

8a, avenue V. Maistriau B-7000 Mons

Tél : +32 (0)65 33 81 54

Fax : +32 (0)65 31 30 51

E-mail : tech-mons@heh.be

[www.heh.be](http://www.heh.be)

Travail de fin d'études

# Développement d'une application ludique sur Android dans le cadre d'un projet de recherche pédagogique

*Je voudrais remercier toutes les personnes qui m'ont soutenu durant l'élaboration de ce projet mais aussi concernant la reprise d'un bachelier de plein exercice.*

*Un grand merci à mes promoteurs, Mr Vandenberghe et Mr Malaise pour leur confiance, et sans qui je n'aurais pu prendre part à un projet si gratifiant.*

*Un grand merci encore à mes collègues de promotion. Pour l'entraide dans les périodes d'examen et la bonne humeur qu'ils ont su transmettre en classe.*

*Merci à mes proches, parents et amis, pour leur présence durant cette aventure. Leur soutien pendant ces années, leurs critiques et leurs relectures, ont été d'une aide inestimable.*

*Enfin, un grand merci à ma compagne, Aurore Dupuis, qui n'a de cesse de croire en moi même dans les moments difficiles.*

## Table des matières

Table des matières .....	3
Table des figures .....	5
1. Abstract .....	6
2. Introduction .....	8
3. Choix de l'environnement de développement .....	10
3.1. Unity.....	10
3.2. GameMaker Studio.....	11
3.3. Godot.....	13
4. Mise en place du Moteur.....	14
4.1. Les outils Android .....	14
4.2. Export Android.....	16
5. Godot : De Nœuds et de scènes .....	17
5.1. Les scènes .....	17
5.2. Les nœuds.....	18
5.3. Bien choisir ses nœuds .....	19
5.4. Scripts et Signaux.....	20
5.5. Les ressources et Android.....	21
6. Réalisation du projet.....	23
6.1. Structure et Mockup.....	23
6.2. Les écrans de jeux.....	25
6.3. Les éléments fixes : Merge, Branch et instanciation.....	25
6.4. La souplesse de GDScript et l'instanciation procédurale .....	27
6.5. Changement de design .....	31
6.6. L'ampoule .....	32
6.7. Le memory .....	35
6.8. La gestion d'utilisateur .....	36
6.9. Les Scores.....	37
7. Problèmes et mésaventures .....	38
7.1. GameMaker, la version perdue .....	38
7.2. Les ressources.....	39
7.3. La récursivité, saturation des ressources système et boucle infinie.....	39
7.4. Les ressources 2 le retour : Attack of the node.....	40
7.1. Le module Caméra.....	41

---

8.	Conclusions et améliorations.....	42
8.1.	Bilan .....	42
8.2.	Points forts du projet.....	42
8.3.	Limitations .....	43
8.4.	Évolution future.....	44
8.5.	Conclusion finale.....	44
9.	Bibliographie .....	46

## Table des figures

Figure 1: Pokémon Go succès commercial développé sur Unity .....	11
Figure 2: Sans de Undertale .....	12
Figure 3: Deponia, porté sur IOS grâce à Godot .....	13
Figure 4: Android SDK manager .....	14
Figure 5: Configuration d'export Android pour Godot .....	15
Figure 6: Template d'export Android .....	16
Figure 7: Écran principal du projet .....	17
Figure 8: Scène d'un élément graphique d'interface .....	18
Figure 9: Création d'un nœud .....	19
Figure 10: Signaux .....	20
Figure 11: Ajout de signal en script .....	21
Figure 12: Déclaration des ressources .....	22
Figure 13: Ressource correcte pour Android .....	22
Figure 15: Mockup de l'écran principal .....	24
Figure 14: Structure de l'application .....	24
Figure 16: Mr Hibou, la mascotte de l'application .....	25
Figure 17: Exemple de merge entre deux scènes .....	26
Figure 18: Liste de séries de mots générée dynamiquement .....	27
Figure 19: Instanciation dynamique .....	28
Figure 20: L'écran choix du joueur .....	29
Figure 21: Schéma du Design Pattern Singleton .....	30
Figure 22: Changement de design bénéfique .....	32
Figure 23: Structure du code du jeu de l'Ampoule .....	33
Figure 24: Appel des sous-fonctions de l'ampoule .....	34
Figure 25: Le jeu du memory .....	35
Figure 26: Création de joueurs .....	36
Figure 27: Scores et gestions joueur .....	37
Figure 28: Plan tarifaire de GameMaker Studio 2.0 .....	38
Figure 29: Exemple de récursivité .....	40

## 1. Abstract

Dans le cadre d'un projet de recherche pédagogique, M. Vandenberghe s'intéresse à l'apprentissage de vocabulaire par des enfants de troisième maternelle. L'idée est d'exploiter et de prolonger l'apprentissage qui se fait lors des lectures d'histoires en classe

Le projet consiste à faire jouer les enfants à des jeux sur tablette qui vont leur permettre de découvrir, d'apprendre et de mémoriser des mots qui appartiennent au même champ lexical que celui qui a été travaillé lors des séances consacrées à l'étude des histoires.

Pour réaliser l'application nous avons choisi d'utiliser Godot. Beaucoup de moteurs existent et chacun offre une variété de fonctionnalités intéressantes. Ayant plusieurs exigences à satisfaire pour le projet d'étude, mais aussi pour mes projets personnels et futurs, le choix final fût Godot.

Afin de faciliter le paramétrage de Godot pour l'export Android, j'ai utilisé certaines fonctions d'Android Studio. Le reste de la préparation pour l'export Android s'est fait sous Godot en lui renseignant les divers chemins de fichiers nécessaires.

Avant de pouvoir commencer le projet, j'ai dû apprendre comment Godot fonctionnait en commençant par la création d'une scène. Cette scène représente un écran de jeu et contiendra des nœuds qui constitueront tout élément de notre scène. La notion de scène se complique par la suite puisque chaque nœud peut être une scène à part entière afin de facilement l'instancier en tant que nœud dans d'autres scènes.

Chaque nœud dispose de méthodes et de paramètres qui lui sont propres. Une hiérarchie est suivie et applique le concept d'héritage de la programmation orientée objet. Il en va de même pour les signaux dont chaque nœud dispose pour créer des interactions et des déclenchements de script.

J'ai aussi appris comment gérer les ressources. Lors de l'ajout d'une ressource au projet (images, sons,...) Godot créera de lui-même un fichier « .import ». C'est ce fichier que Godot utilisera pour retrouver l'emplacement des ressources lorsque l'application sera en exécution.

La structure de l'application ayant été définie lors d'un entretien avec Mr. Vandenberghe, j'avais une liste d'écrans de jeux comme objectif de départ. J'ai commencé par ajouter les

éléments constants à mes écrans de jeu. Chacun de ces éléments fût également sauvegardé en tant que scène afin de pouvoir être réutilisé facilement dans chaque scène.

Pour les éléments variables, comme la liste de joueurs ou les séries de mots, j'ai utilisé les signaux et les scripts. En liant un script à un élément et un signal, il est possible de créer des événements personnalisés. Dans mon cas, lorsque la grille qui contiendrait les avatars des joueurs serait prête, celle-ci exécuterait les fonctions que j'ai codées et qui ont pour but de récupérer les avatars des joueurs et de les ajouter dans cette même grille.

Lors du développement, je me suis aperçu que certains éléments d'interface devenaient inutiles ou faisaient doublons avec d'autres. J'ai adapté le design en fonction des besoins du projet tout en restant le plus pertinent possible.

Pour ce projet, deux jeux sont prévus, l'ampoule et le memory. Chacun offre un challenge de programmation différent.

Les problèmes rencontrés lors de ce projet ont été nombreux : version beta sur GameMaker, surcharge du système par récursivité, problèmes de chargement des ressources ou d'export Android,... Chacun de ces problèmes m'a permis de corriger non seulement le projet de manière plus stable, mais aussi mes connaissances. Ils m'ont fourni des pistes de réflexion intéressantes qui m'ont poussé à rechercher toujours plus loin les raisons de leur présence.

En conclusion, ce projet est pour moi un succès. Il est viable en l'état et mes objectifs sont pour la majeure partie atteints. Certes il manque la gestion utilisateur, mais cela m'offre un challenge intéressant pour la suite. Même s'il s'agit ici d'un projet de fin d'études, son maintien au fil du temps me permettra d'améliorer son fonctionnement au fur et à mesure de ma propre évolution.

