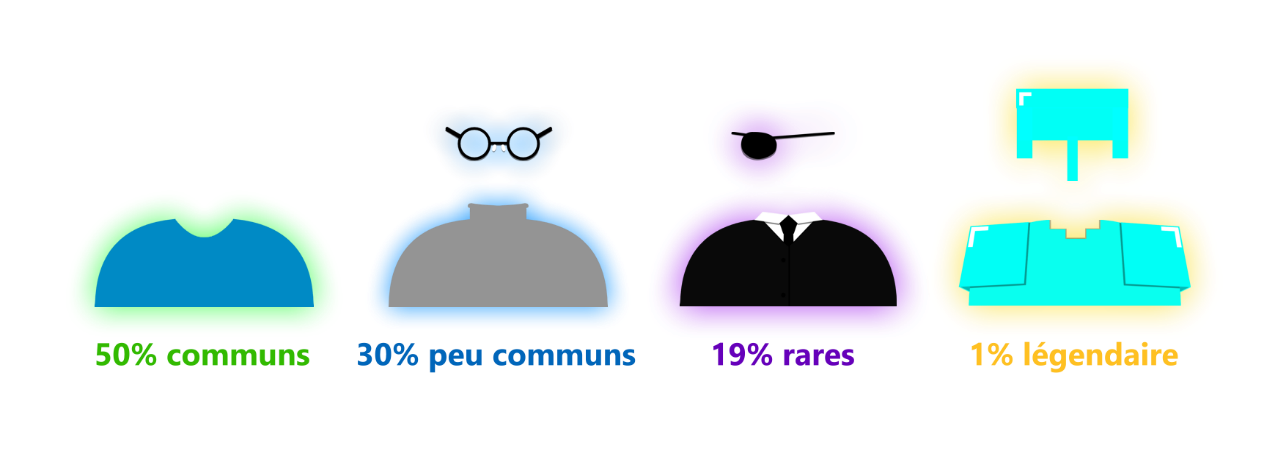
Sélectionne aléatoirement un fichier image d’une catégorie (ex. : "hair", "clothes") en respectant une logique de rareté :

Figure 8 : Raretés des textures

* **string[] FilterFiles(string[] allFiles, Func<string, bool> predicate)**

Filtre les fichiers en fonction d’un prédicat booléen.

Particularités :

* La méthode GeneratePortraits empêche les doublons via la méthode IsSimilarTo(...) de la classe Portrait
* Les duplications finales permettent de créer des plateaux symétriques ou comparables pour les deux joueurs.
* La rareté introduit une couche de collection et de différenciation visuelle utile à l’expérience de jeu.

### PortraitRenderer

Rôle :Gère le chargement, l’affichage et la libération mémoire des textures associées à un portrait.

Propriétés :

* **Dictionary<string, Texture2D> textures**

Stocke les textures déjà chargées pour éviter les redondances et optimiser les performances.

Méthodes :

* **void LoadPotraitTextures(Portrait portrait)**

Charge toutes les textures nécessaires à l’affichage d’un portrait donné, couche par couche.

* **void DrawPortrait(Portrait portrait, int x, int y, int size)**

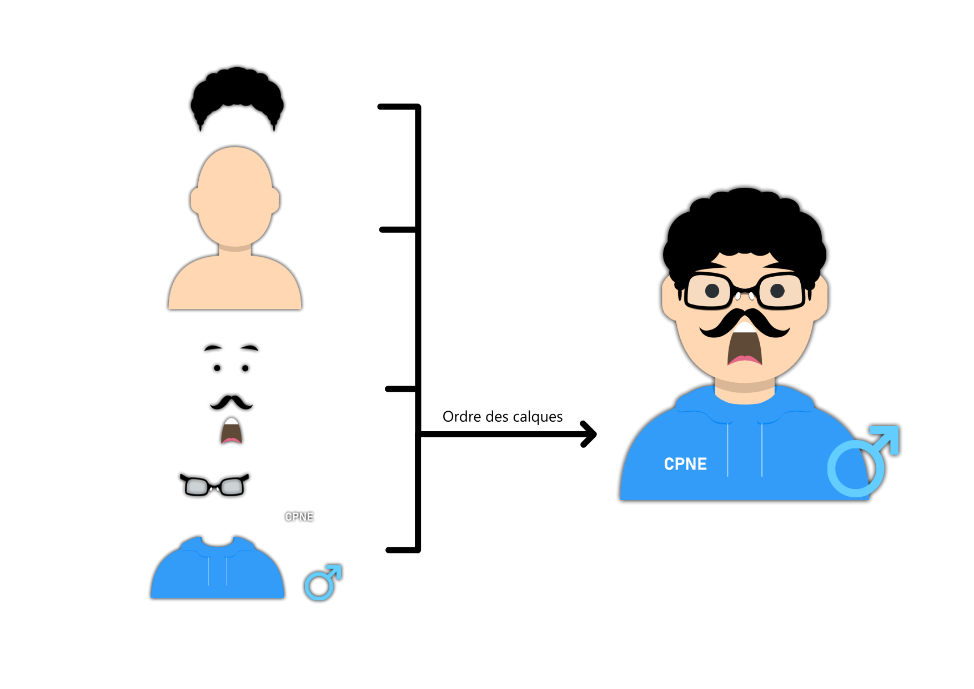
Dessine le portrait à la position (x, y) avec une taille donnée, en superposant les calques dans un ordre précis :  
Peau → Vêtements → Logo → Yeux → Sourcils → Cheveux → Barbe → Bouche → Lunettes → Genre.

Figure 9 : Ordre d'affichage des textures

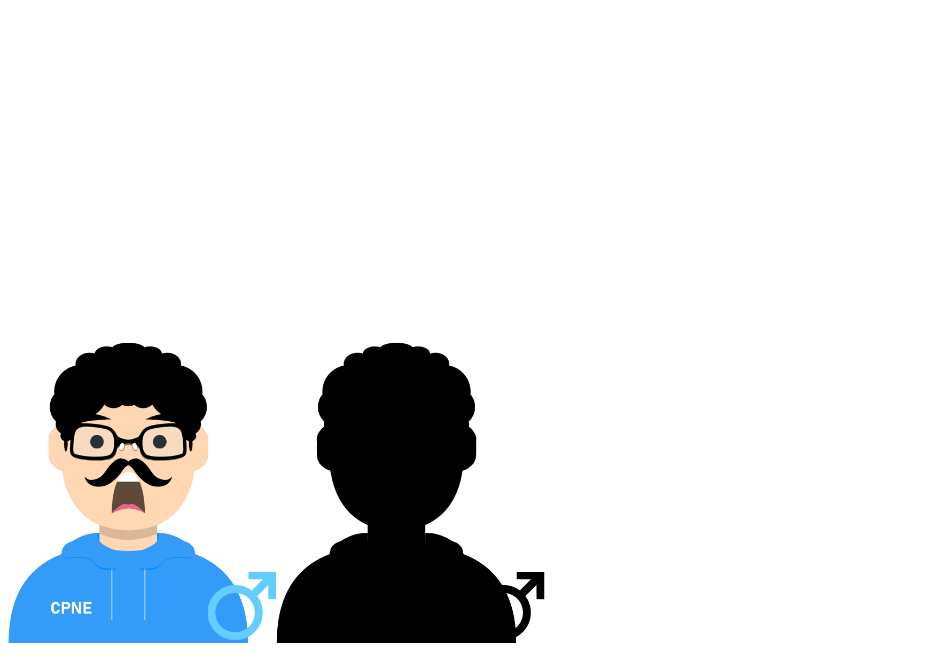
Colore l’ensemble en noir si le portrait est éliminé.

