Développement d’application

Rioma

**Kylian DEFFAUT**

**Pour le 10/12**

**Elodie DECRAIE**

**Mr. Tomczak**

**Romain DEBRUE**

**L2-INFO**



<https://github.com/Deerman59/TP-Developpement>

# Résumé de notre projet

Notre projet, Rioma, réinvente le célèbre jeu vidéo **Super Mario Bros** en inversant ses éléments clés.

Ce concept permet aux joueurs de redécouvrir l'univers de Mario sous une perspective inversée. Chaque personnage jouable possède des attributs spécifiques, rendant le gameplay plus intéressant.

Le jeu a été développé avec la bibliothèque **Pygame**, qui a facilité la création de l'interface graphique, des mécaniques de jeu, et de l'ambiance sonore.

Ce projet est le fruit de notre passion pour les jeux vidéo et notre volonté de rendre hommage à un classique tout en le revisitant de manière créative.

Table des matières

[Résumé de notre projet 2](#_Toc184595813)

[I. Partie commune 4](#_Toc184595814)

[Présentation de notre projet 4](#_Toc184595815)

[Comment fonctionne-t-il ? 4](#_Toc184595816)

[Point de vue sécurité 5](#_Toc184595817)

[Analyse des Risques de Sécurité 5](#_Toc184595818)

[Conformité et Réglementations 5](#_Toc184595819)

[Définition des Exigences de Sécurité 6](#_Toc184595820)

[Identification des Interactions Externes 6](#_Toc184595821)

[Solutions envisagées 6](#_Toc184595822)

[Répartition des tâches 6](#_Toc184595823)

[Budget et ressources du projet 8](#_Toc184595824)

[Critère d’acceptation 8](#_Toc184595825)

[Risques et plan de gestion des risques 9](#_Toc184595826)

[I. Partie personnelle 9](#_Toc184595827)

[Etudiant 1 - Kylian Deffaut 9](#_Toc184595828)

[Introduction 9](#_Toc184595829)

[Analyse 9](#_Toc184595830)

[Recherche de solutions 10](#_Toc184595831)

[Conception 11](#_Toc184595832)

[Implémentation 12](#_Toc184595833)

[Difficultés Rencontrées 12](#_Toc184595834)

[Tests unitaires et validation 13](#_Toc184595835)

[Etudiant 2 - Elodie Decraie 13](#_Toc184595836)

[Introduction 13](#_Toc184595837)

[Analyse 14](#_Toc184595838)

[Conception 15](#_Toc184595839)

[Implémentation 18](#_Toc184595840)

[Tests unitaires et validation 20](#_Toc184595841)

[Tests d’intégration 20](#_Toc184595842)

[Discussion 21](#_Toc184595843)

[Etudiant 3 - Romain Debrue 21](#_Toc184595844)

[Introduction 21](#_Toc184595845)

[I. Aspect Visuel 21](#_Toc184595846)

[II. Musique et Son 27](#_Toc184595847)

[III. Suite et fin de la partie générale 28](#_Toc184595848)

[Conclusion et perspectives 28](#_Toc184595849)

[Annexes 28](#_Toc184595850)

[Bibliographie / Sitographie / Références 29](#_Toc184595851)

# I. Partie commune

## Présentation de notre projet

Notre projet s’inspire du célèbre jeu vidéo **Super Mario Bros**. Nous avons décidé d’inverser plusieurs éléments emblématiques de ce classique, donnant naissance à un jeu que nous avons intitulé **"Rioma"** (inversion de Mario).

Dans **Rioma**, les rôles sont complètement renversés : les méchants deviennent les héros, et Mario, transformé en **Rioma**, devient l’antagoniste qu’il faut battre.

Les personnages habituellement hostiles dans l’univers de Mario, comme **Bowser**, les **Goombas**, et d’autres, deviennent ici des héros jouables, chacun avec des attributs spécifiques. Notre concept va, nous l’espérons, permettre de vous faire redécouvrir un gameplay inversé et original.

Pour donner vie à ce projet, nous avons utilisé la bibliothèque **Pygame**, qui nous a permis de concevoir l’interface graphique, les mécaniques de jeu, et l’ambiance sonore. Inspirés par notre passion commune pour les jeux vidéo et par l’univers de Mario, nous avons voulu rendre hommage à cette icône tout en apportant notre propre touche créative.

## Comment fonctionne-t-il ?

Pour lancer **Rioma**, vous aurez besoin :

1. D’un **IDE Python** (nous recommandons Thonny pour sa simplicité, mais les autres sont également envisageables).
2. D’avoir installé les paquets **Pygame**, **Sys** et **Random**.

Une fois le dossier compressé téléchargé depuis notre GitHub, ouvrez le programme principal avec l’aide de votre compilateur Python, puis exécutez le programme en appuyant sur **F5**. Le jeu devrait être lancé.

**Fonctionnalités principales**

* **Menu principal :**
  + Bouton « **Jouer** ».
  + Choix du personnage : **Para-Koopa**, **Para-Goomba**, **Bowser**, **Chomp** ou **Boo**.
* **Fonctions adjacentes :**
  + En appuyant sur **M**, le jeu se met en **pause** et affiche un petit menu. Depuis ce petit menu, il est possible de soit **reprendre** **la** **partie**, soit de **modifier** **le** **volume** **de** **la** **musique**.
  + En appuyant sur **Echap**, le jeu se met également sur **pause** et demande si le joueur veut **quitter** **le** **jeu** ou rester.
* **Interface :**
  + La résolution s’adapte automatiquement à l’écran utilisé.
  + Notre jeu peut donc convenir à plusieurs types d’écrans tels que ceux d’ordinateurs portable, de télévisions, etc. -> Jeu multiplateforme.
* **Compatibilité matérielle :**
  + Notre jeu ne nécessite pas forcément une machine performante.
  + Il fonctionne aussi bien sur Windows, Mac, et Linux (tant que les bibliothèques sont installées sur la machine).

**Points techniques et accessibilité**

* **Plateformes supportées :** Windows, Mac, et Linux.
* **Logiciels conseillés :** Thonny, Spyder, ou tout IDE **Python** avec **Pygame**, **Sys** et **Random**.
* **Guide d’installation :** Pour installer Pygame, suivez les instructions officielles sur leur site : [Pygame - Getting Started](https://www.pygame.org/wiki/GettingStarted).

## Point de vue sécurité

### Analyse des Risques de Sécurité

D’un point de vue sécuritaire, notre projet ne présente pas de failles majeures. Ce n’est ni un jeu multijoueur, ni un jeu nécessitant une connexion à Internet, à l’exception de l’installation des modules requis et du téléchargement du jeu depuis notre page GitHub.  
De plus, tous les modules utilisés sont natifs de Python ou issus de bibliothèques largement reconnues et vérifiées, ce qui réduit considérablement les risques associés à d’autres dépendances inconnues.

### Conformité et Réglementations

Concernant le RGPD (Règlement Général sur la Protection des Données), notre projet récupère uniquement les dimensions de l’écran du joueur, pour adapter au mieux sa résolution, mais en aucun cas nous traitons ces données à l’extérieur du programme. Le jeu peut donc être considéré comme conforme et sans risque sur cet aspect.

### Définition des Exigences de Sécurité

Pour garantir le bon fonctionnement de l'application, nous recommandons de ne pas modifier le code source sans connaissance approfondie du projet. Cela nous permet non seulement de préserver l'intégrité du jeu, mais aussi de respecter le travail réalisé par l’équipe.

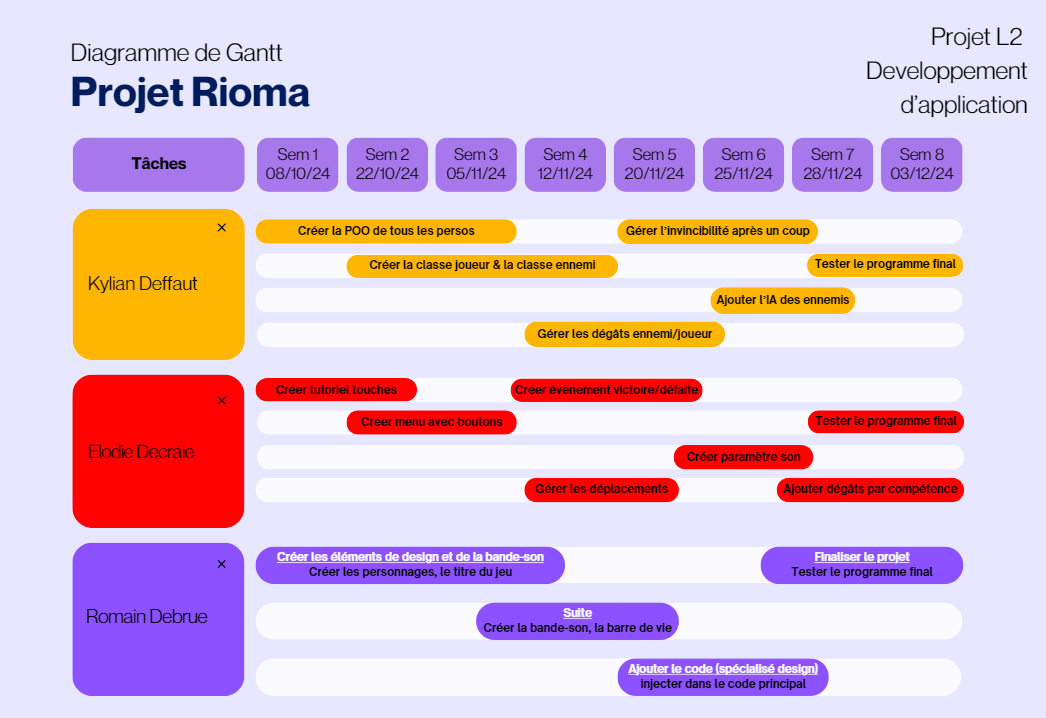
### Identification des Interactions Externes

Le jeu n’interagit avec aucun service tiers ou API externe. Les seules interactions externes possibles concernent le téléchargement des dépendances Python nécessaires à l’exécution de notre jeu vidéo.

### Solutions envisagées

Pour éviter tout problème de sécurité, nous avons veillé à utiliser des bibliothèques fiables. C’est pourquoi la mise à disposition du code sur GitHub est accompagnée de recommandations et d’explications d’installation pour éviter des erreurs non souhaitées.

## Répartition des tâches



Nous avons tous ensemble analysé les tâches à faire pour l’accomplissement du projet.