## 2. Introduction

Dans le cadre d'un projet de recherche pédagogique en Haute École, M. Vandenberghe s'intéresse à l'apprentissage de vocabulaire par des enfants de troisième maternelle.

Lors des lectures d'histoires en classe, les élèves sont amenés à enrichir leur capital mot. L'idée est d'exploiter et de prolonger l'apprentissage qui se fait à cette occasion.

Comment ? L'idée de Mr Vandenberghe est de faire jouer les enfants à des jeux sur tablette qui vont leur permettre de découvrir, d'apprendre et de mémoriser des mots qui appartiennent au même champ lexical que celui qui a été travaillé lors des séances consacrées à l'étude des histoires.

Le souhait de M. Vandenberghe est donc de disposer d'une application sur tablette proposant plusieurs « jeux » ou activités à base d'images et de sons.

Afin de réaliser ce projet, nous allons donc nous pencher sur l'étude de moteur de développement de jeu. Pour que notre choix soit valable, il faudra prendre en compte certaines fonctionnalités spécifiques.

Les moteurs de développement multiplateforme étant monnaies courantes de nos jours, nous allons en comparer certains afin de déterminer lequel sera le plus apte à remplir non seulement les objectifs de notre projet, mais également ses possibilités sur des projets subséquents.

Notre objectif sera donc, avec l'aide de ce moteur, de développer une application qui sera à destination du Play Store<sup>1</sup>. Notre application est à destination des professeurs qui feront jouer leurs élèves aux divers jeux de l'application afin de récolter des données pertinentes sur leur apprentissage.

Ils pourront ensuite jauger de l'influence que les jeux auront sur le développement des connaissances de l'enfant en lien avec les sessions d'exercices qu'ils dispenseront à leurs classes d'élèves.

Ayant pour but de développer à terme diverses applications à destination d'Android, ce projet me permettrait d'apprendre énormément sur le sujet. Qui plus est, ayant moi-même eu pas mal de soucis d'apprentissage par le passé, l'objectif même de l'application me paraît

---

<sup>1</sup>Magasin en ligne officiel d'application Android

pertinent surtout dans le contexte actuel où l'apprentissage à distance, voir purement autodidacte, représente un atout des plus précieux pour tous.

Nous partirons donc des informations fournies par Mr Vandenberghe pour réaliser l'application. Ayant une idée précise de ce qu'il attend, il a pu fournir de précieuses informations sur l'objectif à accomplir.

Le plan de l'application sera de fournir en premier lieu un écran pour la sélection du joueur, ce qui par la suite nous permettra d'enregistrer ses scores. Il lui sera ensuite présenté une liste de mots à travailler parmi lesquels il fera son choix. Viendra ensuite le choix du jeu auquel il voudra jouer.

Nous utiliserons les possibilités de scripting afin d'implémenter les fonctions plus pointues de notre application. Même si l'interface graphique du moteur permet déjà beaucoup de choses, nous avons pour objectifs le maintien de l'application sur le long terme ainsi que la notion d'aléatoire dans le choix des réponses fournies aux joueurs. Ce qui ne sera possible que par l'ajout de fonctions en code sur nos éléments.

### 3. Choix de l'environnement de développement

Il existe plusieurs solutions concernant le développement d'applications sous Android. Android studio, bien évidemment, mais aussi des moteurs de jeux plus largement utilisés selon les besoins et secteurs visés par le projet.

Dans notre cas, puisque l'on parle d'un « jeu » sur Android, il est pertinent de se pencher sur les possibilités offertes par les divers moteurs de développement disponibles.

En effet, ces derniers offrent des options permettant de rendre le développement d'une application ou d'un jeu beaucoup plus fluide. Mais par où commencer ?

Afin de choisir un moteur adapté au projet, ce dernier doit répondre à quelques critères :

1. Permettre l'export vers Android
2. Être gratuit
3. Avoir une documentation bien fournie

Plusieurs options ont été envisagées pour le projet. Le nombre de moteurs étant copieux et varié il a d'abord fallu faire un tri préliminaire sur les critères cités ci-dessus.

Au final, 3 moteurs se sont démarqués par leur accessibilité et leur potentiel. Néanmoins, même si tous les critères furent remplis, certains n'auraient pas tenu sur le long terme. C'est d'ailleurs le cas du premier moteur de notre liste.

#### 3.1. *Unity*

Gratuit sous conditions, Unity est une solution populaire qui a fait ses preuves. Moteur versatile tout support et disposant d'une communauté des plus actives, je ne doutais pas de pouvoir trouver toutes les ressources techniques nécessaires.

Cependant concernant la gratuité, bien que ce dernier soit gratuit dans sa version la plus basique, il ne l'est plus au-delà d'un certain palier de bénéfices générés par le développeur.

Le moteur est d'ailleurs à l'origine de nombreux jeux populaires. On ne présente plus « Pokémon Go », véritable phénomène social et ludique sorti en 2016 et développé par Niantic et justement sur le moteur de développement Unity.



Figure 1: PokéMon Go succès commercial développé sur Unity

Bien qu'étant un moteur de premier choix et tout à fait valable pour le projet, les tarifications en cas de bénéfices sur l'application est un élément qui m'a personnellement dérangé. Cela est dû à un biais de ma part puisqu'au-delà de ce projet d'études, le moteur se doit de rester complètement gratuit pour mes propres projets futurs.

Fort heureusement, même si certains moteurs sont forfaïtaires comme pour Unity, la plupart sont à achat « unique ». Il se trouve justement que par le passé j'ai eu l'occasion d'acquérir une licence pour l'un d'eux. Nous allons donc faire une entorse à nos pré-requis, puisque aucune dépense ici ne sera pas faite dans le cadre du projet, pour parler un peu de GameMaker Studio

### 3.2. GameMaker Studio

GameMaker est un moteur de développement multiplateforme. Tout aussi versatile et à la prise en main fort aisée, il est un moteur idéal pour le débutant qui souhaite apprendre comment sont faits les jeux.

De par son approche « Drag and Drop » visuel mais aussi par ses possibilités de script personnalisé à appliquer aux divers objets, GameMaker montre un aspect facile d'utilisation mais subtil et puissant à maîtriser.

Sa tarification n'étant pas forfaitaire comme Unity et disposant d'une licence à titre personnel, GameMaker s'annonce comme « LE » moteur de choix pour ce projet.

C'est d'ailleurs sur ce moteur qu'a été développé « Undertale », le jeu de rôle indépendant de Toby Fox, au succès retentissant dès sa sortie.



Figure 2: Sans de Undertale

Avec de tels avantages, difficile de ne pas choisir GameMaker comme fer de lance pour mon projet de TFE. Les ressources sont nombreuses et bien présentes et il permet d'exporter vers Android par le biais d'une licence optionnelle payante que je possède également. Amenant le coût total à zéro euro pour le projet.

GameMaker est cependant récemment passé en version 2.0. Un changement qui s'explique par une refonte complète du cœur du moteur qui est désormais codé en C# au lieu de C++. Les règles de mise en ligne d'applications du Play store stipulent que dorénavant toutes les applications requièrent de supporter les SDK<sup>2</sup> et NDK<sup>3</sup> 64 bit. Ce que ne supporte pas la version 1.4, poussant les gens vers la version 2.0.

Le problème de GameMaker dans le cadre de ce projet est que ma licence n'est valable que pour la version 1 qui s'est arrêtée en 1.49 pour laisser place à la version 2.0. Et bien que ma licence reste valable et l'export Android toujours possible, l'application ne pourra pas être mise en service sur le Play store.

<sup>2</sup> Software Development Kit : ensemble d'outils logiciels facilitant le développement.

<sup>3</sup> Native Development Kit : Interface de programmation d'application (API) Android permettant de développer directement dans le langage du matériel cible.

### 3.3. Godot

Petit moteur multiplateforme ouvert au public depuis Janvier 2014, Godot a su s'imposer comme un moteur incontournable.

Capable de 2D comme de 3D, écrit en C puis C++ avec un code source disponible et éditabile afin de patcher sa propre version ou ajouter ses propres fonctionnalités au moteur, il se présente comme un outil flexible et capable d'à peu près tout ce que vous pourriez imaginer et il est également complètement gratuit. Pas de forfait, pas d'achat unique, rien.