Figure 10 : Portrait masqué (éliminé)

* **void UnloadAll()**

Libère toutes les textures de la mémoire GPU et réinitialise le cache.

* **void LoadPortrait(string path)**

Charge une texture depuis le chemin donné si elle n’est pas déjà présente dans le cache.  
Applique un filtre bilinéaire pour améliorer le rendu visuel.

Particularités :

* L'utilisation du cache (textures) réduit considérablement la latence de chargement.
* L’ordre de superposition garantit une construction graphique cohérente du personnage.
* L’opacité en noir total signale clairement l’élimination d’un portrait.

### Board

Rôle :  
Gère l’ensemble des portraits d’un joueur, leur affichage, leur réinitialisation et leur élimination selon les réponses aux questions.

Propriétés :

* Portrait[] Portraits  
  Tableau des portraits affichés sur le plateau du joueur.

Constructeur :

* **Board(Portrait[] portraits)**

Initialise le plateau avec un ensemble donné de portraits.

Méthodes :

* **void EliminatePortraitsByQuestion(string attribute, string value)**

Élimine tous les portraits ne correspondant pas à la valeur de l’attribut spécifié.  
Utilise un switch pour comparer l’attribut demandé avec la propriété correspondante de chaque portrait.

* **void Reset()**

Réinitialise l’état de tous les portraits en les marquant comme non éliminés.

* **void Display()**

Prévue pour l’affichage des portraits non éliminés. (Implémentation encore vide dans le code actuel.)

Particularités :

* La méthode d’élimination repose sur une correspondance stricte entre une propriété textuelle (attribute) et la valeur recherchée.
* Le plateau est un conteneur passif : il ne gère pas l'affichage graphique, seulement la logique d’état des portraits.

### Player

Rôle :

Représente un joueur du jeu. Gère son plateau, son portrait cible, sa tentative de devinette, ainsi que sa zone d'affichage.

Propriétés :

* **Board Board** : Plateau associé au joueur, contenant tous les portraits.
* **Rectangle Zone** : Zone d'affichage sur l'écran réservée au joueur.
* **Portrait TargetPortrait** : Portrait que l’adversaire doit deviner.
* **Portrait SelectedGuess (settable)** : Portrait sélectionné comme devinette actuelle.
* **int Id** : Identifiant unique du joueur, défini à l’instanciation.

Constructeur :

* **Player(Portrait[] portraits, int id)**

Initialise le joueur avec un identifiant et un plateau généré à partir des portraits fournis.

Méthodes :

* **void Reset()**

Réinitialise le plateau du joueur à son état initial.

* **bool MakeGuess(Portrait guess)**

Enregistre la tentative de devinette et retourne un booléen indiquant si elle est correcte.

### GameManager

Rôle :

Orchestre l’ensemble du cycle de jeu : génération des portraits, gestion des états, transitions, sons, tours, et interface.

Propriétés :

* **GameState (enum)** : Enumération des différents écrans et états du jeu : Menu, Rules, Generation, Creating, InGame, Guessing, Victory, etc.
* **CurrentState :** État courant du jeu, pilote les comportements dans la méthode Update.

gameStarted, portraitsGenerated, generatedExample : Indiquent la progression de la session.

* **isMusicMuted, isSfxMuted :** Gèrent les préférences audio.
* **StateSelectingPortrait :** Active la sélection du portrait secret.
* **Portrait[] allPortraits :** Contient tous les portraits générés.
* **Player player1, player2 :** Deux instances de joueur.
* **PortraitRenderer renderer :** Gère le rendu visuel des portraits.
* **PortraitGenerator generator :** Produit des portraits visuellement distincts.
* **UIManager uIManager et SoundManager soundManager :**

Contrôlent interface et ambiance sonore.

* **int currentPlayerTurn** : Identifie le joueur actif.
* **GetCurrentPlayer(), NextTurn(), ResetTurn() :**

Gèrent le cycle de jeu à tour de rôle.

Méthodes :

* **void Update(GameManager gamemanager)**

Méthode centrale. Active les composants visuels et logiques selon l’état courant (CurrentState).

Chaque état appelle l’interface et déclenche des actions conditionnelles (génération, choix, victoire, etc.).

* **void Generate()**

Génère 48 portraits en deux groupes de 24 pour les deux joueurs.

Charge les textures associées à chaque portrait.

* **void GenerateExample()**

Produit deux portraits à des fins de démonstration ou test du créateur.

* **void CheckVictory(Player guesser, Player opponent)**

Compare le portrait deviné au portrait cible. Si identiques, passe à l’état Victory.

* **void SelectedPortrait(Player player, Portrait portrait)**

Assigne le portrait secret d’un joueur.

* **void Reset(Player player1, Player player2)**

Réinitialise les états de jeu et relance une partie depuis l’étape de sélection.

* **void EndGame(bool player1Won)**

Passe à l’écran de victoire, sans distinction du gagnant dans l’état actuel.

* **void LoadUIAndSounds(GameManager gamemanager)**

Charge les textures et active ou désactive musique/effets selon les préférences.

* **void Initialize() et InitializeCreator()**

Initialisent le gestionnaire d’interface, le son, et les composants selon le mode (jeu ou création).

Particularités :

* Gestion d’un mode double-écran via userHasDualScreen (non encore exploité ici).
* Architecture extensible basée sur des états bien séparés.
* Intègre à la fois la logique pure, l’affichage et les ressources multimédia.

### UIManager

Rôle :  
Gère l’ensemble de l’interface graphique du jeu. Affiche les différents écrans (menu, jeu, sélection, règles, options, génération, création, victoire) et gère les interactions utilisateur.

Propriétés :

* **string[] menuLabels :** Libellés des boutons du menu principal.
* **GameState previousState :** État précédent du jeu, utilisé pour éviter les rechargements inutiles.
* **Model guessWhoTitle :** Modèle 3D affiché dans le menu principal.
* **Texture2D backgroundMenu, backgroundInGame, backgroundInGameSelecting, screenIcon, addCharacter, speakerIcon, sfxIcon, rules1Icon, rules2Icon, rules3Icon :** Textures utilisées dans l’affichage des différents écrans, éléments interactifs, icônes et illustrations.
* **Color trueYellow :** Couleur utilisée pour les éléments survolés.
* **int BasePortraitSize :** Taille de référence des portraits.
* **int cols :** Nombre de colonnes dans la grille de portraits.
* **int page :** Page actuelle dans l’écran des règles.
* **int lastPage :** Dernière page de l’écran des règles.
* **bool inputInitialized :** Indique si la valeur de départ a été assignée à la boîte de saisie.
* **bool inputActive :** Indique si la boîte de saisie est active.
* **string inputText :** Texte actuellement présent dans la boîte de saisie.
* **Rectangle inputBox :** Boîte de saisie utilisée dans l’écran de génération.
* **Camera3D camera :** Caméra utilisée pour afficher le modèle 3D dans le menu.

Méthodes :

* **void UpdateMenu(GameManager gameManager)**

Gère les interactions dans le menu principal et change l’état du jeu selon le bouton sélectionné.

* **void DrawMenu(GameManager gameManager)**

Affiche le menu principal avec fond, boutons, icônes sonores et modèle 3D.