Voici, plus ou moins, en combien de temps nous avons réalisé certaines tâches :

**Kylian Deffaut :**

* **Utiliser la Programmation Orientée Objet (P.O.O) pour :**
  + La classe Joueur et les classes des ennemis (en une séance de 3 heures).
* **Gérer les collisions :**
  + Implémentation des collisions entre les joueurs et les ennemis, avec attribution des dégâts (3 heures).
* **Implémenter l'invincibilité :**
  + Ajout de l’état d’invincibilité des personnages après un dégât, accompagné d’un clignotement visuel (environ 8 heures, réparties sur 2 séances en plus du travail personnel).
* **Créer et gérer les Toads :**
  + Développement des ennemis supplémentaires du jeu, les Toads, en deux séances (environ 6 heures).
* **Tester le programme :**
  + Validation du bon fonctionnement global du jeu via divers tests (quelques minutes).

**Elodie Decraie :**

* **Créer le menu principal :**
  + Réalisation du menu principal, avec ajout des boutons nécessaires pour naviguer entre les différentes sections (environ 2h30 à 3h).
* **Créer le menu de gestion du son :**
  + Développement du menu sonore, comprenant l’ajout d’une classe dédiée à la gestion des sons ainsi que la modification du volume (environ 2h30 à 3h).
* **Créer le menu de sélection des personnages :**
  + Conception du menu permettant de choisir les personnages jouables, une tâche qui a pris environ 2h30 à 3h, voire plus en raison de difficultés rencontrées.
* **Créer le menu de pause :**
  + Implémentation d’un menu de pause avec des options pour reprendre le jeu, ajuster les paramètres ou quitter la partie (environ 30 minutes à 1h).
* **Créer le menu pour quitter :**
  + Ajout d’un menu dédié à quitter le jeu, une tâche qui a pris environ 30 minutes à 1h, avec une recherche sur l’utilisation de la bibliothèque sys pour réaliser cette fonctionnalité.
* **Créer le tutoriel :**
  + Élaboration d’un tutoriel affiché au centre de l’écran, ce qui a nécessité environ 2h30 en raison des ajustements pour centrer l’affichage correctement.
* **Gérer la présentation du GitHub et rédiger le README :**
  + Organisation et structuration de la page GitHub du projet, avec rédaction d’un README clair et détaillé, une tâche réalisée en environ 1h30.

**Romain Debrue :**

* **Créer les images des personnages :**
  + La création des images, en raison de la minutie nécessaire pour les détails, a été l’une des tâches les plus chronophages. Par exemple, la conception de Bowser a pris plus de 2h30, tandis que Boo a nécessité environ 1h30.
* **Rechercher et modifier les fichiers sonores :**
  + La recherche et la modification des effets sonores ont demandé environ 10 heures de travail.
* **Créer et implémenter les écrans de fin :**
  + Après avoir terminé les images, j’ai conçu les écrans de victoire et défaite, puis les ai intégrés au programme en fonction des résultats. Cette étape m’a pris un peu plus de 2 heures.
* **Collaborer avec l’équipe pour l’intégration des assets :**
  + Pendant que je réalisais les images, Kylian et Elodie importaient mes créations dans le programme, ce qui a permis d’avancer efficacement en parallèle pour dessiner les autres images.

## Budget et ressources du projet

Le projet en lui-même n’a pas nécessité l’achat de quoi que ce soit, étant donné des ressources gratuites utilisées et mises à disposition partout sur Internet.

Les seuls coûts à prévoir seraient :

* Un ordinateur utilisable (portable ou fixe)
* Un écran (une télévision, un moniteur, …)
* Un clavier et une souris (ou tout autre pointeur comme un pavé tactile)
* Une connexion à Internet uniquement lors de l'installation du logiciel et des bibliothèques.

Une fois cela effectué, notre programme sera tout de même exécutable sans connexion à Internet après le quatrième point effectué.

## Critère d’acceptation

Pour que notre projet soit considéré comme terminé, nous nous sommes tous fixé de faire un maximum de tâches avant une date limite pour ne pas prendre trop de retard sur d’autres travaux à effectuer. Si une tâche était trop compliquée ou longue à faire (et facultative surtout), on la considérait tout simplement comme bonus. Si notre projet fonctionne sans problème et comporte plusieurs fonctionnalités, alors un premier jet de ce projet peut voir le jour. Après, il est tout à fait possible de l’améliorer par la suite en tant que projet personnel.

## Risques et plan de gestion des risques

De notre point de vue, un programme aussi simpliste ne devrait pas causer de risques potentiels. Le seul gros risque à s’imaginer pourrait être que quelqu’un pirate l’un de nos 3 comptes GitHub et remplace notre programme principal par un virus ou modifie le code source de manière à récupérer des données ou agit tout simplement dans un but malveillant. Cependant, nous pensons très clairement qu’une telle situation ne pourrait se produire car notre jeu vidéo n’est pas la principale cible de personnes mal intentionnées, n’étant pas connu du grand-public.

Si ce cas venait tout de même à se produire, il faudrait simplement protéger nos comptes avec certaines sécurités (comme la double authentification par exemple) pour éviter tout incident.

# I. Partie personnelle

## Etudiant 1 - Kylian Deffaut

### Introduction

J’ai dû gérer la programmation des classes des ennemis et des personnages, dont la création des attributs respectifs de chaque personnage, la gestion de certaines mécaniques comme les collisions, ainsi que l’ajout de la barre de vie inversée.

#### Positionnement

Ce projet est un jeu vidéo rétro en 2D, inspiré du célèbre jeu Super Mario Bros, mais avec une inversion des rôles des personnages gentils et méchants. Il a été développé en Python à l’aide du module Pygame, pour faciliter le développement du jeu, ayant des fonctions intégrées à la bibliothèque qui sont indispensables.

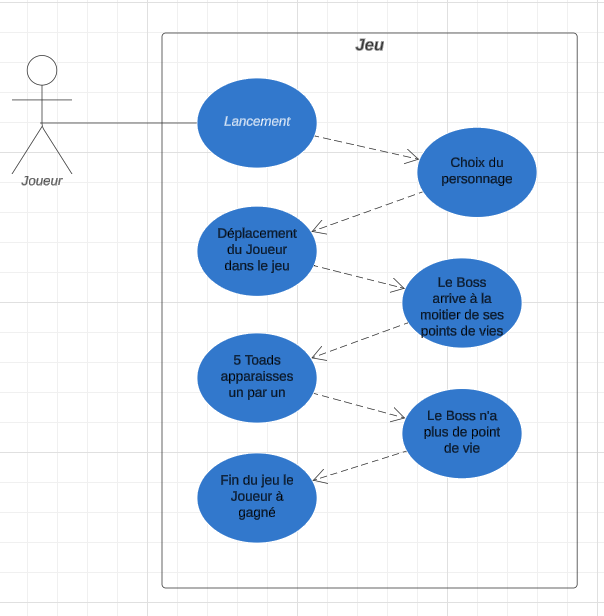
### Analyse

#### Identification des Scénarios et Cas d'Utilisation

**Scénario :**

Le joueur démarre le jeu, choisit un personnage dans le menu de sélection, puis affronte des ennemis en leur sautant dessus tout en évitant les attaques du boss principal, Rioma.

Lorsque le boss arrive au milieu de sa vie, des **Toads** apparaissent pour augmenter la difficulté. Si le joueur réussit à réduire les points de vie de Rioma jusqu’à zéro, il gagne, ce qui déclenche l’écran de victoire avec une musique qui se déclenche. Sinon, l’écran de défaite s’affiche avec un autre bruitage.



### Recherche de solutions

J’ai rencontré des difficultés lors de la création de l’invincibilité et le clignotement des personnages après un dégât. L’une des solutions était de le faire avec la fonction time (pygame.time.get\_ticks()), car celle-ci permet d’enregistrer le temps du jeu à un moment donné et de résoudre le problème en prenant le temps actuel et le moment où le personnage perd un point de vie tout en prenant une valeur fixe du délai qu’on veut pour l’invincibilité, en l’occurrence 3 secondes, et de faire :

Si le temps\_actuel - dernier\_temps\_perte\_vie est supérieur ou égal au délai de collision, alors les dégâts sont reçus.

De même avec le clignotement des personnages.

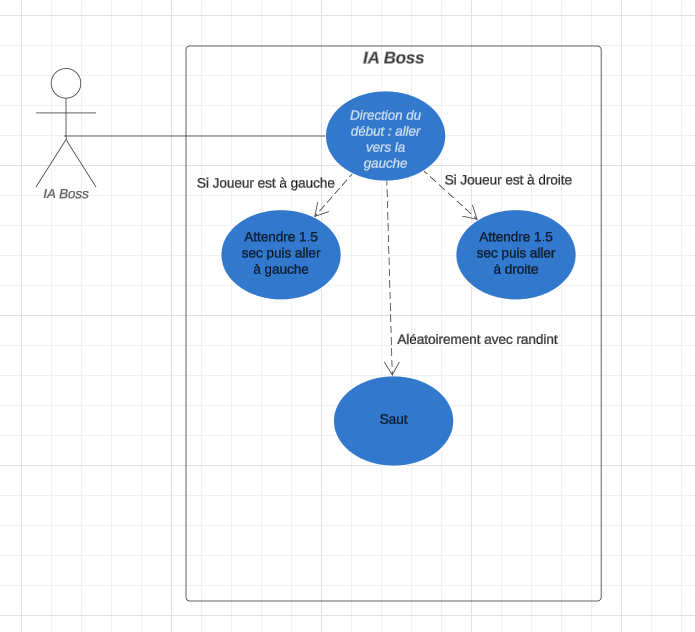
Nous avons choisi comme option technologique Python avec le module Pygame pour avoir un rendu graphique et une simplicité dans la programmation.

J’ai personnellement opté pour l’utilisation de la P.O.O pour faciliter les changements d’attributs, d’images et de sons des différents personnages avec la méthode perso(), se situant dans la classe Joueur.

### Conception

#### Conception Générale

Diagramme UML de l’IA du Boss :



Le boss change de direction lorsque le joueur passe derrière lui. Cependant, pour éviter qu’il « colle » constamment le joueur, j’ai introduit un délai de 1,5 secondes avant qu’il ne change de direction. De plus, le boss dispose d’une probabilité de sauter, fixée à 1/400 par itération dans la boucle while, pour éviter des comportements trop répétitifs ou un gameplay déséquilibré.

##### Interface Utilisateur

Le joueur arrive sur un menu avec un bouton « Jouer » puis arrive sur la page de sélection des personnages où il doit en choisir un pour pouvoir jouer. Après qu’il l’ait choisi, le jeu commence et il se retrouve face à Rioma qu’il doit affronter. Après qu’il ait battu le boss, l’écran de victoire s’affiche avec une musique victorieuse en arrière-plan. À l’inverse, s’il s’est fait battre, l’écran de défaite s’affiche avec également un bruitage en arrière-plan.