Il présente également l'avantage d'être portable et ne nécessite aucune installation. Il permet donc d'être utilisé d'une machine à l'autre simplement via une clé usb. La cerise sur le gâteau ? Godot pèse à peine 65Mo ! Non content de cela, même en exécution ce dernier ne consomme presque pas de ressources !

Nous avons donc avec Godot un moteur multiplateforme entièrement gratuit, open source, portable, léger, avec une énorme communauté en soutien et qui malgré cela ne fait aucun compromis sur ce qu'il peut accomplir.

Il n'existe à ce jour pas de grosse licence du jeu vidéo ayant été développée sur ce moteur. Étant assez jeune, il lui faut encore se faire une place aux côtés d'autres moteurs beaucoup plus prisés tel qu'Unity. Et c'est sans compter sur la concurrence de tous les autres moteurs professionnels payant du marché, Unreal Engine, CryEngine, GameMaker, Frostbiteetc,...

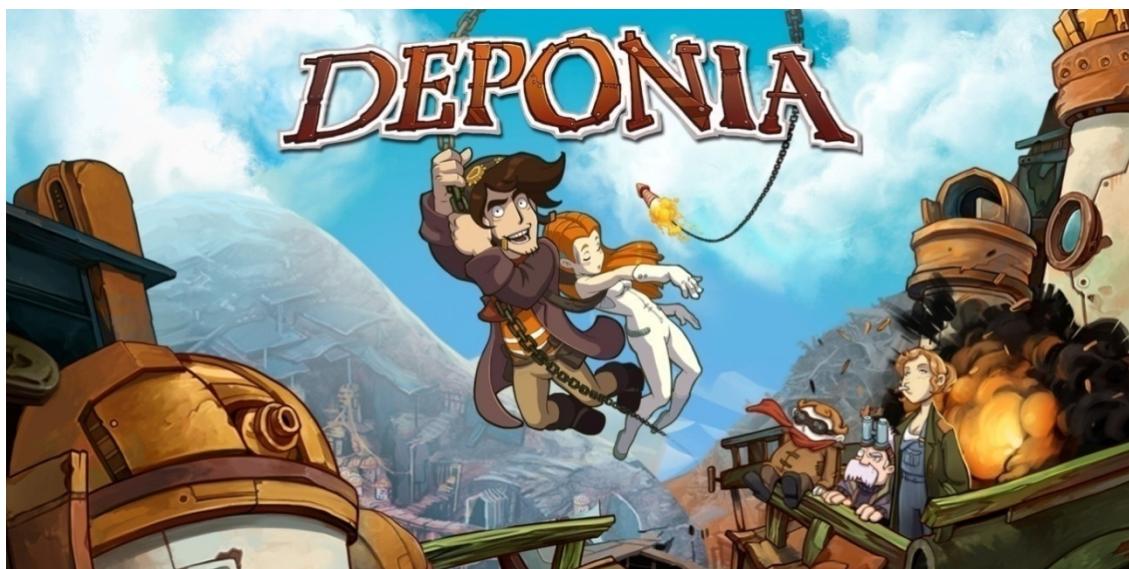


Figure 3: Deponia, porté sur iOS grâce à Godot

## 4. Mise en place du Moteur

Nous avons donc au final choisi Godot pour le projet. Léger mais puissant, complètement gratuit et versatile, il nous offrira une solution parfaitement adaptée non seulement au projet actuel, mais aussi pour nos projets personnels futurs.

Avant toute chose, il nous faut commencer par installer les outils nécessaires à tout développement sur plateforme Android : SDK, NDK, JDK<sup>4</sup>, ADB<sup>5</sup>. Ceux-ci sont facilement trouvables sur le net ou même via les outils d'Android Studio pour certains.

### 4.1. Les outils Android

J'ai personnellement utilisé Android Studio afin de récupérer les SDK des différentes versions d'Android à supporter (qui inclut également ADB) ainsi que le NDK.

Name	API Level	Revision	Status
<input type="checkbox"/> Android R Preview	R	4	Not installed
<input checked="" type="checkbox"/> Android 10.0 (Q)	29	4	Update available
<input checked="" type="checkbox"/> Android 9.0 (Pie)	28	6	Installed
<input checked="" type="checkbox"/> Android 8.1 (Oreo)	27	3	Installed
<input checked="" type="checkbox"/> Android 8.0 (Oreo)	26	2	Installed
<input checked="" type="checkbox"/> Android 7.1.1 (Nougat)	25	3	Installed
<input checked="" type="checkbox"/> Android 7.0 (Nougat)	24	2	Installed
<input checked="" type="checkbox"/> Android 6.0 (Marshmallow)	23	3	Installed
<input checked="" type="checkbox"/> Android 5.1 (Lollipop)	22	2	Installed
<input checked="" type="checkbox"/> Android 5.0 (Lollipop)	21	2	Installed
<input checked="" type="checkbox"/> Android 4.4W (KitKat Wear)	20	2	Installed
<input checked="" type="checkbox"/> Android 4.4 (KitKat)	19	4	Installed
<input type="checkbox"/> Android 4.3 (Jelly Bean)	18	3	Not installed
<input type="checkbox"/> Android 4.2 (Jelly Bean)	17	3	Not installed
<input type="checkbox"/> Android 4.1 (Jelly Bean)	16	5	Not installed
<input type="checkbox"/> Android 4.0.3 (IceCreamSandwich)	15	5	Not installed
<input type="checkbox"/> Android 4.0 (IceCreamSandwich)	14	4	Not installed

Figure 4: Android SDK manager

JDK fût installé ensuite afin de compléter les fichiers nécessaires.

Pour la création du keystore débogué, la commande étant donnée sur la documentation officielle de Godot, je n'ai eu qu'à l'exécuter en ligne de commande :

```
keytool -keyalg RSA -genkeypair -alias androiddebugkey -keypass android
keystoredebug.keystore -storepass android -dname "CN=Android
Debug,O=Android,C=US" -validity 9999
```

<sup>4</sup> Java development Kit : ensemble de bibliothèques du langage Java.

<sup>5</sup> Android Debug Bridge : Outil en ligne de commande. Permet la communication avec les périphériques Android.

Cette commande a permis de créer un fichier « debug.keystore » que j'ai ensuite déplacé dans mon dossier d'installation Android pour plus de facilité.

Il ne me resta ensuite qu'à paramétrier Godot afin d'utiliser ces fichiers lors de mes exports.

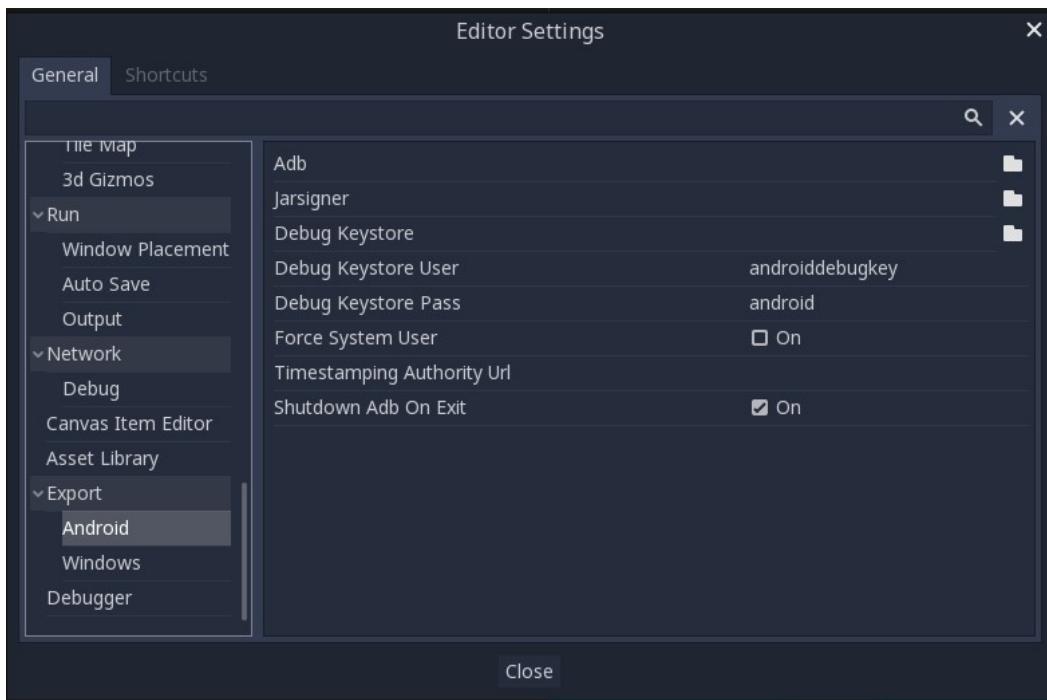


Figure 5: Configuration d'export Android pour Godot

Trois fichiers étaient à renseigner ici :

- L'exécutable ADB
- L'exécutable « jarsigner » du JDK
- Le fichier « keystore »

En surplus de tout cela, nous avons aussi créé par avance un second « keystore release» non débogué. Cette clé sera utilisée pour vérifier l'identité du développeur lors de sa mise en place sur le Play Store.

```
keytool -v -genkey -v -keystore mygame.keystore -alias mygame -keyalg RSA -validity 10000
```

Cette commande permettra de générer une clé et un mot de passe qui seront liés à un alias (-alias mygame ci-dessus).