* **void DrawSelectingPortraits(GameManager gameManager)**

Affiche la phase de sélection de portrait avec grilles interactives et message personnalisé pour chaque joueur.

* **void DrawGame(GameManager gameManager)**

Affiche les plateaux de portraits pendant la partie, le portrait cible à deviner et les titres contextuels.

* **void DrawGeneration(GameManager gameManager)**

Affiche l’écran de génération avec une boîte de saisie pour paramétrer les similarités génétiques.

* **void DrawRules(GameManager gameManager)**

Affiche les règles du jeu avec pagination et visuels associés.

* **void DrawCreator(GameManager gameManager)**

Affiche la galerie des portraits créés avec un bouton pour accéder à la génération.

* **void DrawOptions(GameManager gameManager)**

Affiche les options de configuratison : simple/double écran, volumes audio, icônes sonores.

* **void DrawPortraitGrid(Portrait[], PortraitRenderer, Rectangle, int, int, int, int, GameManager)**

Affiche une grille de portraits avec effet de survol et gestion des clics pour élimination ou sélection.

* **void DrawBackToMenuButton(GameManager gameManager, GameState lastState)**

Affiche un bouton permettant de revenir à un autre état du jeu (souvent le menu principal).

* **void DrawEndScreen(GameState state, int winner)**

Affiche l’écran de fin de partie avec le joueur gagnant et un message de redémarrage.

* **void DrawSoundButtons(GameManager gameManager)**

Affiche les icônes de son et permet de couper/activer la musique ou les effets sonores.

* **void TextureLoader(GameManager gameManager)**

Charge dynamiquement les textures et modèles selon l’état courant du jeu.

* **void UnloadAll()**

Décharge toutes les textures et modèles pour libérer la mémoire utilisée.

Particularités :

* Affiche un modèle 3D animé dans le menu avec Raylib.
* Grilles de portraits interactives avec survol fluide (hover offset).
* Saisie clavier personnalisée dans l’écran de génération.
* Interface adaptative selon le mode simpsle ou double écran.
* Chargement conditionnel des ressources pour limiter l’usage mémoire.

### SoundManager

Rôle :  
Gère l’ambiance sonore du jeu. S’occupe de la lecture des musiques selon l’état du jeu, ainsi que du volume et du chargement des effets sonores.

Propriétés :

* Music backgroundMusic : Flux de musique en cours.
* Sound flickSound : Effet sonore joué lors d’une action de sélection (ex. : clic sur un portrait).
* Sound restartSound : Effet sonore joué lors d’un redémarrage de partie.
* bool isMusicPlaying : Indique si une musique est actuellement en lecture.
* int MusicVolume : Volume de la musique (de 0 à 100).
* int SfxVolume : Volume des effets sonores (de 0 à 100).
* string currentMusicPath : Chemin du fichier audio actuellement chargé.
* GameState previousState : État précédent du jeu, utilisé pour éviter les rechargements inutiles.

Méthodes :

* **void SoundsLoader(GameManager gamemanager)**

Charge et lance la musique appropriée selon l’état du jeu. Charge également les effets sonores de base (clic, redémarrage).

* **void UpdateMusic()**

Met à jour le flux musical (appelé chaque frame). Ajuste le volume si besoin.

* **void StopMusic()**

Arrête et décharge proprement la musique en cours.

* **void UpdateSFX()**

Ajuste le volume des effets sonores en fonction de la propriété SfxVolume.

* **void StopSFX()**

Arrête les effets sonores si ceux-ci sont en cours de lecture.

## Les interfaces

Comme mentionné plus haut, le GameManager redirige le joueur vers plusieurs états, les états qui ont été finalisé sont les suivants :

* **Menu**
* **Rules**
* **Generation**
* **Options**
* **Creating**
* **InGame**

Chacun de ces états affiche donc une page différente à travers le UIManager.   
Afin de garantir une expérience utilisateur (UX) optimale il a fallu réfléchir à la disposition de tous les éléments visuels. Voici comment ils se présentent :

### Menu

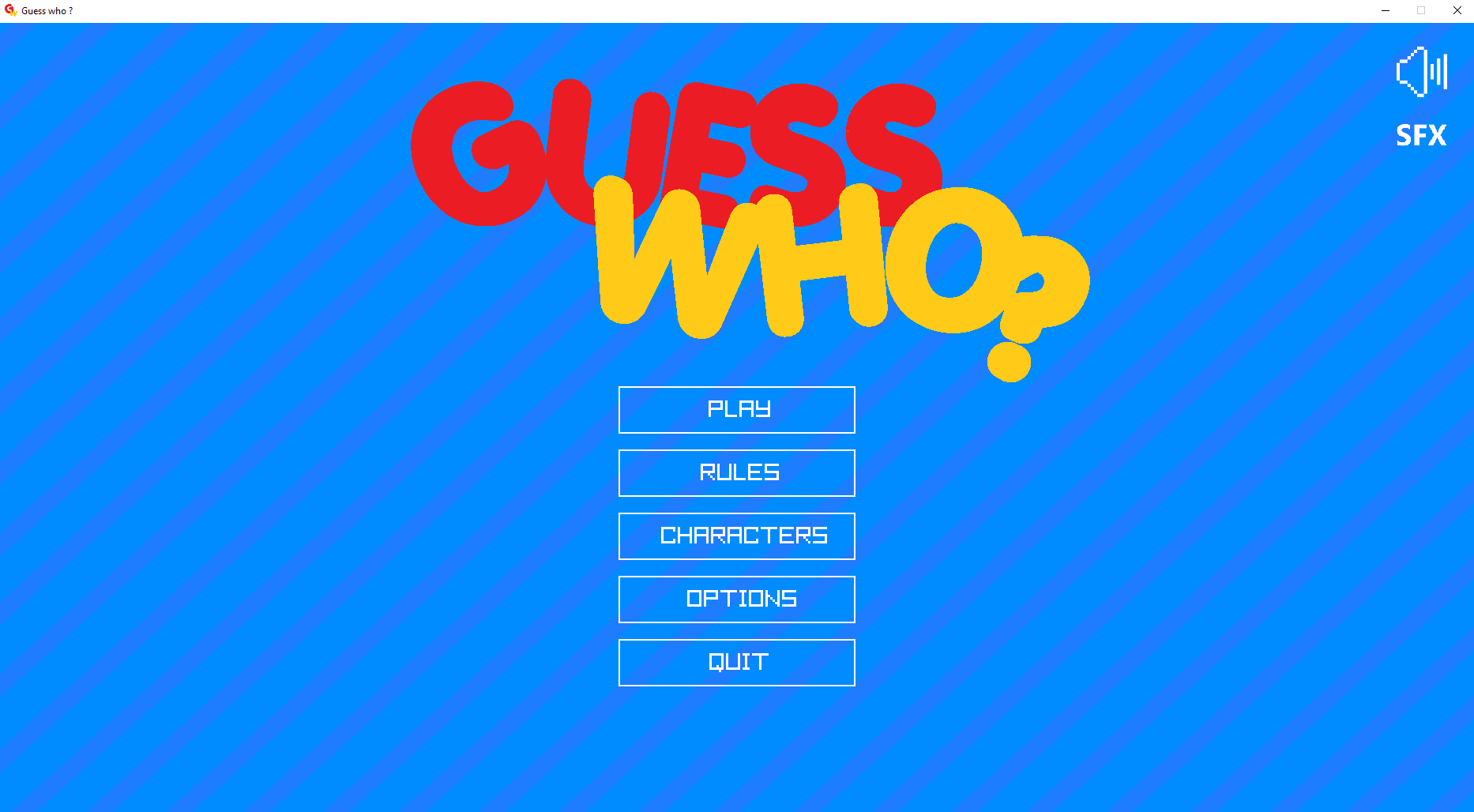


Figure 11 : Menu

Au démarrage l’application se présente ainsi, mettant à disposition des boutons pour accéder aux différentes fonctionnalités qui précèdent le jeu et deux boutons pour arrêter la musique ou les effets sonores.