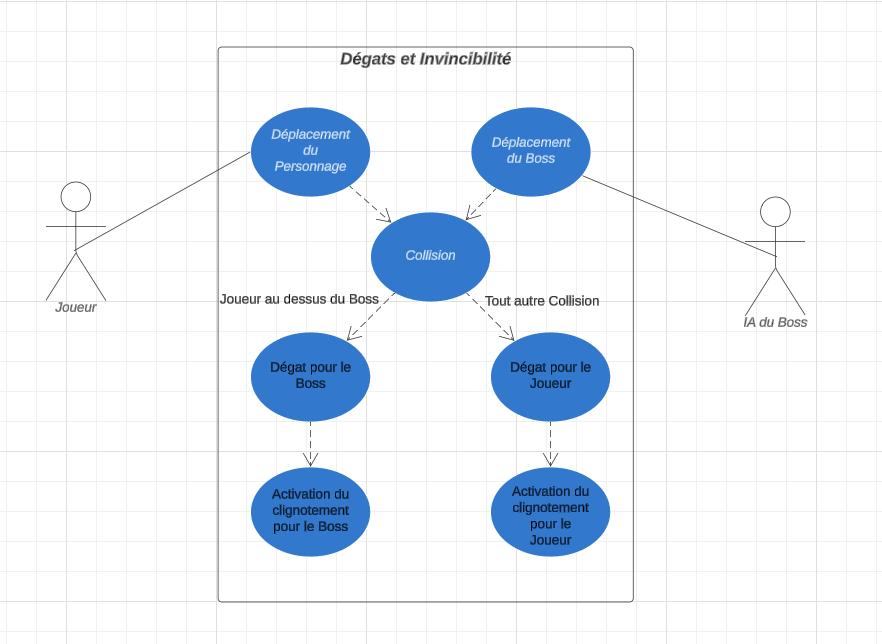
#### Conception Détaillée

**Classes créées :**

* **Classe Joueur :**
  + Attributs : choix, vie, nom, taille, speed, jump, song\_jump, imgD, imgG, imgDH, imgGH, gravity.
  + Méthode principale : perso() pour gérer les statistiques, images et son de saut du personnage sélectionné.
* **Classes Boss et Toad (les ennemis) :**
  + Attributs similaires, avec un attribut additionnel : direction pour gérer leurs déplacements.

L’utilisation de la Programmation Orientée Objet m’a aidé à faciliter les changements de statistiques, d’images et de sons des différents personnages avec la méthode perso(), se trouvant dans la classe Joueur.

Diagramme UML des dégâts et de l’invincibilité entre le joueur et l’IA du boss :



### Implémentation

L’implémentation des images et des sons sont également réalisés grâce au module Pygame et sont gérés avec la P.O.O. Si ce personnage est sélectionné dans la méthode perso() de la classe Joueur, ce sera **ce** son et **cette** image pour **ce** personnage. Il y a aussi les fonds, l’écran de victoire et de défaite qui sont implémentés grâce à cette même bibliothèque.

### Difficultés Rencontrées

Une des difficultés que j’ai rencontrées était celle de l’orientation des images du joueur et du boss suite aux déplacements. Par exemple, lorsque le joueur allait à droite, il fallait charger l’image du joueur allant à droite et si le joueur saute et en même temps se dirige vers la gauche, il fallait charger l’image correspondante, mais cette situation se complique s’il retombe au sol car il faut de nouveau recharger l’image du joueur sans l’animation de saut, cette fois-ci, et avec l’orientation correspondante. Pour résoudre cela, il a fallu faire une série de conditions à l’aide de la variable « is\_jumping », qui permet de voir si le joueur est entrain de sauter ou non, mais également vérifier les états des time.ticks qui vont vérifier si la touche gauche et droite sont pressées en même temps pour que cela ne charge pas les deux images en même temps mais bien qu’une seule.

Une autre difficulté notable était celle pour gérer les clignotements des personnages pour qu’ils deviennent invincibles, comme expliqué précédemment.

De même avec les Toads et leurs apparitions, au début je souhaitais qu’ils apparaissent lentement devant la porte puis qu’ils sautent, mais étant donné la complexité de cette manœuvre, j’ai, à la place, juste fait en sorte que dès leur apparition, ils sautent puis avancent tout droit jusqu’à toucher une paroi car, de plus, lorsqu’ils arrivent ils doivent « sortir du mur ». Il a donc fallu que je désactive les collisions pour cette condition puis que je les réactive lorsque le Toad est tombé.

### Tests unitaires et validation

#### Tests unitaires

* **Tests de collisions** : Vérification que la détection de collision entre le joueur et les ennemis fonctionne correctement. → Vérifié
* **Tests de l’IA et de la difficulté** : Vérification que la difficulté (incluse dans l’IA de l’ennemi) ne soit pas trop dure ni trop facile. Pour la rendre plus aléatoire, il était préférable d’utiliser la bibliothèque « random » qui nous a permis de générer des nombres aléatoires pour que l’adversaire puisse sauter à un intervalle plutôt irrégulier, ce qui laisserait penser qu’on joue contre un vrai joueur. De même pour le fait qu’il traque le joueur, il était préférable de le faire toutes les 1,5 secondes pour ne pas que cela soit trop dur si ce temps est moins élevé, ni trop facile si plus élevé. → Vérifié

#### Validation

* Tests du gameplay global : Validation que le jeu fonctionne comme prévu (choix du personnage, démarrage et fin de partie).

## Etudiant 2 - Elodie Decraie

### Introduction

Mon rôle dans ce projet consistait à gérer l’interface utilisateur (IU en français ou UI en anglais).

Cela comprenait :

* La conception et la mise en place de tous les menus du jeu (menu principal, menu pause, menu de sélection des personnages, menu de volume, etc.).
* La création de la classe son, intégrée au menu de gestion du volume.
* Le développement du tutoriel d’introduction pour guider les joueurs au démarrage.
* La rédaction du fichier README.md sur la page GitHub.

#### Positionnement

Mes tâches étaient plutôt orientées sur l’utilisateur, où mon objectif principal était de donner envie de jouer au jeu, que ce soit dans l’expérience de navigation fluide, intuitive et facile d’accès, tout en gardant une présentation professionnelle et ludique du projet sur notre page GitHub.

### Analyse

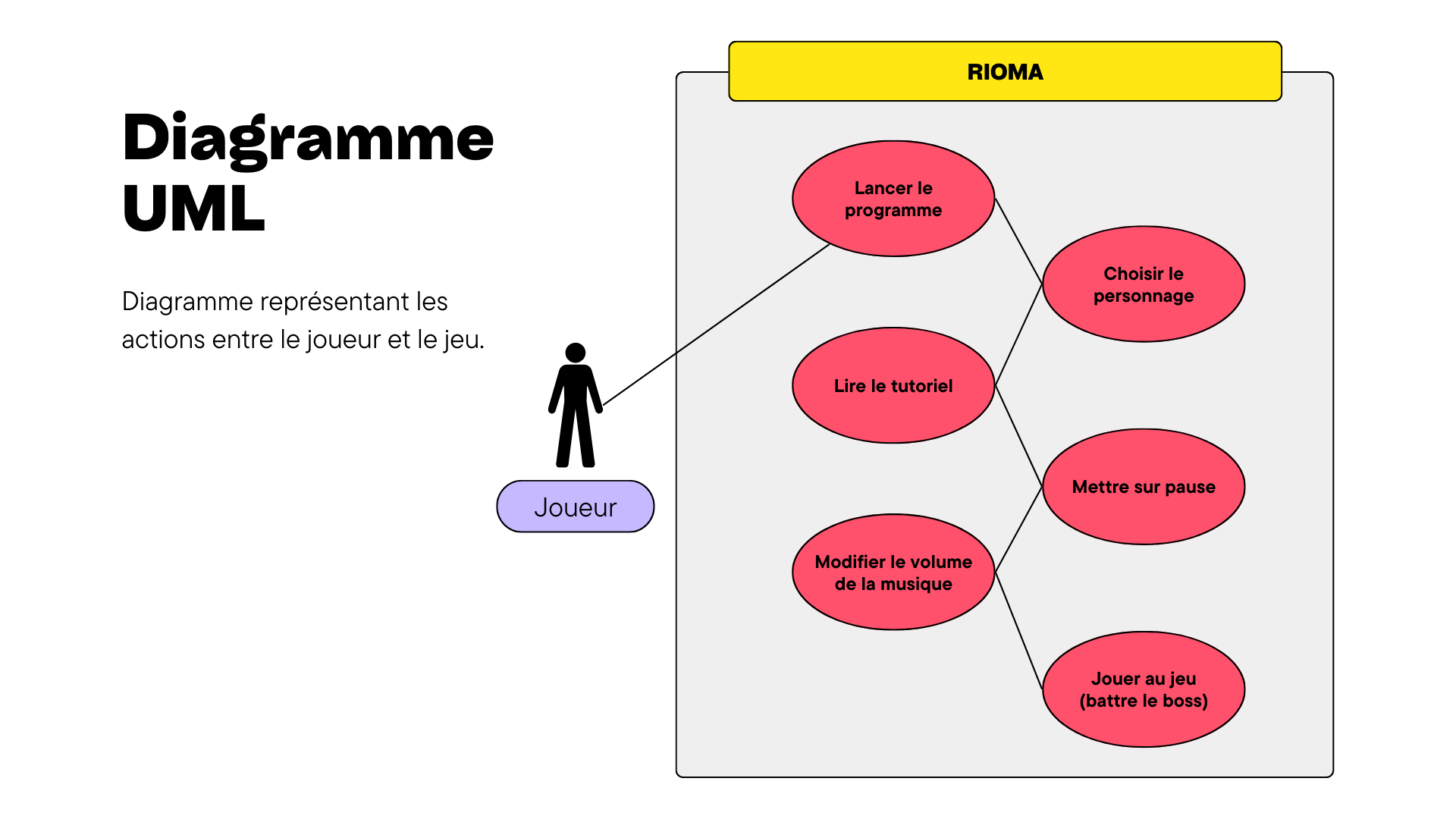
#### Identification des Scénarios et Cas d'Utilisation

**Scénario principal :**

La première chose que l’utilisateur verra lorsqu’il lancera le jeu est bel et bien le menu principal. Pour ne pas perdre l’utilisateur, j’ai opté pour une utilisation simple des boutons avec uniquement le bouton « Jouer » cliquable, une fois cliqué celui-ci amène le joueur vers le choix du personnage jouable. Il aura le libre choix entre 5 figures connues du jeu originel, tels qu’un Para-Goomba, un Para-Koopa, Bowser, Chomp et même Boo (tous conçus par le graphiste de l’équipe). Ensuite, une fois que l’utilisateur aura choisi son personnage, le jeu se lancera, admettons qu’il souhaite suspendre le jeu, un menu pause est mis à disposition en appuyant sur la touche M, il est alors possible pour lui de soit reprendre la partie, soit de changer le volume de la musique du jeu en rappuyant sur M de nouveau. Une fois le volume choisi avec les flèches haut et bas, il pourra revenir sur le menu précédent et ainsi reprendre sa partie là où elle s’était arrêtée. Imaginons que le joueur souhaite quitter la partie, il est possible de le faire en appuyant sur Echap puis « O » pour confirmer celle-ci.