Nous avons désormais un Moteur presque prêt à exporter une application vers Android. Il nous reste en effet une dernière étape à préparer afin de pouvoir tester notre application sur notre périphérique Android.

## 4.2. Export Android

Afin de réaliser un export à proprement parler, nous devons prévenir Godot des paramètres à utiliser.

Cela se fait par le biais de Template d'export Android dans lesquels nous pourrons définir certaines options. Les plus importantes pour la suite étant de bien vérifier que l'option « Runnable » soit bien cochée afin de pouvoir utiliser le template lors de nos tests pendant la période de développement.

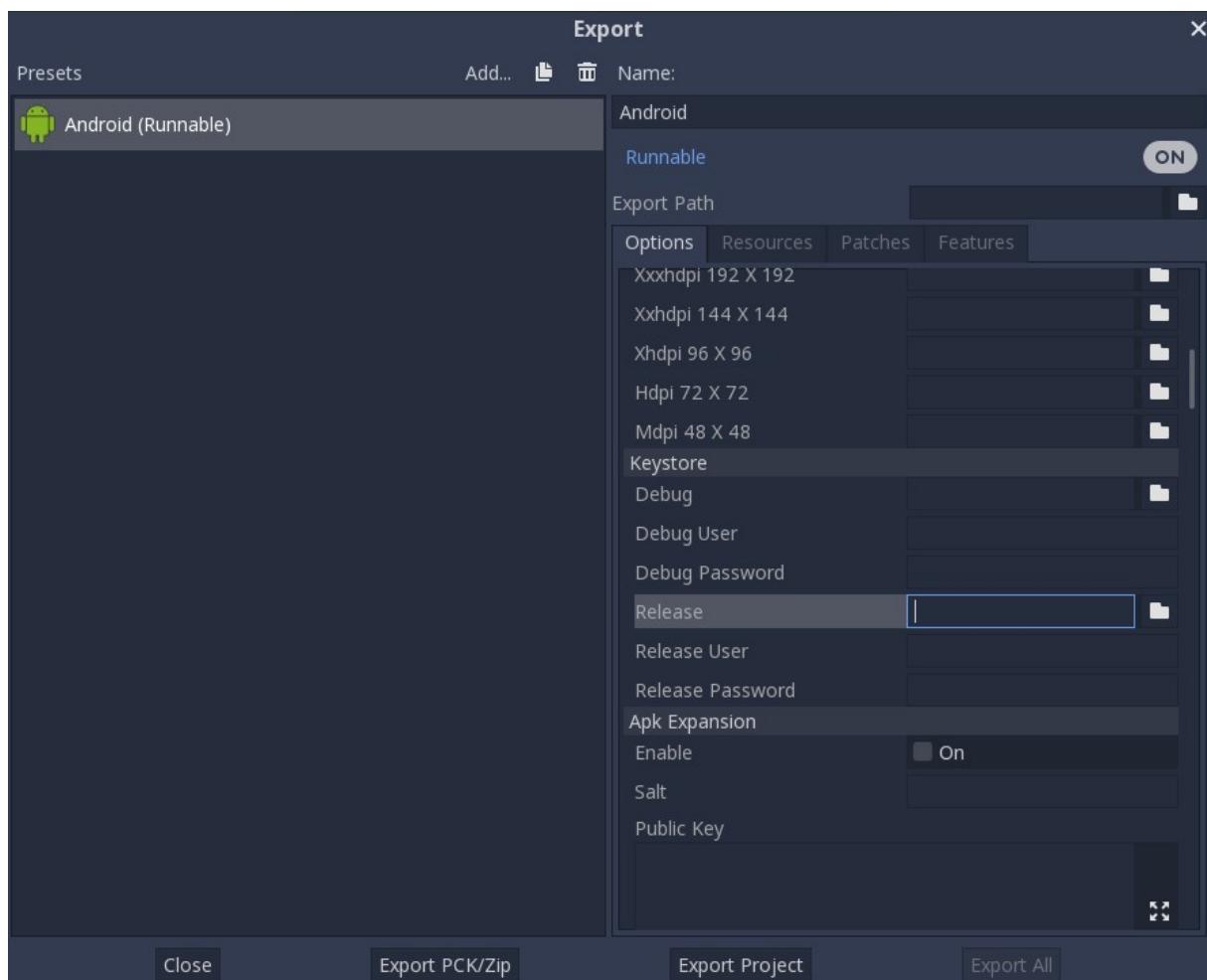


Figure 6: Template d'export Android

Il conviendra aussi de prendre note des champs suivants :

- Release : Chemin d'accès au Keystore release
- Release User : Alias de votre application (défini à la création de la keystore release)
- Release Password : Le mot de passe (fourni lors de la création de la keystore release)

## 5. Godot : De Nœuds et de scènes

L'architecture de projet sous Godot fonctionne avec un arbre fait de scènes et de nœuds. Chaque nœud est un élément hiérarchique de notre arbre et permettra l'organisation des éléments dans chaque scène.

Une scène ne définit ici pas seulement chaque « écran » d'une application ou d'un jeu mais définit aussi tout élément.

« Mais...ce ne sont plus des nœuds finalement? »

Si ! Et c'est ce qui fait la force et la versatilité de la méthodologie utilisée dans Godot.

### 5.1. Les scènes

Une scène dans Godot se définit comme étant un nœud (ou ensemble de nœuds) défini ayant certaines propriétés et pouvant être instancié dans une scène.

Pour exemple, voici le premier écran du projet en cours :

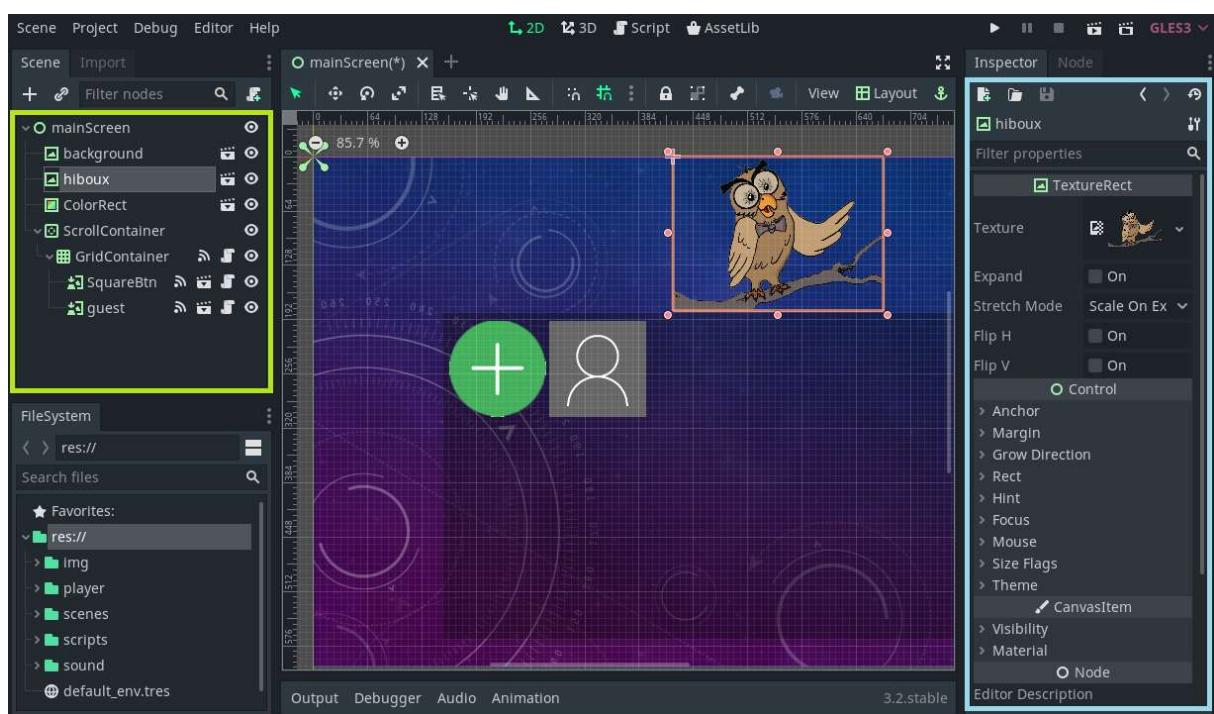


Figure 7: Écran principal du projet

Vous pouvez voir dans la figure 7, encadrée en vert, l'organisation de ma « scène » qui ici correspond à l'écran complet.