* **Play** : démarre une partie.   
  (Passe à l’état **InGame** du GameManager)
* **Rules** : ouvre une fenêtre explicative des règles de « Qui est-ce ? ».   
  (Passe à l’état **Rules** du GameManager)
* **Characters** : ouvre une fenêtre pour changer le nombre d’attributs similaires maximum et pour créer vos propres portraits.  
  (Passe à l’état **Creating** du GameManager)
* **Options**: ouvre une fenêtre pour changer les paramètres d’effets sonores, de musique et de configuration d’écran.  
  (Passe à l’état **Options** du GameManager)
* **Quit**: ferme l’application.

### Rules

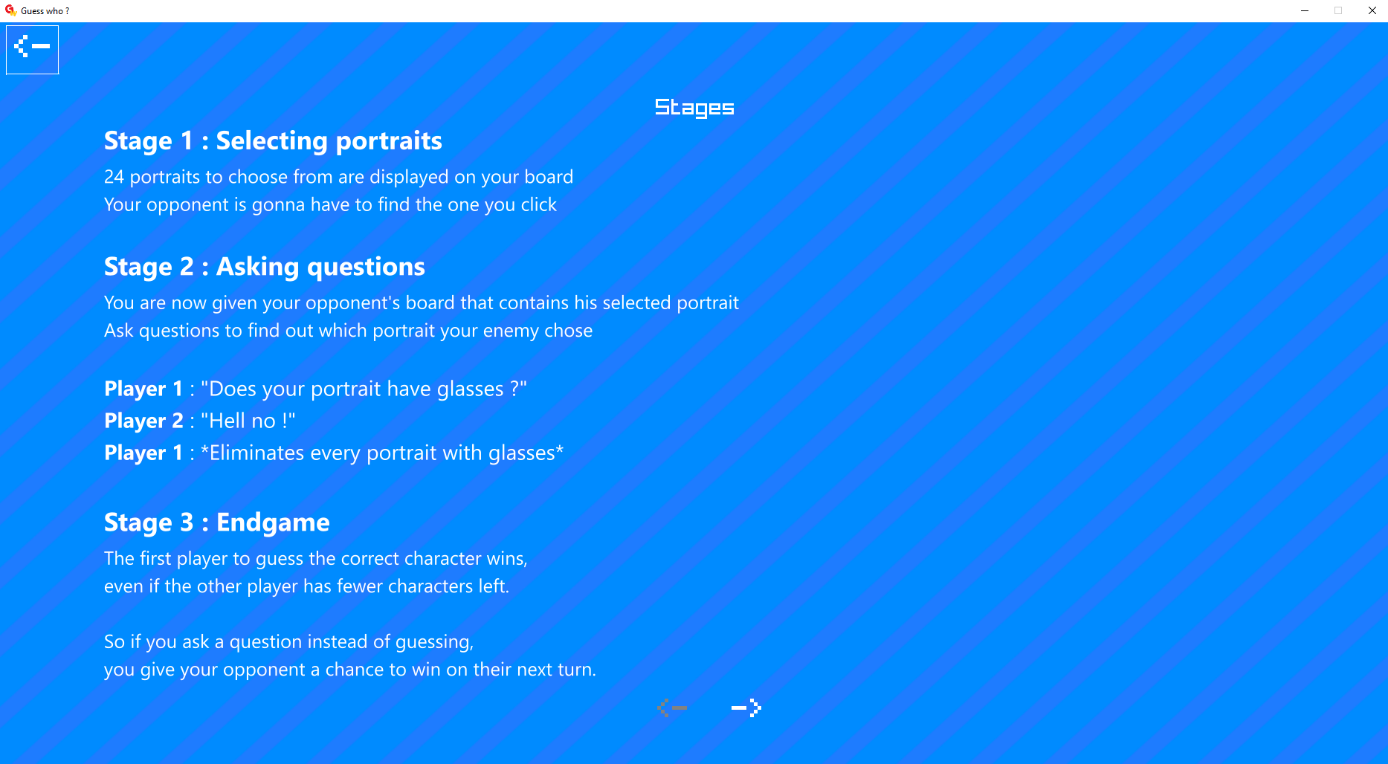
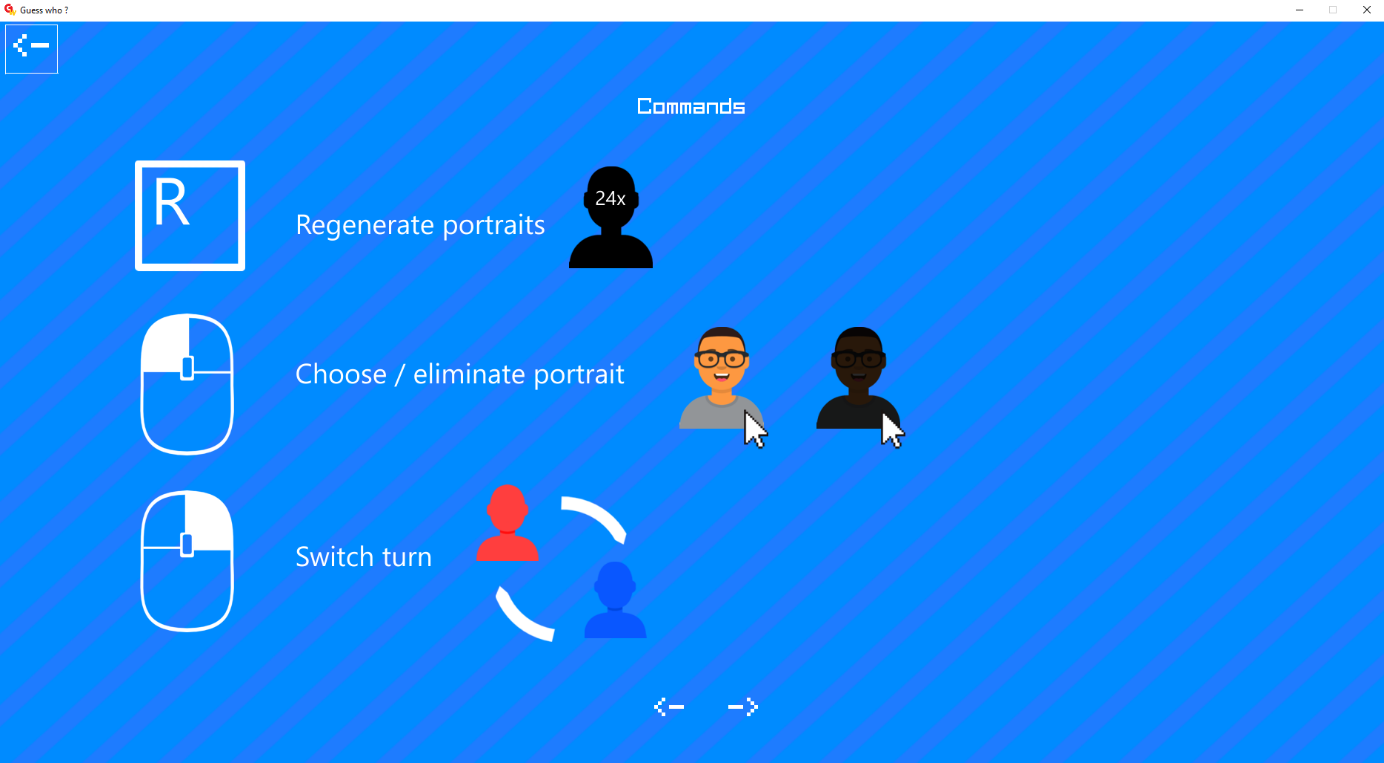
Grâce à des entiers appelés « page » et « lastPage » je permets au joueur de naviguer entre plusieurs pages en cliquant sur deux flèches au bas de la page. Comme indiqué dans le nom de cet état, ces pages relatent des règles principales du jeu et des commandes pour interagir avec le plateau.