#### Diagramme UML

Voici un aperçu des interactions possibles entre le joueur et notre jeu vidéo :



#### Recherche de Solutions

##### Technologies envisagées

Python avec Pygame a été retenu comme principal outil de développement. Ce choix repose sur sa simplicité d’utilisation, sa large compatibilité avec les interfaces utilisateur et ses fonctionnalités adaptées aux jeux vidéo 2D, comme la gestion des évènements (avec les touches par exemple) ou des graphismes (plutôt simplistes, destinés à n’importe quel ordinateur)

##### Alternatives considérées

Des Frameworks comme Unity ou Godot auraient également pu être envisagés. Cependant, n'ayant jamais travaillé sur ce type de plateforme, leur niveau de complexité aurait « amputé » sur la qualité du rendu final. De plus, les recommandations reçues nous ont conduits à privilégier Pygame, qui est donc restée une solution plus accessible pour toute l'équipe. Que ce soit pour la partie graphisme (assez simpliste qui correspond au style de notre graphiste), pour le codage assez familier en Python et toutes les bibliothèques mises à disposition gratuitement pour réaliser certaines tâches.

### Conception

Je vais uniquement aborder la conception technique et visuelle, étant donné que le rôle de la conception du côté esthétique et de l’interprétation de ces images était celui de mon collègue Romain Debrue.

#### Conception générale

Je vais présenter de manière générale à quoi ressemble les différents menus que j’ai réalisé :

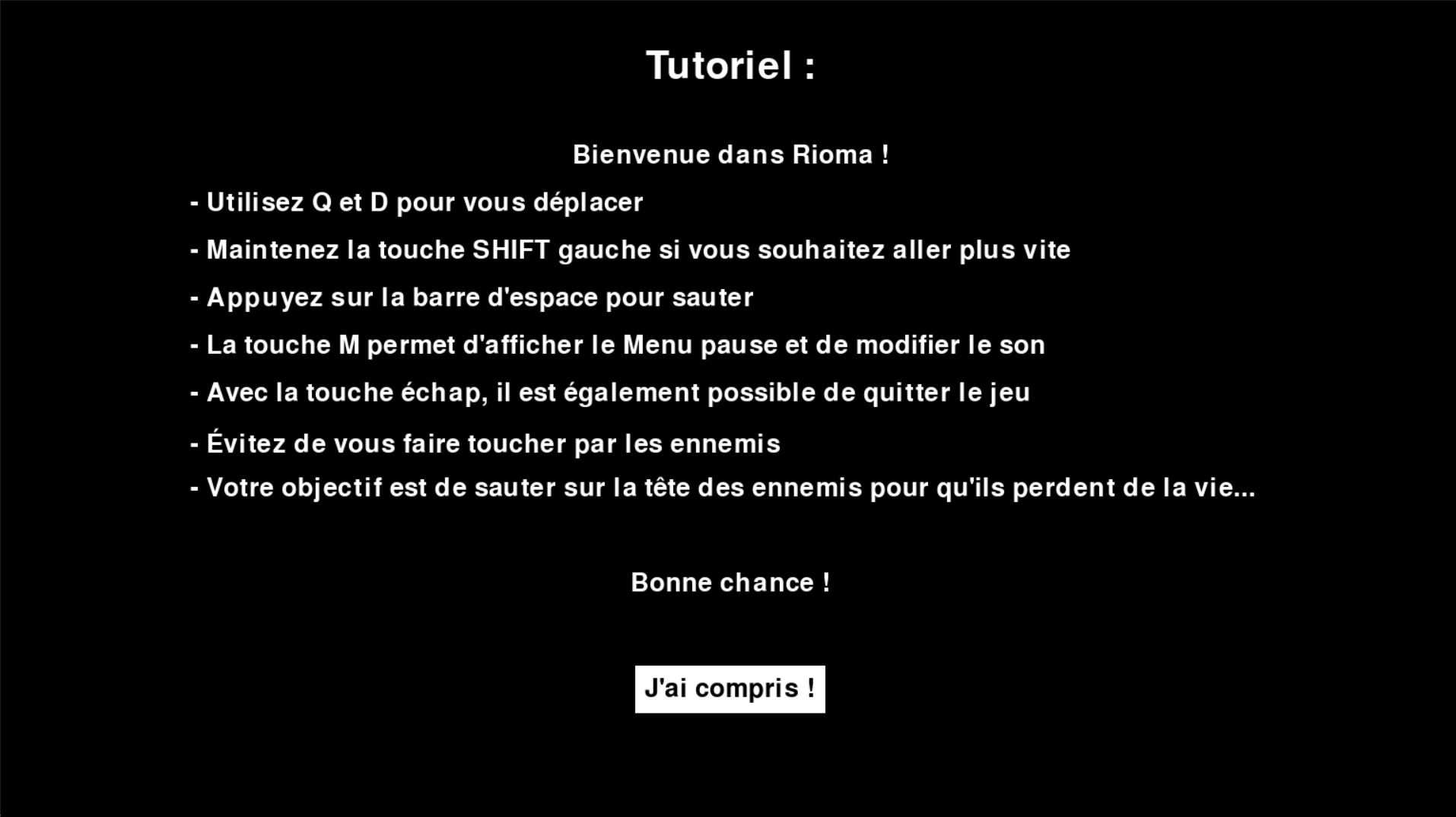
Menu principal :



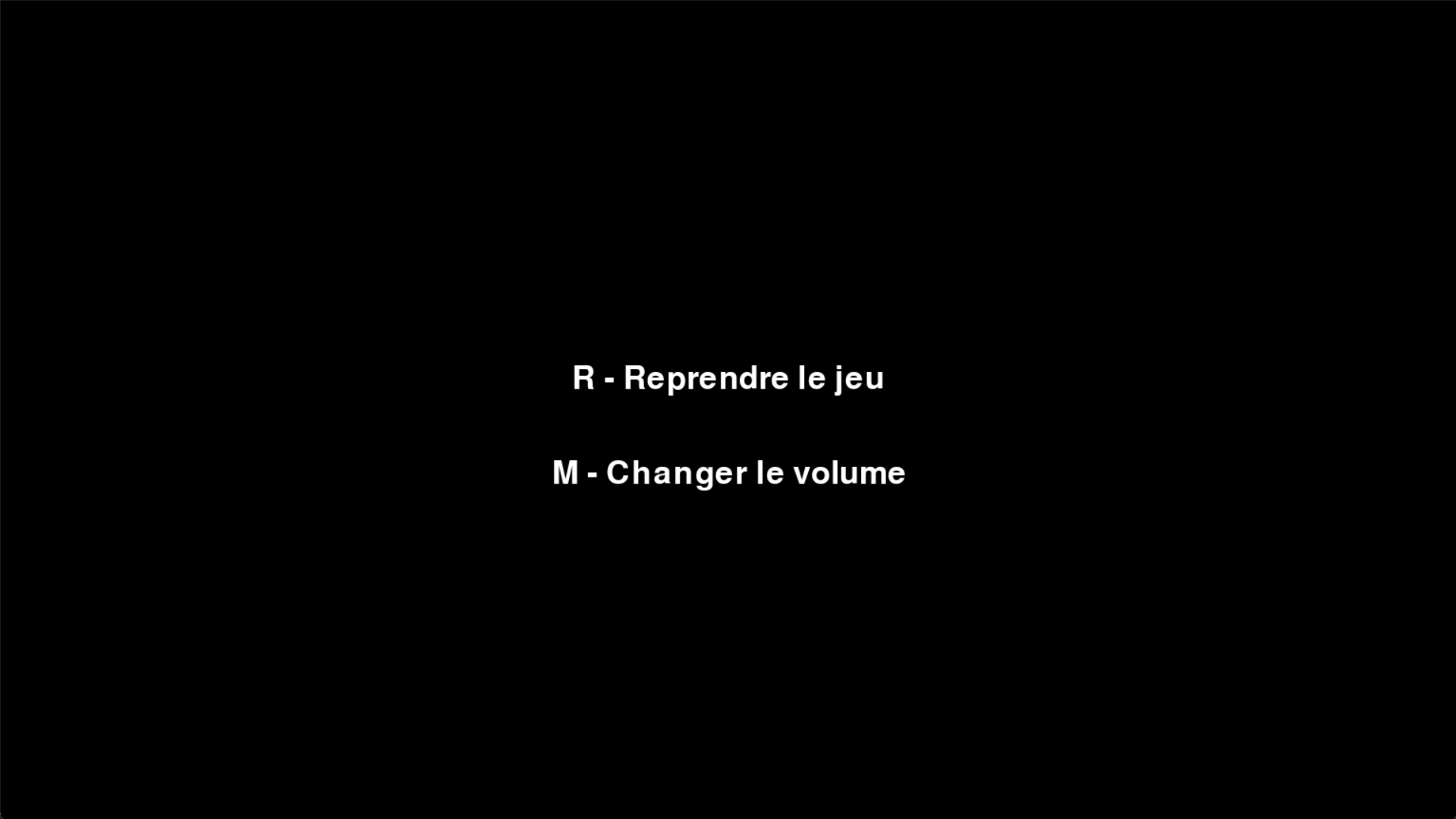
Menu pour le choix du personnage :