Chaque élément de l'arborescence est ici considéré comme un nœud de cette scène, nœud duquel vous pouvez changer les propriétés via « l'inspector » encadré en bleu.

Bien évidemment, chaque écran sera une « scène » mais chaque élément dans cette scène sera un nœud établi à partir de multiples scènes plus petites.

Par exemple, lorsque vous créez un bouton d'interface utilisateur pour un écran, vous pouvez le sauver en tant que scène. Ce bouton devient alors sa propre scène qui contient un nœud : le bouton. Vous pouvez ensuite ré-instancier cette scène dans n'importe quel écran ... ou scène.

Cela permet par exemple de créer un élément et de lui définir ses propriétés afin qu'il puisse rapidement être rajouté dans de nouvelles scènes au besoin. Non seulement cela, mais chaque instance d'une scène dans une autre est modifiable, ce qui permet là encore de modifier le comportement d'un élément sans en modifier sa scène d'origine, ce qui n'est pas sans rappeler les classes, l'héritage et le polymorphisme de la programmation orientée objet.

## 5.2. Les nœuds

Comme nous l'avons vu, chaque scène est en fait un objet manipulable et instanciable dans une architecture nommée également scène. La différence entre le concept de scène et de nœuds se fait alors par le biais de l'arborescence.

La bonne pratique ici est donc de faire de chaque nœud une scène à part entière. Une scène contiendra donc d'autres scènes qui seront les nœuds de celle-ci. Les nœuds ne sont donc valables que dans une scène donnée dans laquelle on instancie les scènes qui viendront se greffer sur l'arborescence en tant que nœud de celle-ci.

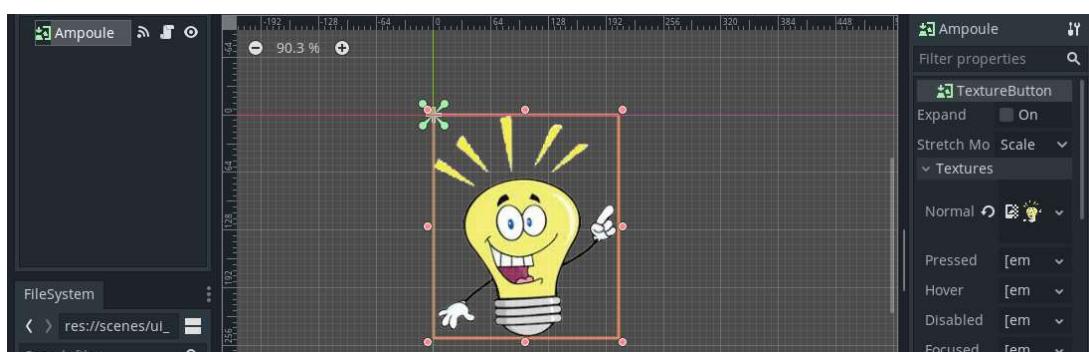


Figure 8: Scène d'un élément graphique d'interface

### 5.3. Bien choisir ses nœuds

Lors de l'ajout d'un nœud à une scène, il vous sera demandé de spécifier quel type de nœud vous désirez ajouter. Bien que la question soit triviale puisque vous pouvez modifier les éléments à loisir, il est de bon ton de bien les choisir afin d'assurer que les propriétés du nœud nouvellement créé soit bien celles attendues.

En effet, chaque type de nœud possédera un ensemble de propriétés qu'il héritera de sa catégorie et que vous pourrez modifier. Ainsi, un nœud « sprite2d » n'aura pas les mêmes méthodes de base qu'un nœud texture.

Il n'est pas impossible de se contenter du nœud « Node » (voir Figure 9) qui est le nœud le plus basique qui soit et d'en forcer les méthodes et attributs. Ce nœud sert de modèle à chaque nœud enfant qui héritera de ses caractéristiques de base.

Ils disposeront cependant de méthodes et attributs plus spécifiques en fonction de leurs usages mais incluront toujours au minimum les attributs de leurs nœuds « parents ». Ce qui ici encore n'est pas sans rappeler certaines caractéristiques propres à la programmation orientée objet.

Chaque nœud répondra donc à un usage spécifique en fonction de sa catégorie par ses méthodes héritées, mais il aura aussi certains types de signaux pré-attribués. Bien choisir les nœuds permet donc de gagner un temps parfois précieux sur le temps de développement.

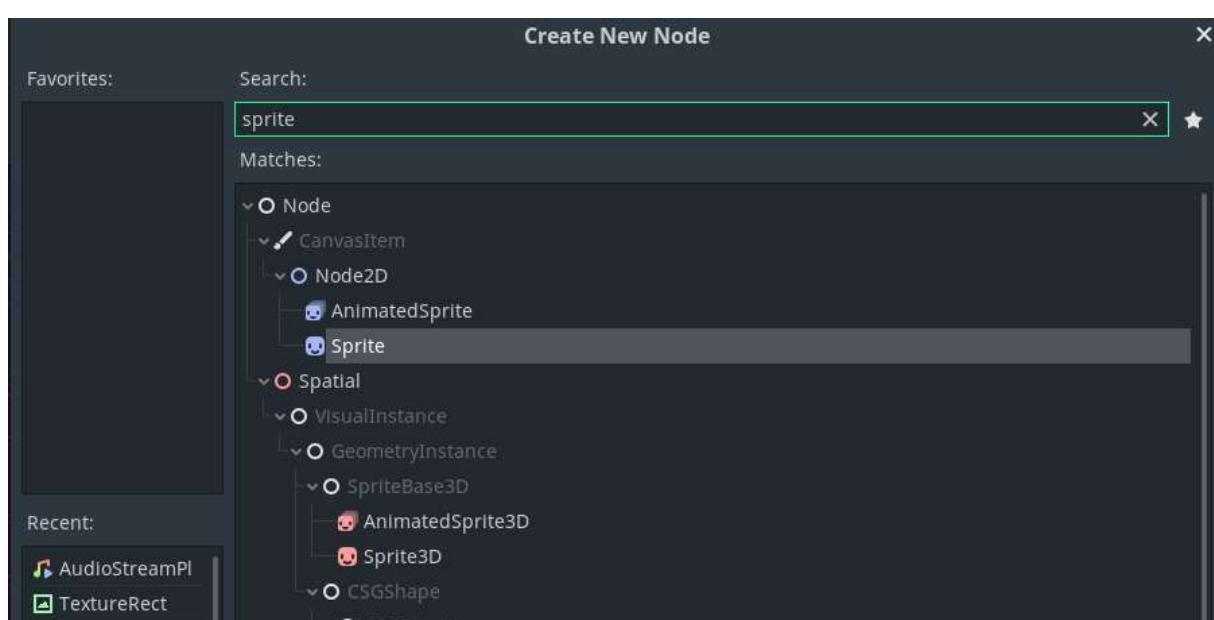


Figure 9: Création d'un nœud

## 5.4. Scripts et Signaux

Jusqu'à présent nous n'avons fait que survoler le moteur. Nous sommes restés confinés à l'interface graphique et à l'usage visuel de notre moteur. Mais Godot offre bien plus de possibilités via son langage de script maison : GDScript.

Ce dernier est fort similaire à Python et repose donc sur sa syntaxe d'espacement en début de ligne afin de déterminer les différentes sections du code comme les fonctions et les boucles. Le choix est cependant laissé à l'utilisateur de réaliser cet espacement via tabulation ou par espaces simples. Le mix des deux n'est cependant pas supporté.