Figure 12 : Règles, page 1

Figure 13 : Règles, page 2

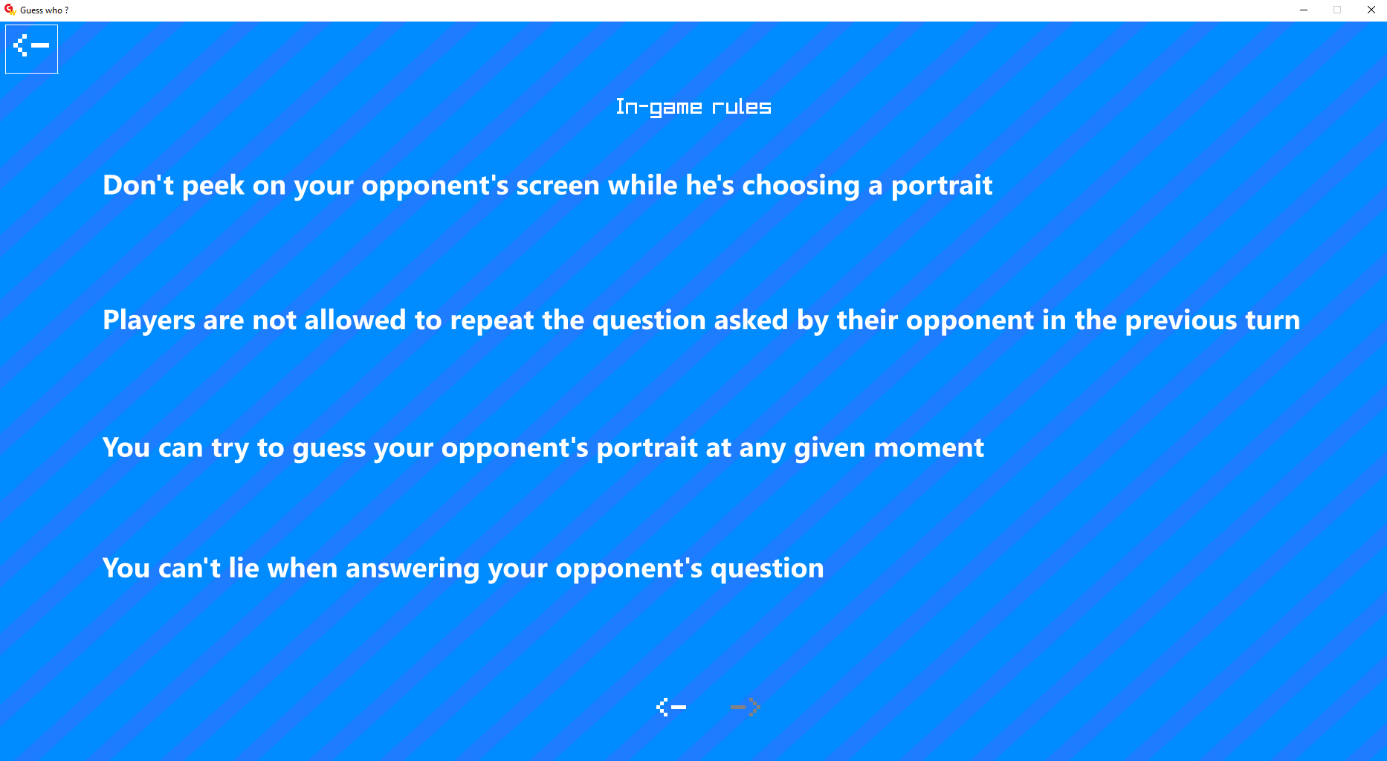


Figure 14 : Règles, page 3

### Characters

Malheureusement la page de création de portrait n’est pas encore implémentée puisqu’elle ne faisait pas partie des objectifs principaux. Cette page afficherait tous les portraits déjà créés par l’utilisateur et permettrait à l’utilisateur de naviguer à travers les portraits horizontalement à l’aide de flèches.

Voici une image représentative de la page qui était prévue :

Figure 15 : Page des portraits actuelles



Figure 16 : Page des portraits exemple

Lorsqu’on clique sur « Génération » en bas à droite, la fenêtre suivante apparaît :