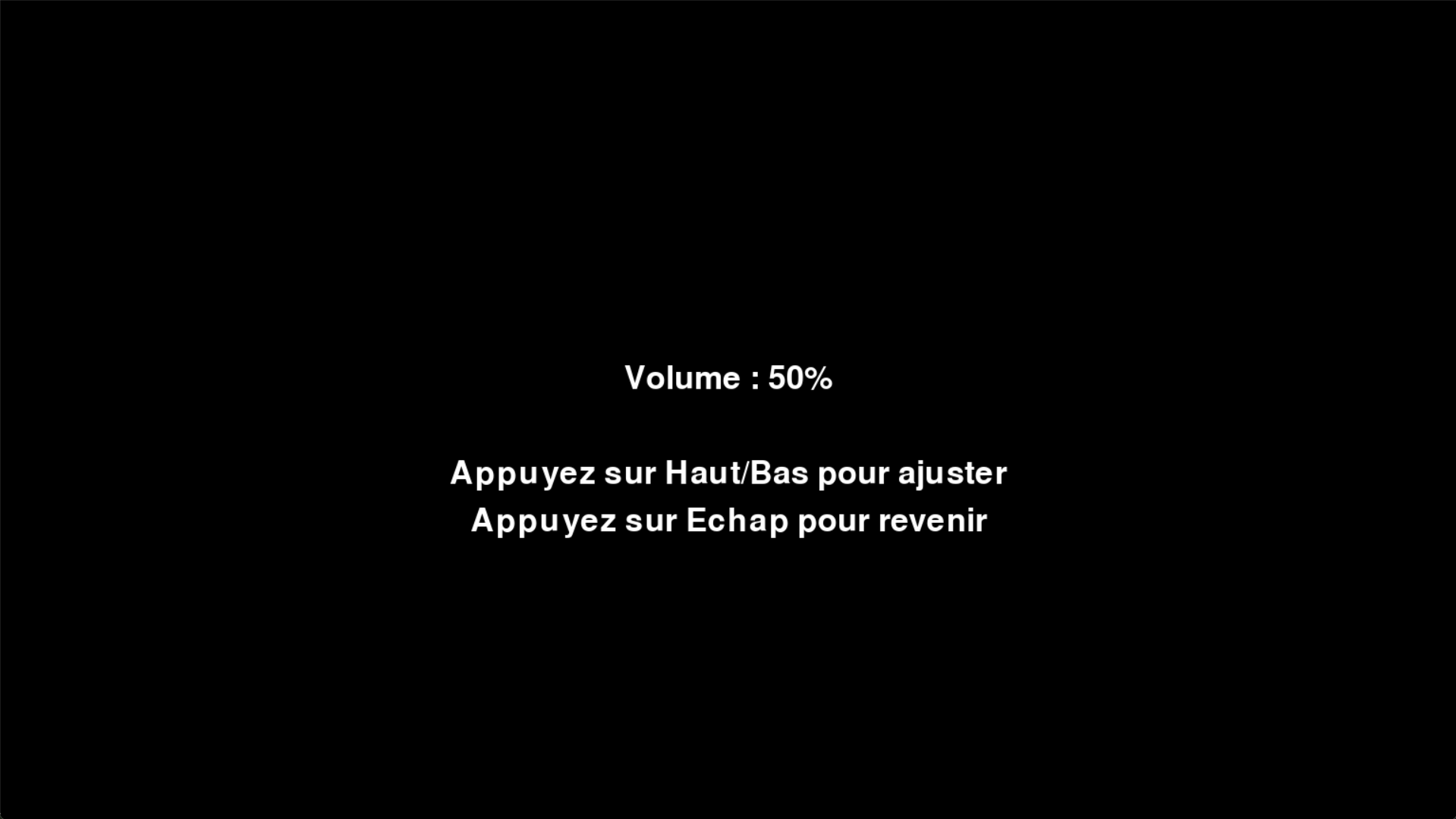
Le tutoriel avant le début de la partie :



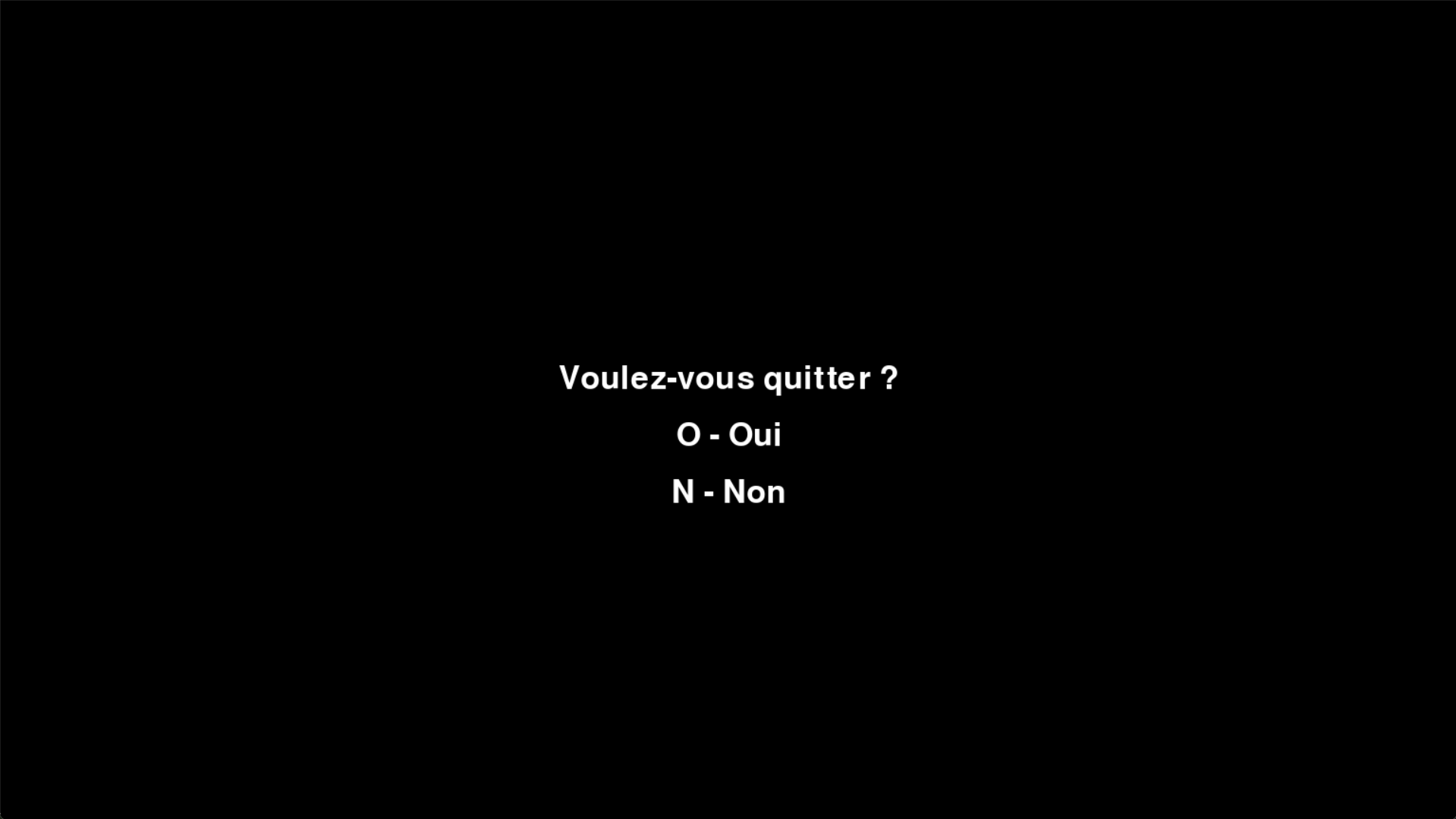
Menu pause :



Menu volume :



Menu quitter :



Pour suivre ce côté simpliste et ergonomique, un fond noir avec des textes blancs paraissaient la meilleure option pour suivre avec le fond du menu principal. De manière pratique, j’ai essayé d’être la plus succincte et compréhensible possible avec un texte court, centré, lisible et en indiquant la touche associée à l’action voulue.

#### Conception détaillée

Pour réaliser ces différents menus, de simples fonctions incluses dans le module Pygame étaient suffisantes. J’en parle de manière plus technique dans la partie d’après.

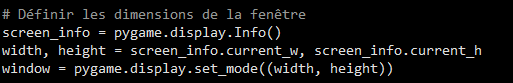
### Implémentation

#### Détails techniques de la réalisation

Je vais, à présent, aborder la partie un peu plus technique, concernant le code et leurs actions.

Pour l’exemple, il sera intéressant de montrer comment le menu principal a été créé, étant la première chose que l’on peut voir sur le jeu.

1. Je récupère d’abord les dimensions de l’écran de l’utilisateur pour m’adapter au mieux :

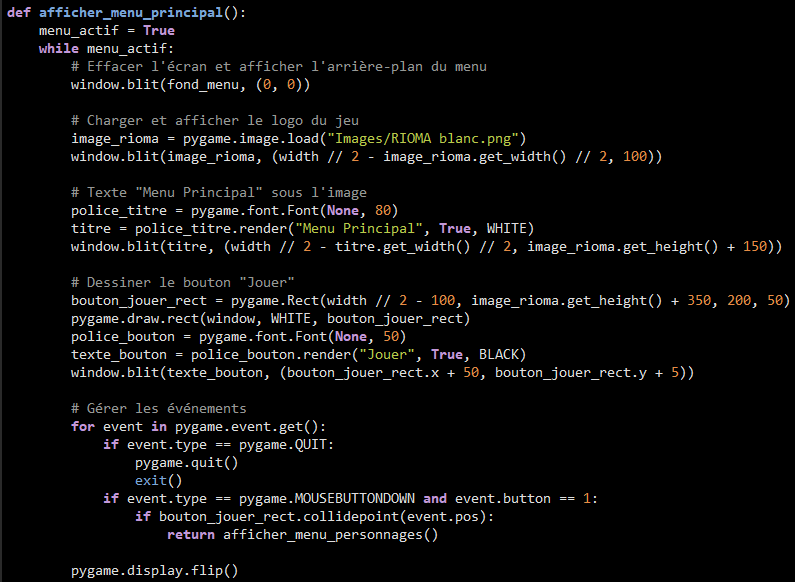


2. J’importe ensuite le fond pour le menu :



3. Je le redimensionne en conséquence par rapport à la largeur et la hauteur de l’écran de l’utilisateur :

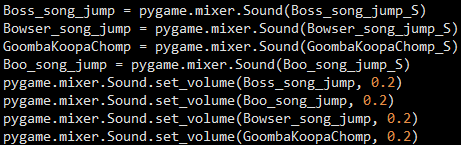


4. Lorsque je l’utilise dans ma fonction pour afficher le menu principal grâce à window.blit(image, (x,y)), mon image de fond s’affiche (voir l’image du Menu Principal) : 

Il fallait dessiner le rectangle du bouton et également gérer l’évènement du clic sur ce bouton pour qu’on soit ensuite amené sur le menu choix du personnage. (La fonction pygame.display.flip() permet d’afficher tout ce qu’on a dessiné juste au-dessus)

##### Problèmes rencontrés

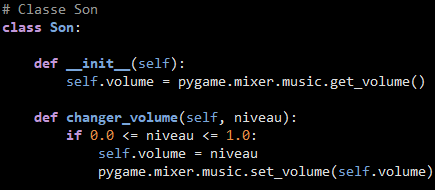
J’ai, personnellement, rencontré plusieurs problèmes lors de la réalisation de certaines tâches :

* Le menu du choix de personnage ne marchait pas bien car lorsque l’on cliquait sur l’un d’eux, le clic ne s’effectuait pas correctement sur le personnage car il fallait une position x et y pour la case.
  + Solution : Corriger la position x et y des cases des personnages.
* Nous souhaitions inclure des effets spéciaux à certains personnages, comme Bowser, qui pouvait lancer des boules de feu mais par manque de temps nous nous sommes limités à une version plus simpliste (sinon, il fallait adapter le tutoriel en fonction du personnage, changer certaines fonctions et attributs, etc.).
  + Solution : On a simplement laissé cette idée de côté.
* Pour la gestion du son, la modification du volume ne se fait que pour la musique et non pour les sauts.
  + Solution : Ces derniers restent tout de même modifiables depuis notre code source : 

##### Bonnes pratiques de codage et patterns de conception utilisés

J’ai dû créer une classe « Son » pour gérer le volume, ainsi que de simples fonctions pour les menus et affichages, puis je les ai appelées lorsqu’elles devaient être affichées à l’écran.