Nous avons vu que chaque nœud est limité à sa propre catégorie et ne permet de paramétrier qu'un nombre précis d'attributs en fonction de sa catégorie. Tous les attributs ainsi que les méthodes de l'objet lui-même peuvent être modifiés par utilisation de script attaché aux objets et déclenché par des signaux.

La première étape est de créer un script dans nos ressources de projet et de le lier à un élément de notre scène. Il est tout à fait possible de l'ajouter en tant qu'orphelin (non lié à un nœud) ou directement de l'ajouter via un clic droit à notre élément en choisissant l'option « Attach script », ce qui créera un script dont vous pouvez choisir le nom et l'emplacement dans la hiérarchie des ressources du projet.



Figure 10: Signaux

La seconde étape sera de définir un signal (ou plusieurs) lié au script et qui permettra l'exécution du code contenu dans le script. Vous pouvez observer sur la Figure 10 que l'élément que nous avons sélectionné regroupe certains signaux hérités des catégories auxquelles il appartient.

On retrouve par exemple les signaux de boutons de base et nous pouvons également voir que nous utilisons le signal « pressed() » pour déclencher une action sur l'élément sélectionné.

Par défaut, tout script créé contiendra une seule fonction « ready » ainsi que quelques commentaires. L'ajout d'un signal ajoutera à cela sa propre méthode en fonction du signal utilisé afin que l'utilisateur puisse ensuite procéder à la rédaction de son propre code.

Nous constaterons d'ailleurs sur la Figure 11 ci-dessous que chaque fonction ainsi ajoutée au code comportera un symbole afin de vérifier qu'elle est bien liée à un signal permettant son exécution. Auquel cas et selon l'utilisation, le code ne sera tout simplement pas exécuté par l'application.

```

1  extends TextureButton
2
3  # Called when the node enters the scene tree for the first time.
4  func _ready():
5      pass # Replace with function body.
6
7  func _on_TextureButton_pressed():
8      pass # Replace with function b|ody.

```

Figure 11: Ajout de signal en script

Ces signaux ne sont cependant pas limités à ceux de la catégorie de l'élément. Il est tout à fait possible de créer son propre signal et d'en définir le comportement afin d'outrepasser les limitations du système en place.

Les signaux fournis sont cependant suffisamment bien pensés que pour couvrir tous les besoins nécessaires pour un large panel d'utilisations classiques. Et bien que le langage soit ici spécifique à Godot, sa similarité à Python en fait un outil très souple à utiliser.

En outre la documentation « in engine » et en ligne regroupe de manière extensive toutes les fonctions, signaux, nœuds et plus encore ainsi que des explications sur leurs utilisations.

## 5.5. Les ressources et Android

Il est très important de noter que Godot a une manière spécifique de traiter les ressources ce qui peut faire varier le résultat en fonction de la plateforme cible. Bien que sur un système classique Windows cela ne pose pas de problèmes, cette gestion fût à l'origine de bien des maux que j'ai eu l'occasion de rencontrer lors de ce projet.

Lors de l'ajout de ressources au projet, Godot va lui-même effectuer un traitement du fichier et en générer un fichier « .import » en addition du fichier ajouté. Ce fichier contiendra en fait une référence vers un dossier de ressources géré par Godot et se trouvant à la racine de votre projet.

Ce dossier contiendra des fichiers de métadonnées qui permettront à Godot de mieux gérer vos ressources. Concrètement, cela permet à Godot d'alléger l'export en n'incluant que les fichiers spécifiés de manière dure dans le projet.

```
var path = ("res://img/"+file)|| #Lien dynamique  
var path = ("res://img/MonImage")|| #Lien Dur
```

Figure 12: Déclaration des ressources

L'utilisation de l'application dans le moteur ainsi que sur système Windows ne posera pas de soucis. Néanmoins, Android ne procédant pas de la même manière, toute ressource variable du projet ne sera pas ajoutée au projet.

Afin de mieux gérer ces ressources, il convient donc lors de l'écriture du script de référencer non pas la ressource elle-même, mais son équivalent « .import » afin qu'Android utilise les fichiers compilés par Godot.

L'exemple de liens dynamiques repris dans la figure 12 ne fonctionnera donc pas sur Android. Là où l'exemple repris dans la Figure 13 fonctionnera lui tant sur Android que sur tout autre support.

```
var path = ("res://img/"+file+".import")|| #Lien dynamique corriger  
var path = ("res://img/MonImage")|| #Lien Dur
```

Figure 13: Ressource correcte pour Android

## 6. Réalisation du projet

Afin de réaliser le projet, je me suis entretenu avec Mr Vandenberghe pour établir les besoins auxquels devait répondre l'application.

En outre, Mr Vandenberghe a également fourni les ressources utilisées actuellement dans le projet. Ces ressources temporaires seront à terme remplacées par des illustrations réalisées spécifiquement pour le projet.

J'ai donc pu me concentrer exclusivement sur la réalisation du projet avec pour objet d'inclure la possibilité de pouvoir changer les ressources de manière simple et souple au fur et à mesure de son évolution.

Le projet en lui-même est utilisé comme projet dans le cadre de mes études, mais sera poursuivi par la suite. Je me devais donc de faire en sorte de pouvoir ajouter tout types de « jeux » en plus des deux jeux choisis comme base pour le projet.

Non seulement cela, mais le but de l'application étant de vérifier son efficacité dans l'aide à l'apprentissage du vocabulaire auprès des enfants, je devais pouvoir ajouter de nouvelles séries de mot afin de satisfaire les besoins des différents professeurs utilisant l'application.

Avant toute chose, il me fallut cependant établir un mockup. Cette étape m'a permis d'établir de manière concrète un point de départ visuel sur lequel je viendrais par la suite ajouter mes propres fonctions pour répondre le plus spécifiquement possible aux besoins émis par Mr Vandenberghe.

### 6.1. *Structure et Mockup*

Pour réaliser un mockup, rien de tel qu'un crayon et un bloc de feuille. Cela peut paraître un peu désuet, mais le besoin d'un mockup étant spécifiquement visuel, j'ai personnellement beaucoup plus facile de cette manière. Les idées fusant dans mon esprit, je peux rapidement les poser à plat telles quelles afin de les retravailler rapidement sans devoir me familiariser avec l'un ou l'autre service/logiciel de mockup spécialisé et qui n'inclurait pas forcément les fonctionnalités que je désire. Le plus facile reste donc de dessiner moi-même le mockup.

Ayant discuté avec Mr Vandenberghe, j'avais déjà établi le nombre d'écrans ainsi que leur utilité. Il me fallait dans un premier temps respecter l'ordre d'apparition des écrans.

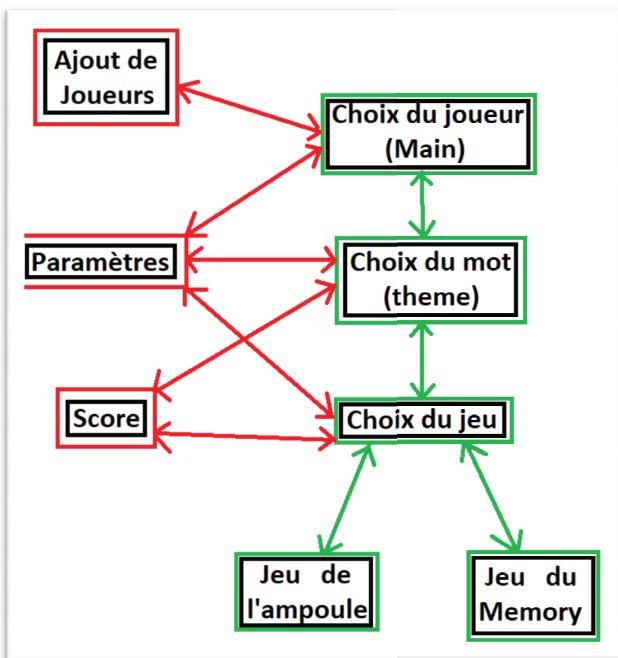


Figure 14: Structure de l'application

Une zone de paramètres était initialement prévue mais est désormais sujette à disparition, tout comme un bouton pour quitter l'application. Les détails concernant ces choix seront repris dans le chapitre 6.5 dédié au changement de design de l'application durant son évolution.