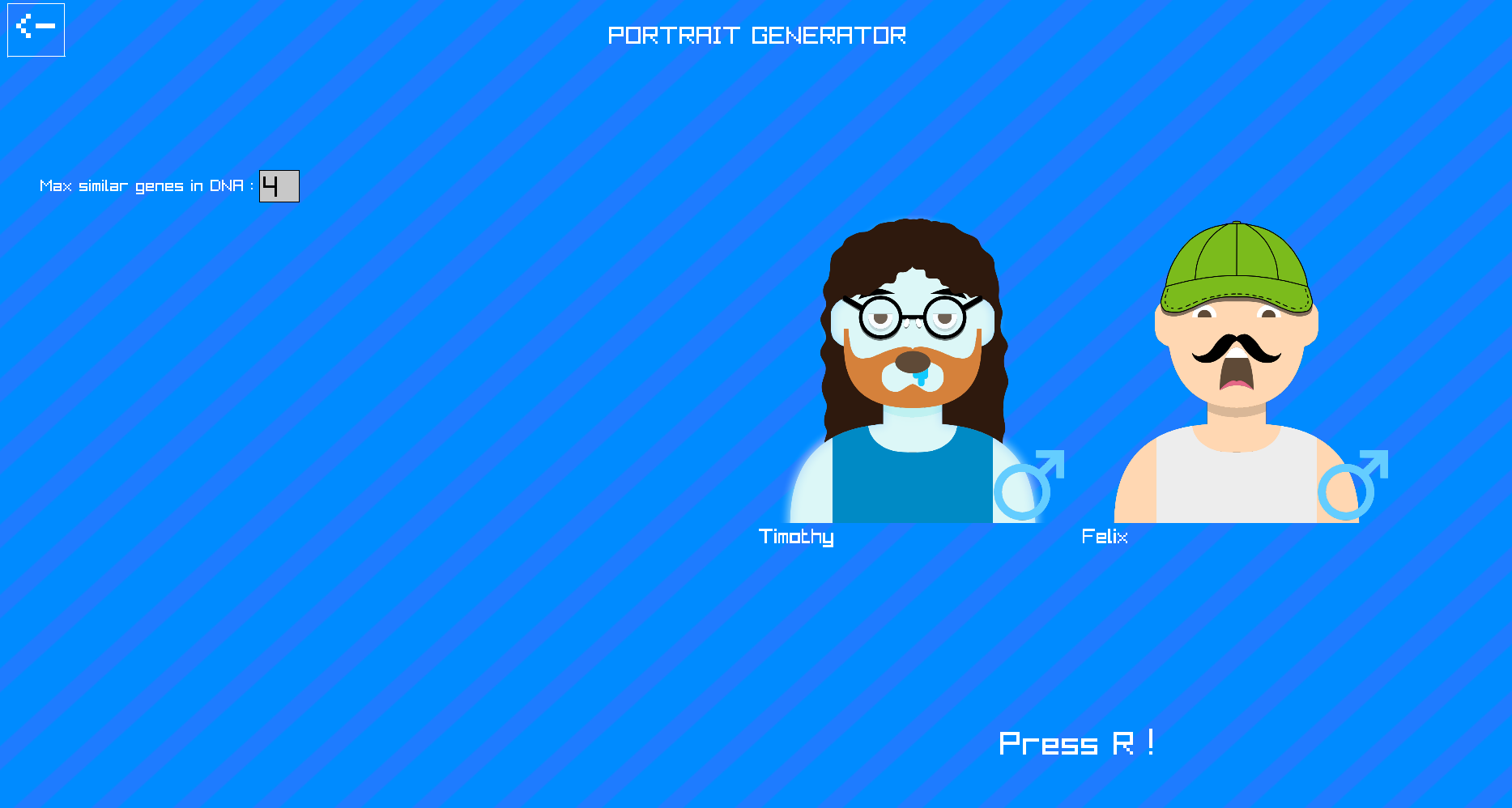


Figure 17 : Page de génération de personnage

Cette page présente comment les portraits sont aléatoirement généré en affichant deux portraits d’exemple qui changent à chaque fois que la touche **« R »** est pressée.

Lorsque l’utilisateur modifie le champ en haut à gauche il change la variable « **maxSimilarAttributes** ». (Mentionnée dans la méthode « **IsSimilarTo** » de la classe **Portrait**)

Donc plus **maxSimilarAttributes** est grand plus il y a de chance que deux personnages se ressemblent.   
Puisque le nombre de textures est insuffisant, il n’y a pas la possibilité de générer des personnages avec une valeur inférieur à 2 dans **maxSimilarAttributes**. La valeur qui est définie ici est réutilisée lorsque on lance une partie.

### Options

Lorsque l’utilisateur souhaite définir ses préférences en cliquant sur « Options » voici ce qui apparaît :

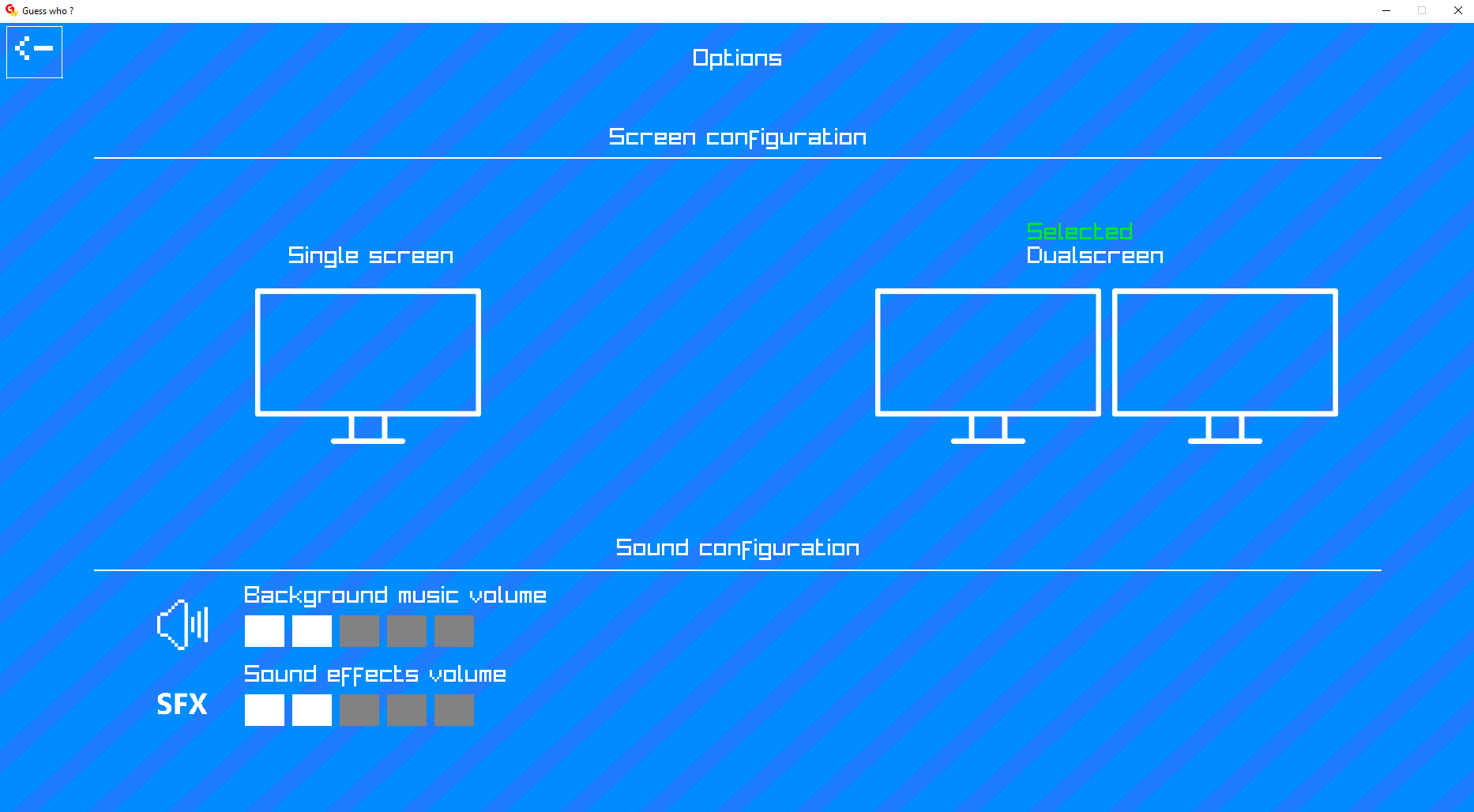


Figure 18 : Page des options

Dans la section « **Screen configuration** », il est possible de modifier la configuration de l’affichage afin que le jeu apparaisse soit sur un seul écran, soit sur les deux simultanément.

Dans la section « **Sound configuration** », il est possible de régler indépendamment le volume de la musique de fond et des effets sonores. Ces deux sources audios fonctionnent de manière autonome : couper l’une n’impacte pas l’autre. Il est également possible de les rendre totalement muets.