Exemple avec ma classe Son :



La fonction commençant par « pygame.mixer.music » ne se charge que de la musique, tandis que « pygame.mixer.Sound » ne se charge que des sons (effets spéciaux, bruits par exemple).

### Tests unitaires et validation

Pour ma partie, j’ai testé tout ce qui était possible de faire, taper sur des touches qui ne correspondent pas pour voir la réaction du programme mais rien ne se passe, ce qui est plutôt positif car les évènements sont gérés de manière assez simple, si la touche que j’ai spécifiée est pressée, alors une action se passe.

Voici les tests que j’ai effectué :

* **Tests du son** : Vérification que les effets sonores se déclenchent au bon moment et que le volume est correctement ajusté.

→ Vérifié

* **Tests de clic sur les boutons**: Vérification que rien ne se passe si un clic est effectué en dehors du bouton.

→ Vérifié

Aucun bug n’a été trouvé.

### Tests d’intégration

De manière globale, mes fonctions se sont bien implémentées avec les autres travaux de mes camarades, que ce soit lorsque j’ai affiché une image par exemple.

### Discussion

* Pour les choix, nous nous mettions régulièrement en contact pour savoir si le travail que nous faisions correspondait bien aux attentes des autres, si l’un avait une suggestion, on la prenait en compte tout en incluant les idées de tout le monde.
* En ce qui concerne le côté logiciel, ayant utilisé Python, il ne fallait pas non plus s’attendre à une interface en 3D avec des animations tout droit venue d’un jeu avec une technologie beaucoup plus avancée, c’est pourquoi un jeu rétro était la solution. Python reste un langage de programmation qui n’est pas forcément optimal pour les jeux vidéo.
* D’autres camarades ont pu créer un jeu sur la même plateforme que nous (ou bien même sur Unity) mais nos projets restaient tout de même différents les uns des autres. Python restait tout de même une bonne solution étant donné les bibliothèques mises à disposition.

## Etudiant 3 - Romain Debrue

### Introduction

Mon rôle dans ce projet consistait à créer les éléments visuels et sonores de notre jeu. J’ai donc créé et modifié toute la partie artistique dont les sprites, les arrière-plans, les animations et les bandes sonores.

Notre idée était de faire ressentir un sentiment d’« inconfort », c’est-à-dire faire ressentir à la personne qui joue l’envie de se dire :

« Hmm, il y a quelque chose qui cloche... »

Grâce à des images et des sons bien choisis, je pense avoir réussi à créer cette ambiance.

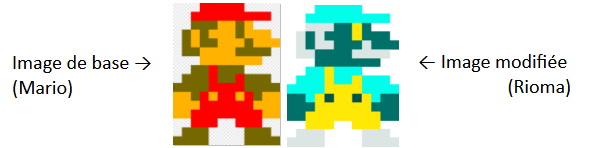
### I. Aspect Visuel

#### a) Une direction artistique fidèle à l’original, avec un petit truc en plus !

Pour la direction artistique nous avons décidé à l’unisson (puisque le jeu est inspiré du jeu « Super Mario Bros » sur NES) d’avoir un style « NES » comme celui-ci :

Néanmoins, puisque le scénario reste globalement les « gentils » (Mario, Toads) devenus les « méchants » et qu’on a le choix de jouer les méchants du jeu original pour le battre, j’ai fait en sorte d’altérer le monde original en mode « négatif ».

Voici un petit exemple avec l’image du boss, prénommé Rioma :



Il a fallu faire un arrière-plan pour notre combat de boss également, donc pour cela je me suis inspiré du dernier niveau du jeu original.

Image originale :

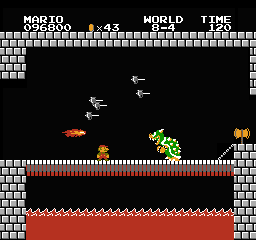
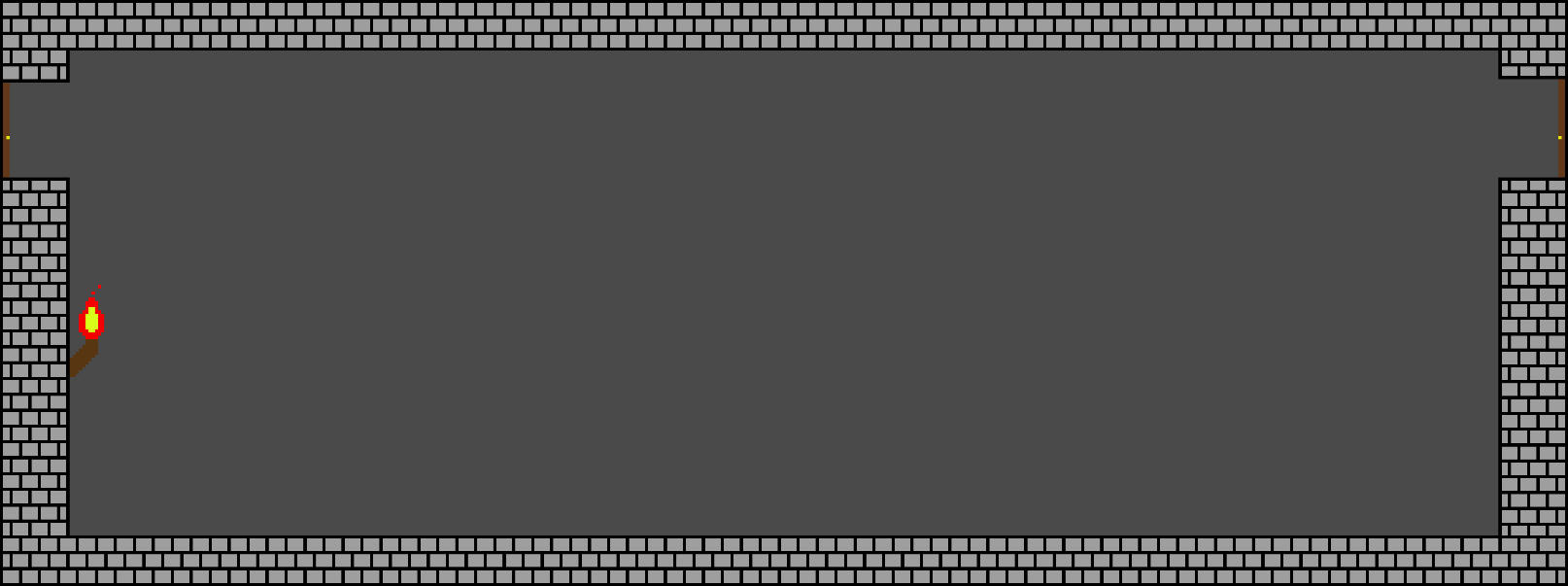


Image modifiée :

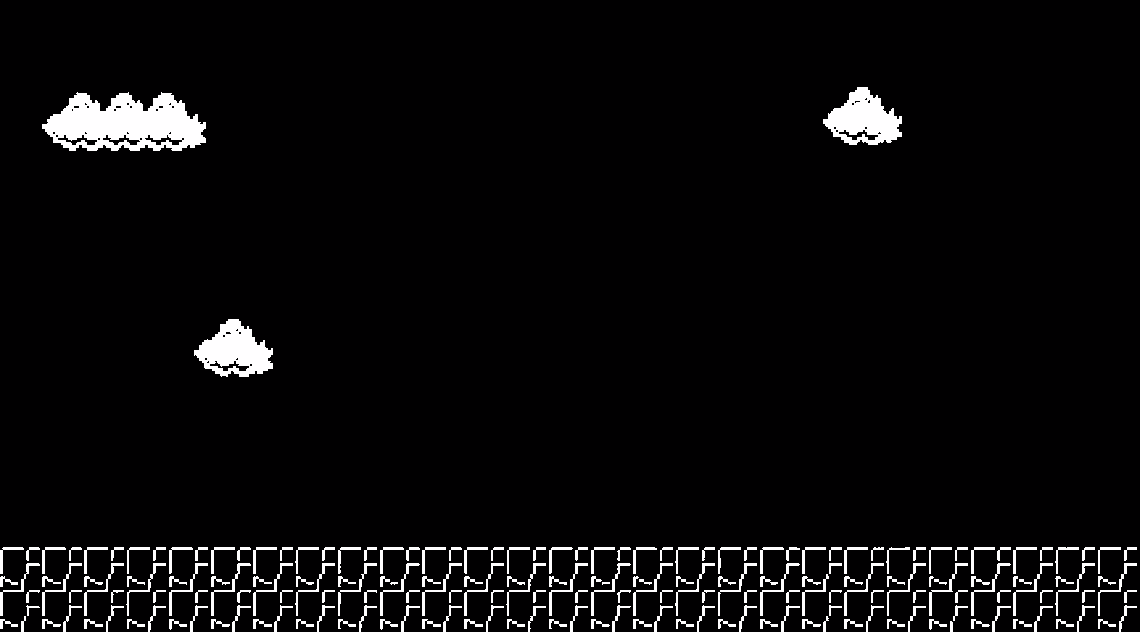
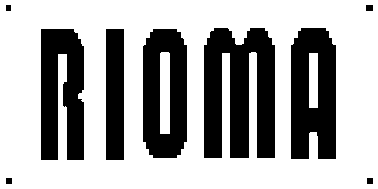


Le fond du menu du jeu est très parlant, avec ses couleurs altérées nous pouvons sentir qu’il y a quelque chose qui cloche et qu’il ne s’agit pas que d’une simple copie du jeu Super Mario Bros…

Menu de base :



Menu altéré :



Le choix du noir et blanc comparé aux couleurs vives montrent bien le contraste entre les deux ambiances.