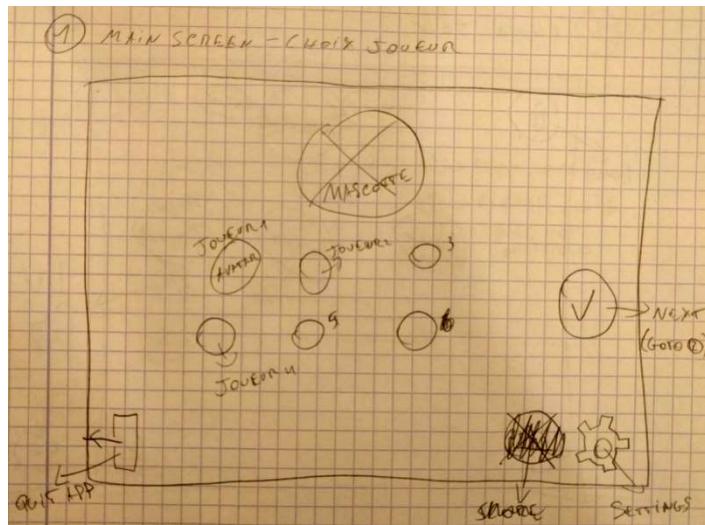


Figure 15: Mockup de l'écran principal

Vous pourrez retrouver le Mockup d'origine complet en annexe de ce document ou encore dans le cahier des charges du projet.

<sup>6</sup> Les éléments en vert représentent les écrans implémentés et fonctionnels de l'application. Ceux en rouge représentent les éléments en cours d'implémentation.

<sup>7</sup> Cette Figure fût réalisée après le mockup original.

## 6.2. Les écrans de jeux

Chaque écran a donc été pensé avec certains objectifs en tête. Ils devront donc fournir des fonctionnalités spécifiques à leurs utilisations. Nous avons vu que l'écran principal devra permettre non seulement d'accéder à l'écran d'ajout de joueurs, mais aussi aux paramètres ainsi qu'au choix du thème une fois le joueur sélectionné.

Une fois le joueur choisi, il n'est plus pertinent de proposer l'ajout de joueurs. Nous n'établirons donc pas de lien entre ces deux écrans. En revanche, il fût pertinent de fournir au joueur actif l'accès à sa page de score afin qu'il puisse consulter ses statistiques sur les jeux de l'application. Ce qui peut aussi s'appliquer à l'écran du choix de jeu.

La procédure décrite ci-dessus ne détermine cependant que les liens de navigation à faire entre les différents écrans de l'application. Hors nous avons vu plus tôt dans ce document que pour ajouter des éléments à notre projet, il est préférable d'en créer des scènes, puis de les ajouter en tant que noeuds dans notre arborescence de projet.

J'ai donc dressé une liste des divers éléments d'interface communs entre nos différents écrans. Chacun de ces éléments fut ensuite créé en tant que scène indépendante du reste. Les paramètres communs et constants de ces éléments furent appliqués afin de faciliter leur mise en place dans le projet.

## 6.3. Les éléments fixes : Merge, Branch et instantiation

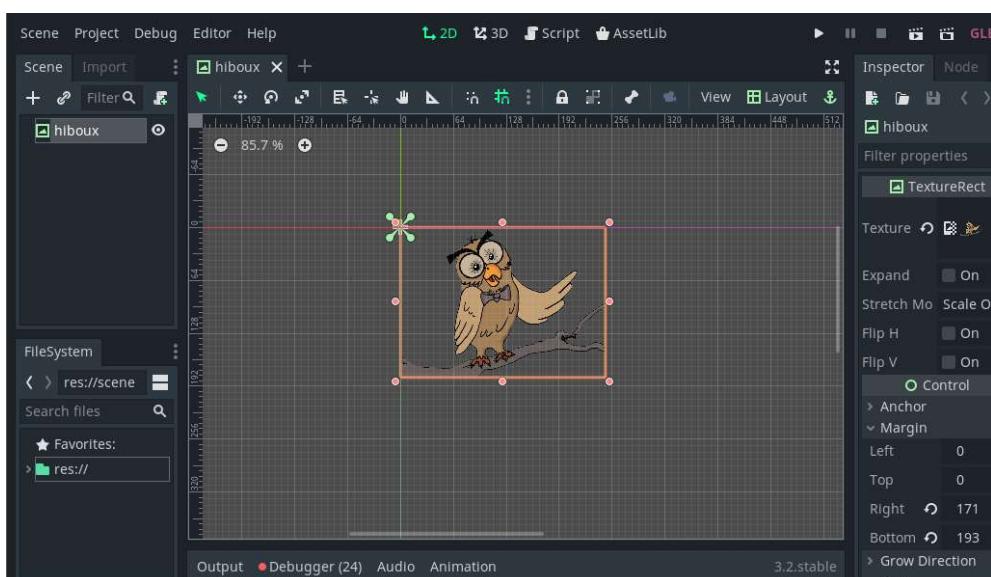


Figure 16: Mr Hibou, la mascotte de l'application

Les éléments invariables entre les scènes n'auront besoin que de peu de traitement. Une fois créés et paramétrés nous pouvons les instancier tels quels dans nos divers écrans. Si d'aventure nous avions besoin que le comportement de l'élément change, nous pourrions tout aussi bien là encore l'instancier tel quel et en modifier les paramètres une fois placé dans notre scène globale. Cela ne modifierait en rien sa scène « modèle » de laquelle nous l'avons instanciée.

De plus, nous pouvons prendre chaque nœud de notre arborescence pour en faire sa propre scène afin d'assurer de pouvoir réutiliser l'élément sans devoir repasser par sa création.

Une fois notre écran créé et lorsque nous créons le suivant, nous pouvons également fusionner certains éléments d'un écran dans un autre via l'option « Merge from scene »

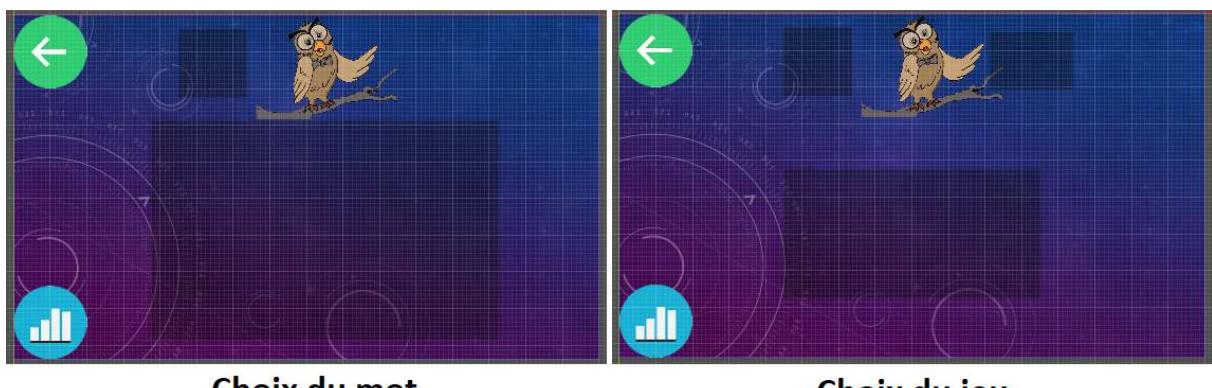


Figure 17: Exemple de merge entre deux scènes

Vous pouvez constater sur la Figure 16 reprise ci-dessus qu'il existe peu de différences entre l'écran choix du mot et l'écran choix du jeu.

Il n'existe réellement qu'un élément supplémentaire dans l'écran de choix du jeu. Les autres étant simplement des merges de composants existants et déjà utilisés dans la scène précédente.

La différence entre le merge et l'instanciation étant que l'instanciation rajoute le composant tel qu'il est spécifié dans son model. Étant donné que nous changeons certains des paramètres du nœud lors de l'ajout dans nos scènes, comme la position en x et y, nous pouvons merge les nœuds afin de ne pas avoir à les redéployer nous-mêmes visuellement à leur place pour correspondre à l'écran précédent.

Dans le cas présent, nous pouvons observer que j'ai simplement procédé à une mise à l'échelle de la zone d'affichage des éléments dynamiques de la scène. Cette zone grisée au

centre étant destinée à accueillir les différentes icônes des mots ou de jeux disponibles, il convenait de les mettre toutes deux à échelle en fonction du nombre d'éléments à y faire apparaître.