### Play

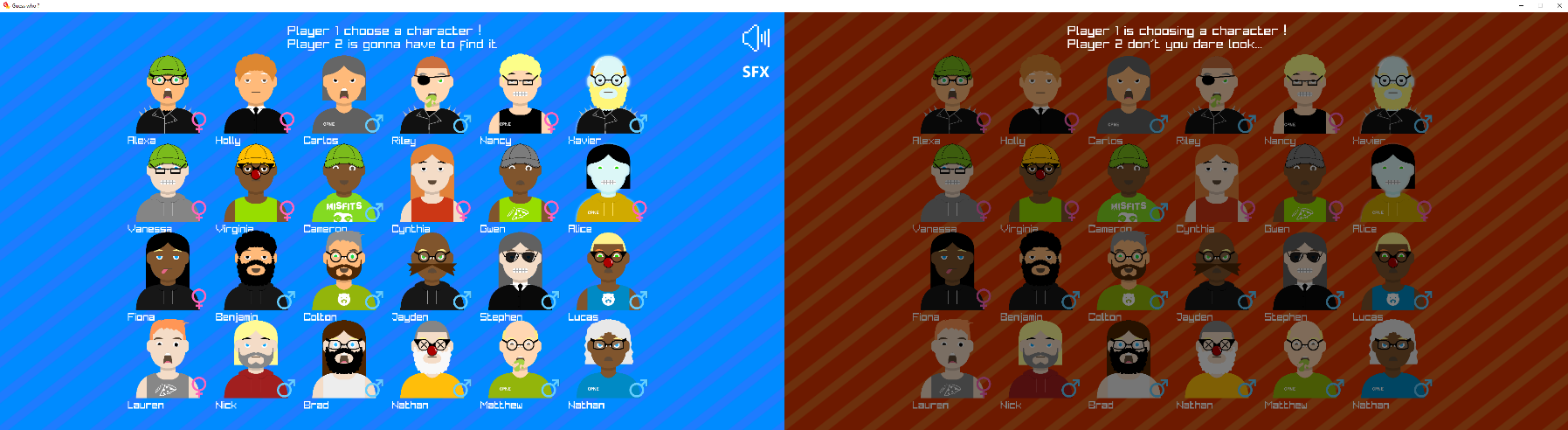
Enfin, une fois le bouton « Play » pressé voici comment l’interface du jeu s’affiche :

Figure 19 : Screenshot de la sélection des portraits

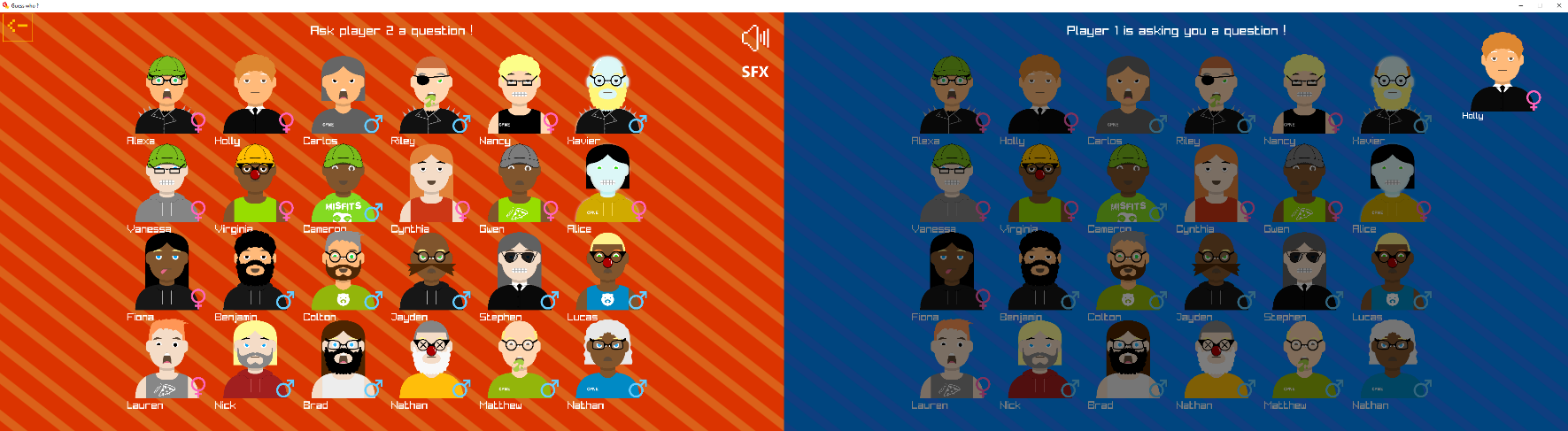
Tant que le joueur 1 n’a pas choisi le portrait à faire deviner, le plateau du joueur 2 est grisé et non cliquable et vice-versa.   
Une fois que les deux joueurs ont choisi leurs portraits, les plateaux sont échangés :

Figure 20 : Screenshot de début de partie

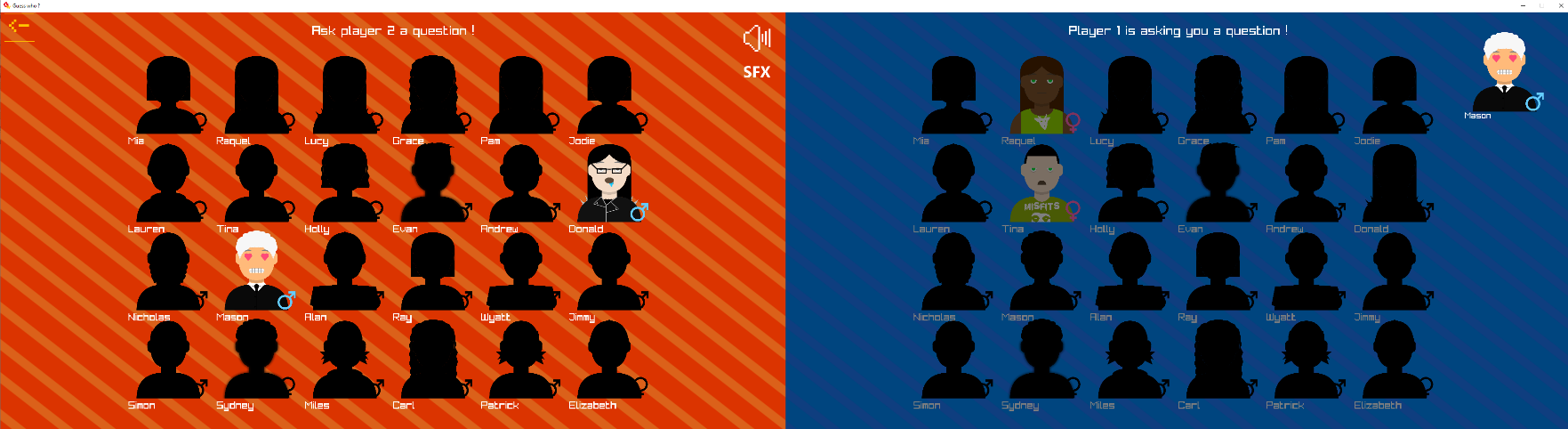
Le joueur chargé de répondre à une question voit s’afficher, en haut à gauche de son écran, le portrait qu’il a préalablement sélectionné. Cela lui permet de ne pas l’oublier, même s’il l’a dissimulé sur le plateau de son adversaire.