La seule chose qui ne change pas restent les « ennemis » (donc ici les personnages sélectionnables) car leurs couleurs vives rendent leur design plus attrayant et gai, étant devenus les gentils du jeu.

#### b) Les outils utilisés pour créer les images et les fonds

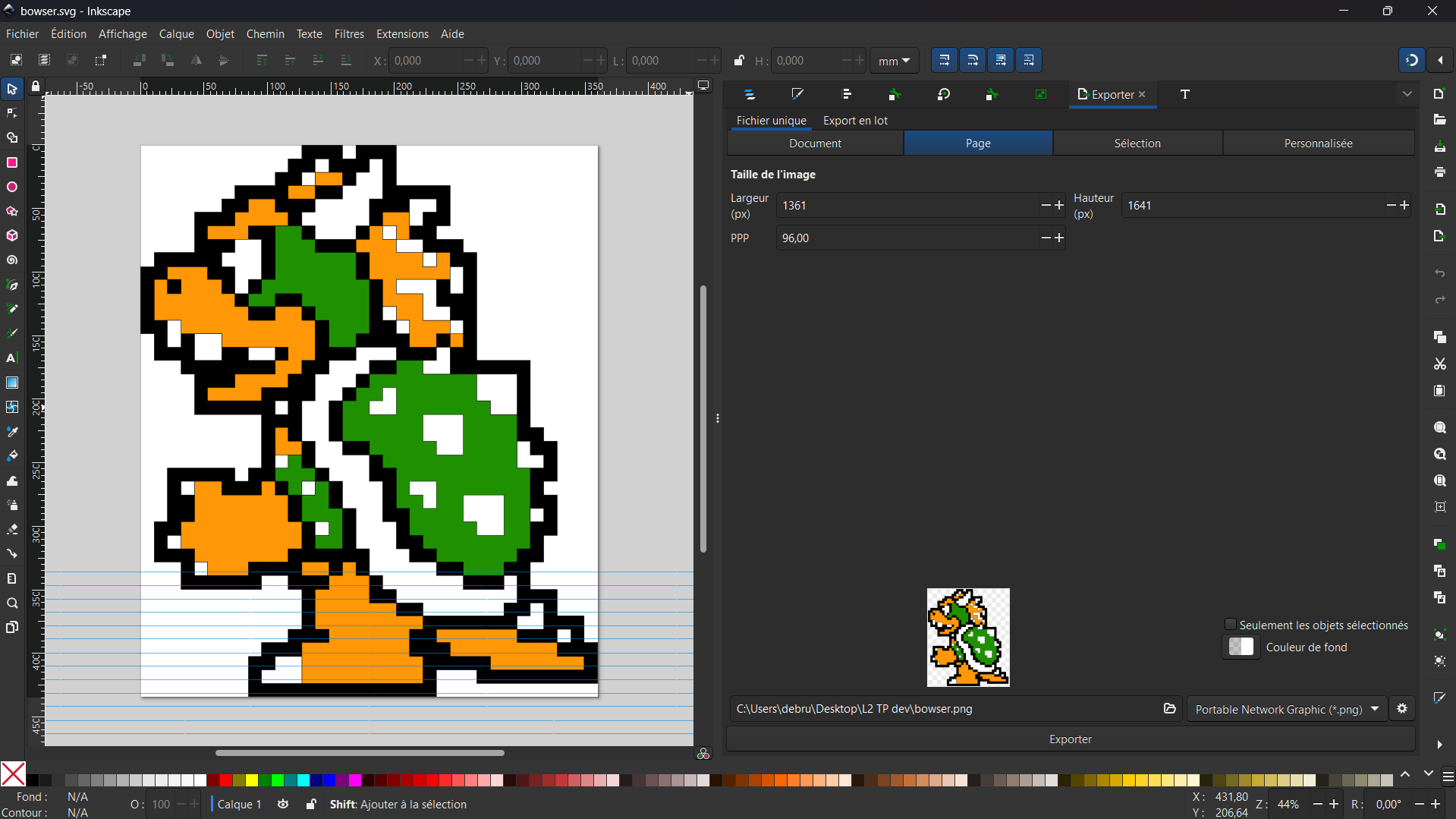
##### 1) Inkscape



- Pour les images et les fonds de victoire/défaite j’ai utilisé **Inkscape**.

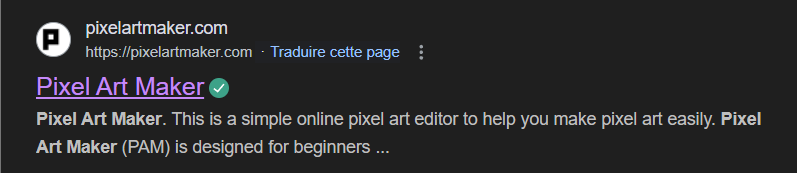
C’est un logiciel facile d’utilisation et gratuit qui permet de créer des dessins vectoriels et de les exporter au format .png (ou bien au format .svg pour pouvoir retoucher l’image dans le futur si besoin).

L’interface ressemble à ceci :



##### 2) Pixel Art Maker

Pour le fond du niveau, j’ai eu besoin d’utiliser un site appelé « Pixel Art Maker » pour être plus efficace.



- Ce site web permet de dessiner pixel par pixel et permet d’enregistrer nos créations au format .png.

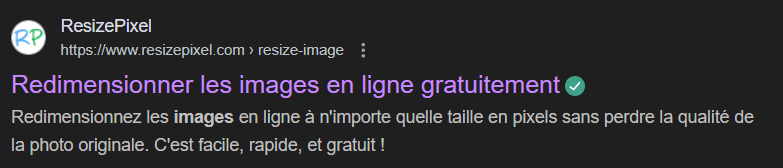
Ce site est aussi très simple d’utilisation, assez limité sur les outils mais cela suffisait amplement pour ma tâche à réaliser.

L’interface ressemble alors à ceci :

Comme expliqué au-dessus, il suffit simplement de dessiner pixel par pixel, en plus des outils basiques pour dessiner des lignes, utiliser le seau ou encore changer la couleur du pixel qui sont également mis à disposition.

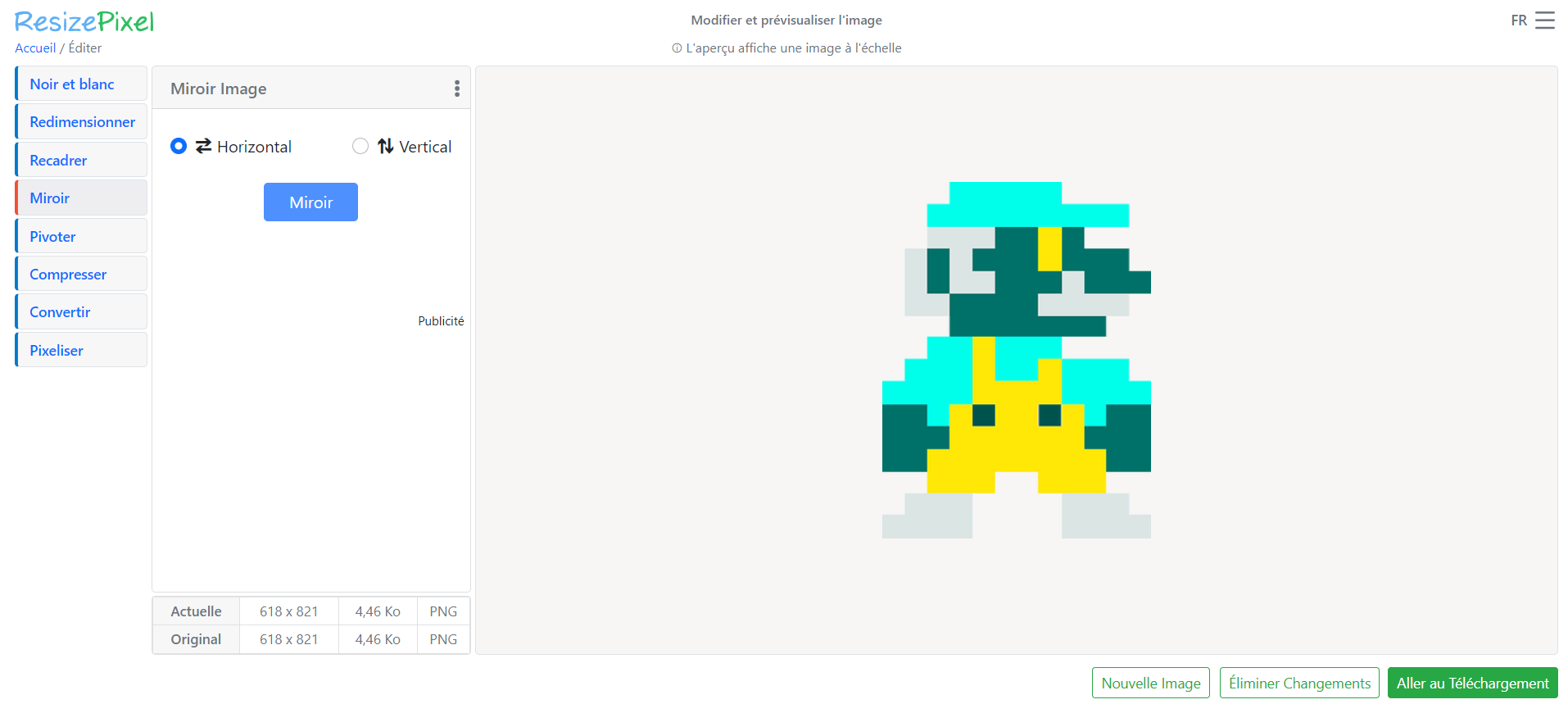
##### 3) ResizePixel

Parfois, les dimensions des images que j’envoyais étaient trop grandes ou trop petites, il était alors nécessaire de les redimensionner, le site web ci-dessous que j’ai utilisé était alors très utile :



Il est possible de transformer en noir et blanc, redimensionner, recadrer, retourner, pivoter, compresser, convertir ou pixeliser une image.

Voici à quoi ressemble l’interface avec toutes les options sur la gauche :



Il suffit d’importer son image et de modifier ses dimensions et le tour est joué !

### II. Musique et Son

#### a) Une bande-son modifiée et adaptée au jeu

Pour les musiques, il fallait de nouveau retrouver ce côté « altéré » donc, avec mes collègues, nous avions songé à une musique terrifiante... Plusieurs idées nous ont alors traversé l’esprit, comme la musique de la maison Boo dans Super Mario Bros sur Wii, mais c’est comme s’il manquait quelque chose…

Ce fut terrifiant, certes, mais ça n’était pas suffisamment « dérangeant » pour autant, nous nous sommes alors dirigés vers une autre idée qui nous a amené à inverser la bande-sonore et, surtout, celle de la musique principale du jeu Super Mario Bros sur Wii, c’est-à-dire la musique qui se lance dans les châteaux des mini-boss (<https://www.youtube.com/watch?v=07ZVb4VWJBE&t=91s&ab_channel=GilvaSunner%3AArchive>).

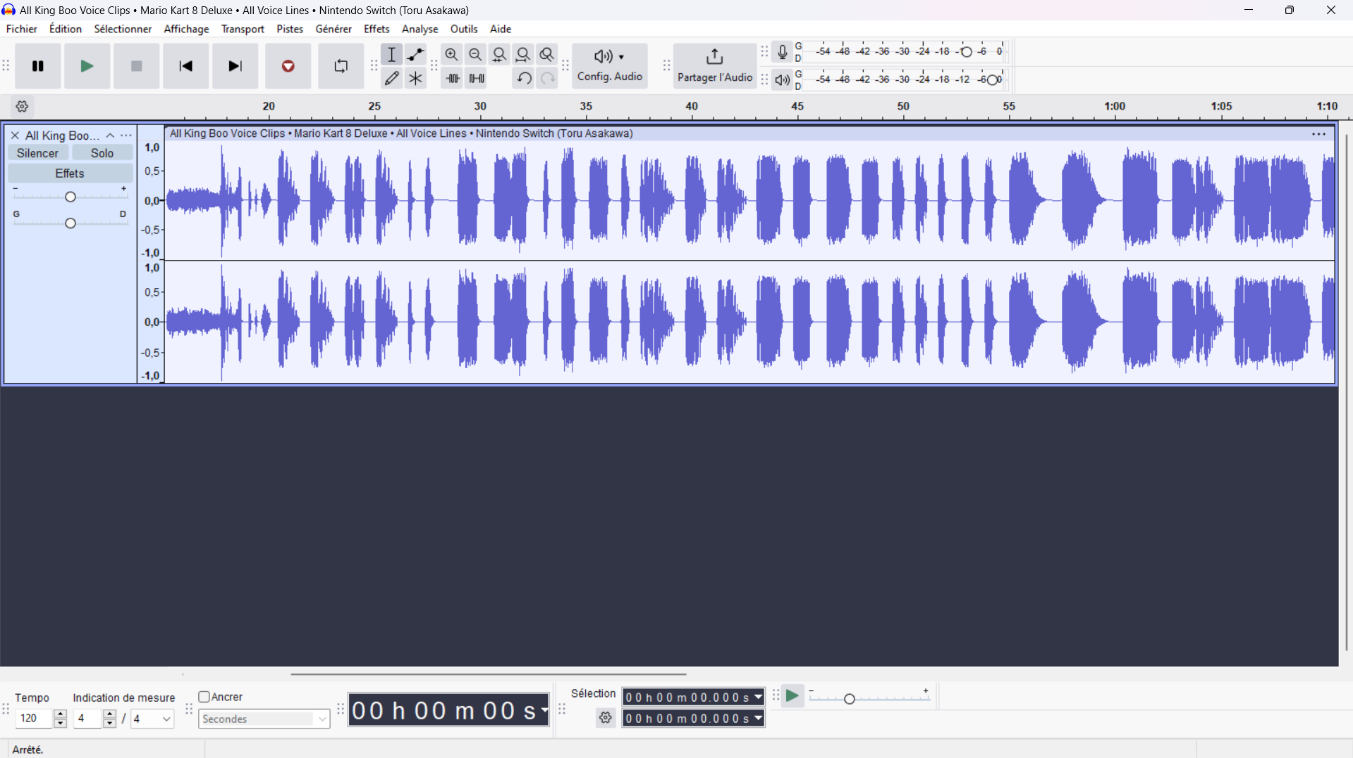
Pour les autres sons, c’est-à-dire ceux des sauts, j’ai utilisé les mêmes que ceux du jeu Super Mario Bros sur NES, cependant, j’ai modifié le son pour certains personnages. Par exemple, Rioma, qui a un saut avec un effet « reverb » (un peu comme de l’écho) pour donner un côté fantomatique, ainsi que Bowser où le bruit du saut est plus grave, étant donné le fait qu’il soit très lourd.

J’ai aussi importé la musique de victoire du jeu vidéo Final Fantasy VII, étant la plus iconique, et la célèbre « trompette triste » pour la défaite.

#### b) L’outil utilisé pour l’audio du jeu

- Pour les musiques et les sons j’ai utilisé le logiciel **Audacity**. C’est un logiciel gratuit qui permet d’enregistrer des sons et de les modifier.

- Pour tous les sons, j’ai été les chercher sur YouTube pour qu’ils puissent correspondre à notre jeu, je les ai ensuite modifiés en conséquence pour qu’ils se collent beaucoup plus aux différents personnages.

L’interface d’Audacity ressemble à ceci :

Il est aussi très simple d’utilisation et possède beaucoup de fonctionnalités. Comme par exemple : modifier la hauteur du son, rendre plus aigu ou plus grave la bande-sonore ou bien ajouter un effet de « reverb », comme expliqué précédemment.

# III. Suite et fin de la partie générale

## Conclusion et perspectives

Cette expérience nous a tous permis de communiquer et de mettre en lien nos différents rôles dans un seul et même projet commun. L’entente entre collègues est également essentielle si l’on souhaite travailler dans une bonne ambiance, ce qui nous motive à travailler tous ensemble plutôt qu'individuellement. Comme l'a dit un de nos professeurs : « Un projet, c’est avant tout une aventure humaine », et c’est justement cette dimension humaine que nous avons tous les trois retenu et apprécié.

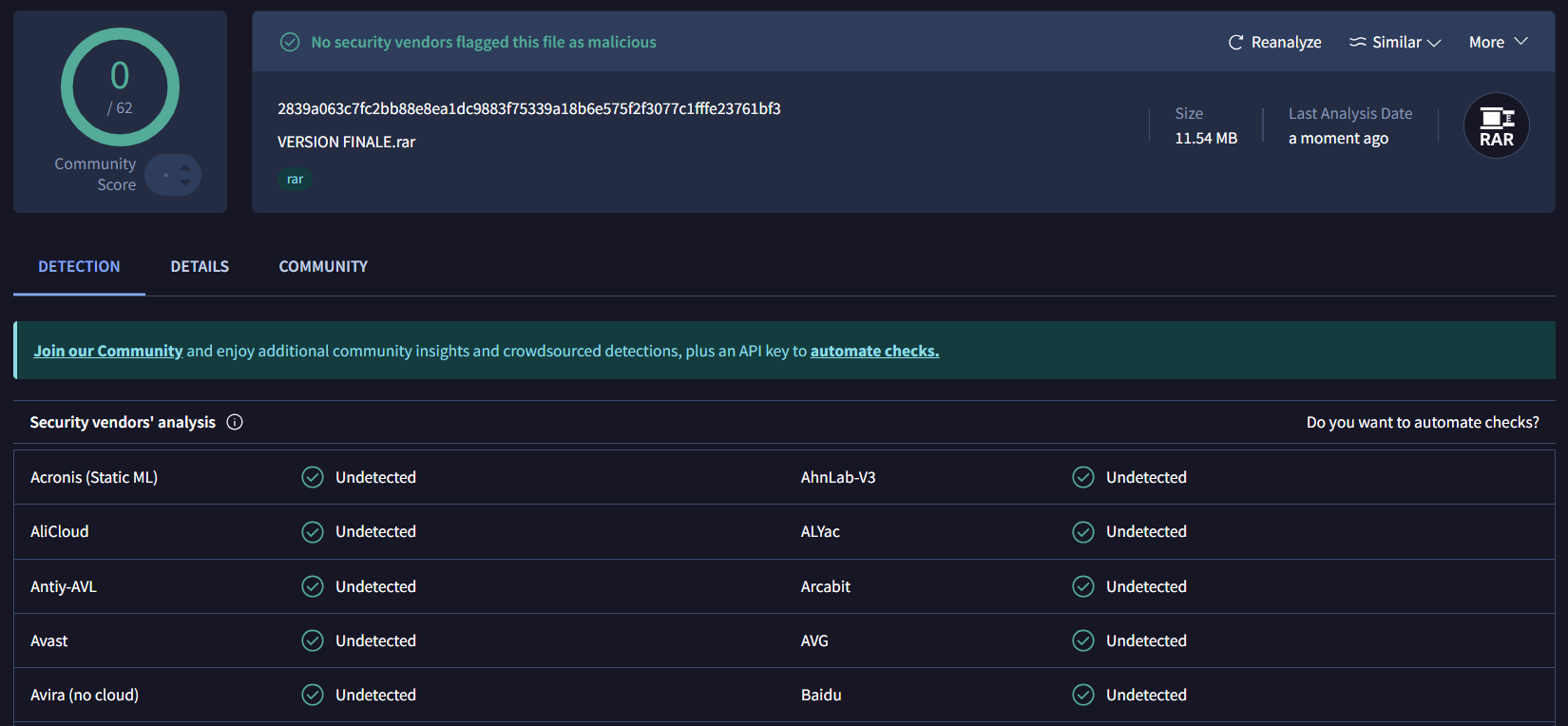
Globalement, ce projet nous a permis à la fois de renforcer nos compétences techniques, mais également à améliorer notre capacité à communiquer et collaborer de manière efficace dans un cadre de développement partagé, grâce à GitHub également.

Nous avions songé à ajouter un « mode Noël », étant donné l’approche des fêtes de fin d’année, pour rajouter un côté un peu plus « fun » en rajoutant des bonnets de Noël aux personnages, un décor un peu plus festif, etc. Également peut-être que nous pourrons ajouter des boules de feu pour Bowser et Rioma, rajoutant des niveaux et une difficulté supérieure.

## Annexes

Notre GitHub est accessible à la page suivante : <https://github.com/Deerman59/TP-Developpement>.

Toutes les captures d’écran ont été déjà été collées directement dans notre rapport à l’endroit correspondant.

Nous avons passé notre jeu sur le site VirusTotal, permettant de détecter si les fichiers comportent des virus, voici le résultat :

## Bibliographie / Sitographie / Références

Lien du résultat de la vérification antivirus sur VirusTotal :  <https://www.virustotal.com/gui/file/2839a063c7fc2bb88e8ea1dc9883f75339a18b6e575f2f3077c1fffe23761bf3/detection>

Site officiel de Pygame sur son installation : <https://www.pygame.org/wiki/GettingStarted>

Site officiel de Pygame sur d’autres créations : <https://www.pygame.org/tags/all>

Site officiel de Pygame sur la documentation des fonctions : <https://www.pygame.org/docs/>

Site officiel de Thonny (notre IDE utilisé pour Python) : <https://thonny.org/>