Lorsque la partie évolue, voici ce à quoi peuvent ressembler les plateaux :

Figure 21 : Screenshot de milieu de partie

Puis en fin de partie :

Figure 22 : Screenshot de fin de partie



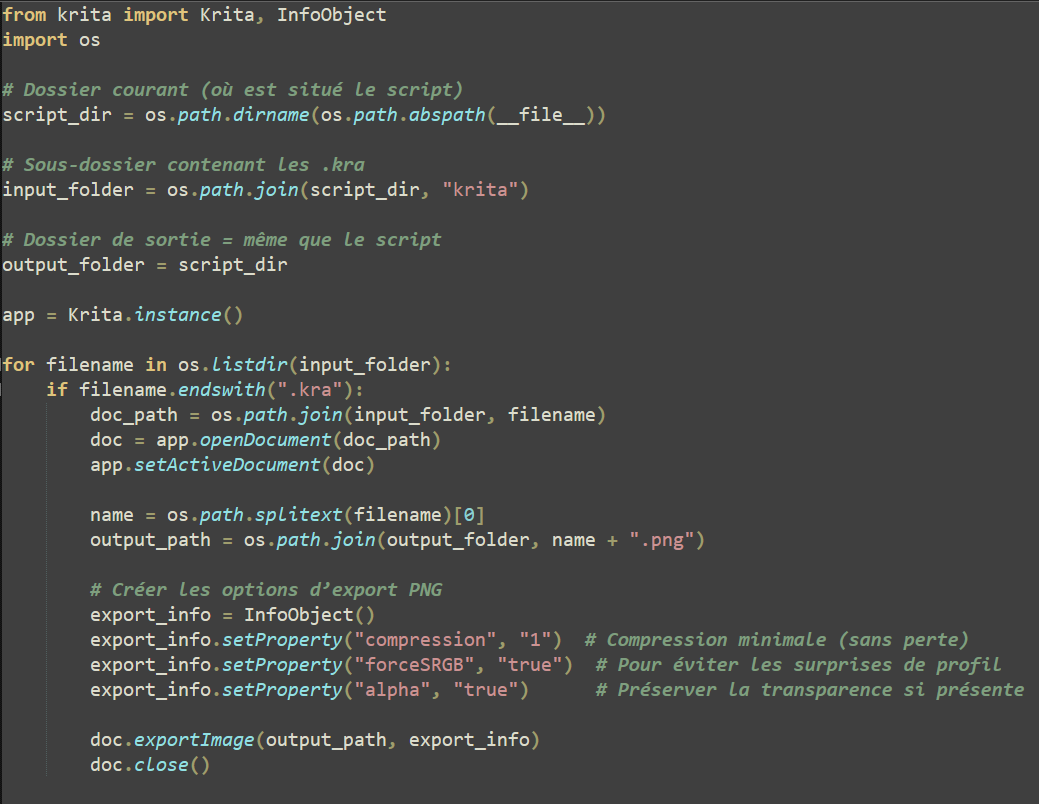
## Les textures

Bien que cette étape ne relève pas directement de la programmation, la création de textures a grandement contribué à rendre mon jeu plus attrayant visuellement, plus lisible et surtout plus ludique.

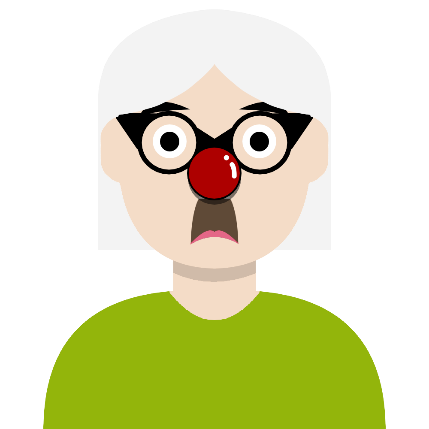
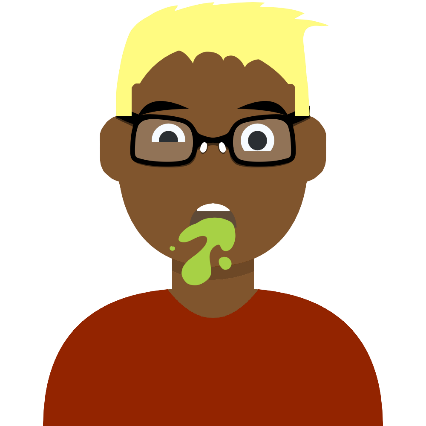
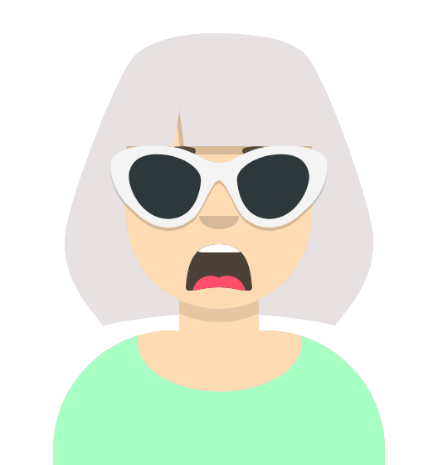
Tous les éléments graphiques présentés dans le chapitre précédent sont le fruit de mon propre travail. Cette phase a requis une petite part du temps imparti, mais l’investissement en valait pleinement la peine. Brièvement, je vais expliquer comment cela s’est déroulé :

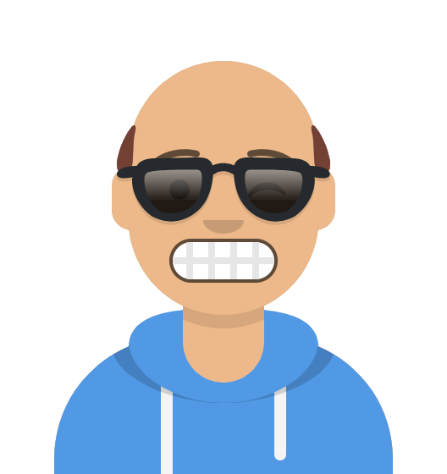
Pour dessiner toutes mes textures, j’ai utilisé **Krita**, une application open-source de dessin numérique. Krita m’a permis de créer rapidement une grande variété de textures à partir d’un seul fichier de base, simplement en jouant sur les propriétés des calques.

En effet, en modifiant uniquement la teinte, la saturation et la valeur (TSV) d’un calque spécifique, je pouvais dériver **plusieurs versions visuellement différentes d’une même texture**, tout en conservant sa structure initiale. Ce procédé m’a fait gagner un temps précieux, tout en assurant une cohérence visuelle dans l’ensemble du jeu.

Afin de simplifier l’exportation de toutes mes textures sans devoir ouvrir manuellement chaque fichier .kra (format natif de Krita), j’ai écrit un petit **script Python** automatisé et exécutable dans Krita :

Ce script : parcourt un dossier contenant tous mes fichiers .kra, les ouvre un à un avec l’API Python de Krita, puis les exporte automatiquement au format .png, tout en appliquant les paramètres nécessaires (compression minimale, profil sRGB, transparence activée).

Pour le design des portraits, je me suis inspiré des avatars du [AvataaarsGenerator](https://getavataaars.com/) fournis par Fang-Pen Lin une développeuse des États-Unis. Je vous invite à tester leur générateur de portrait aléatoire.



Guess Who – Sacha Leone

Avataaars – Fang-Pen Lin

## Les sons et les musiques

## Représentation graphique du chemin du code

# Fonctionnement

# Résultats

C02 - Développer des modèles de données

C03 - Implémenter un modèle de données

C09 - Analyser, identifier et vérifier la validité des données

C10 - Préparer, présenter et évaluer les données

G01 - Documentation des exigences métier et techniques

G08 - Elaboration d'un concept de réalisation

G11 - Elaboration de concepts de tests et définition des cas de tests

G12 - Exécution et évaluation des tests

Projet (screenshot)

# Conclusion

## Problèmes rencontrés

Génération de 48 portraits différents, et non 24 dupliqués

Avaaatars svg

« Toujours copier »

Background

Chargement des textures à chaque fenêtre

## Améliorations possibles

Gestion des fenêtres

Chargement des assets

Écran de gagnant/perdant

## Futur de l’application

3D

4 plateaux

Créateur de portrait

Langues

Changer la couleur des textures

## Avis personnel

# Annexes