Nous avons donc établi désormais une base d'éléments fixes que nous pouvons réutiliser à souhait dans toutes nos scènes. Chacun de nos nœuds sera donc une instantiation de scènes disposant d'attributs prédéfinis pouvant être modifiés après ajout dans chaque écran.

Nous pourrions tout à fait procéder de la même manière pour l'entièreté de l'application. Néanmoins cela nuirait à son maintien sur le long terme. D'autres jeux ou séries de mots viendront par la suite peupler l'application. Le choix a donc été fait de définir ces éléments comme étant variables et requérant un soin particulier par rapport au reste de nos nœuds.

#### **6.4. La souplesse de GDScript et linstanciation procédurale**

Notre écran principal, l'écran de choix de mots, de jeux ou même nos deux jeux vont tous contenir certains éléments qu'il nous est impossible de définir à l'avance. Que ça soit donc la liste de joueurs, la liste de mots ou de jeux ou les images à utiliser lors des jeux, nous devons faire en sorte qu'elles soient chargées de manière dynamique par l'application.

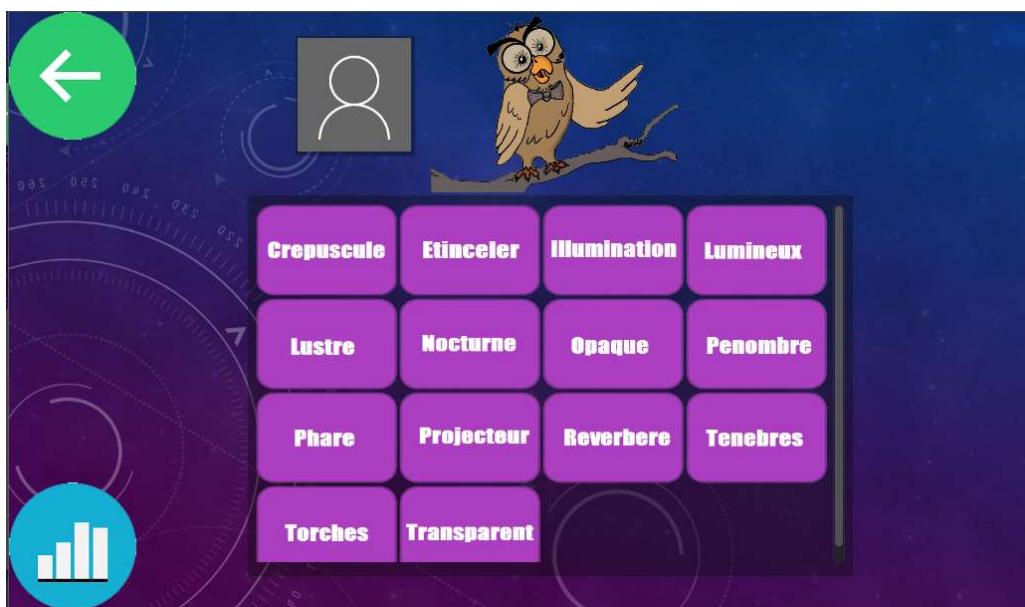


Figure 18: Liste de séries de mots générée dynamiquement

L'idée de base étant que pour chaque joueur, chaque série de mot ou chaque jeu, une image existe. Nous pourrions donc parcourir les dossiers correspondants afin de les ajouter dans

nos écrans. Ainsi, la génération de ces éléments d'interface serait basée sur les fichiers réels présents dans notre application compilée.

Cette approche a notamment permis de préparer non seulement nos menus dynamiques, mais également la procédure qui sera utilisée pour nos jeux qui devront quant à eux rajouter une sélection aléatoire de certains éléments afin de varier les réponses.

C'est à partir de cette étape que nous parlerons plus en détails de GDScript. Car si les options de personnalisation de nos nœuds sont limitées par leurs catégories comme nous avons pu le voir, il n'en est plus rien lorsque l'on parle du système de script. Si vous pouvez l'imaginer, vous pouvez le coder.

Mon objectif a été d'établir du code que je pourrais transporter d'un élément à un autre. J'ai donc commencé par charger une liste de joueurs. Pour ce faire j'ai préparé un container en grille qui servirait à accueillir mes icônes une fois générés. Je me suis également assuré qu'elle puisse devenir scrollable en cas de nombre important de joueurs.

J'ai ensuite appliqué un script ainsi qu'un signal à cette grille. Ce faisant, je suis parti du principe suivant :

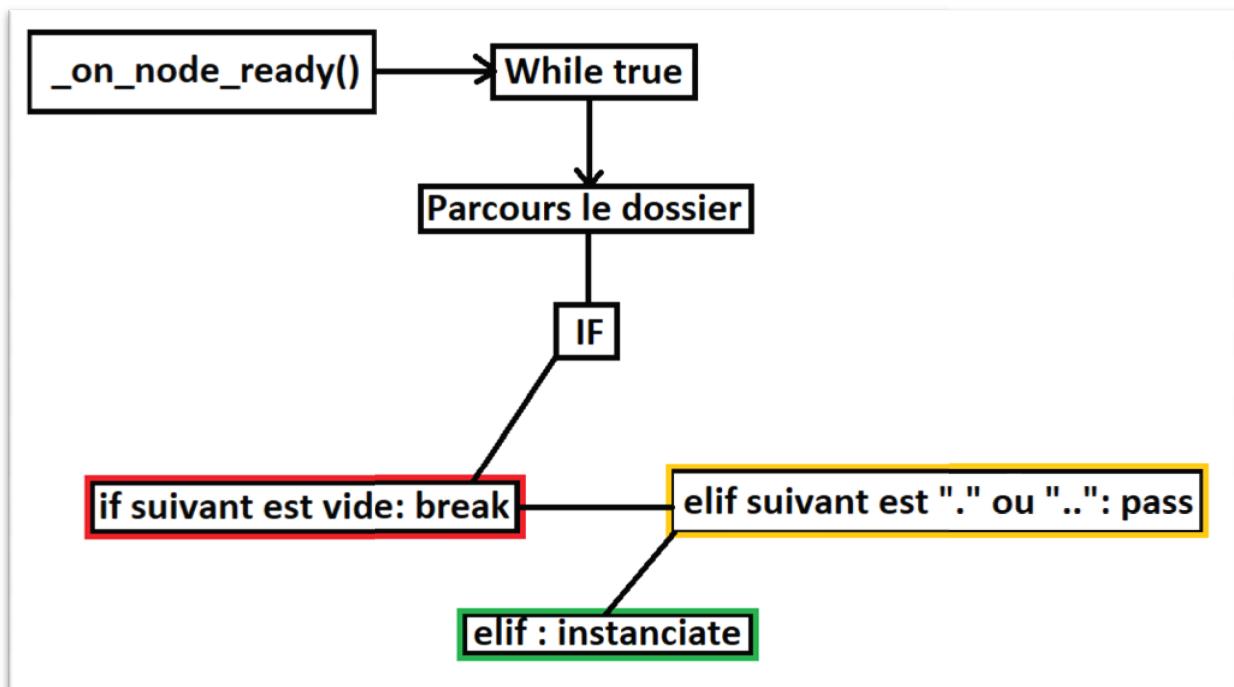


Figure 19: Instantiation dynamique

Ma grille ferait donc la vérification des images dans le dossier. Étant donné l'utilisation d'une boucle while, une condition de sortie était impérative afin d'éviter tout soucis lors de

l'exécution. Il me semblait donc naturel de proposer un break en premier lieu si la boucle avait terminé de parcourir le dossier.

Il me fallut ensuite éviter les éléments indésirables contenus dans le dossier. En plus des images, chaque dossier contient toujours au minimum deux éléments cachés sous la forme du « . » et « .. » représentant respectivement le répertoire courant et le répertoire parent.

Si ces conditions étaient respectées, il ne me resterait plus qu'à initier l'instanciation de mes nouveaux nœuds. A cette fin spécifique, j'ai créé une version « vide » des futures icônes et qui me serviraient de modèle. Je n'eus alors plus qu'à ajouter cette icône vide comme enfant de ma grille et à en modifier ces paramètres au vol lors de sa création comme son nom, son image servant de texture ou encore son script.

Chaque nouvel élément créé aurait donc un nom différent qui serait basé sur le nom du fichier. Leurs textures seraient définies par l'élément que nous rencontrons justement lors du parcours dans notre boucle. Le script quant à lui est un script commun à tous nos éléments.

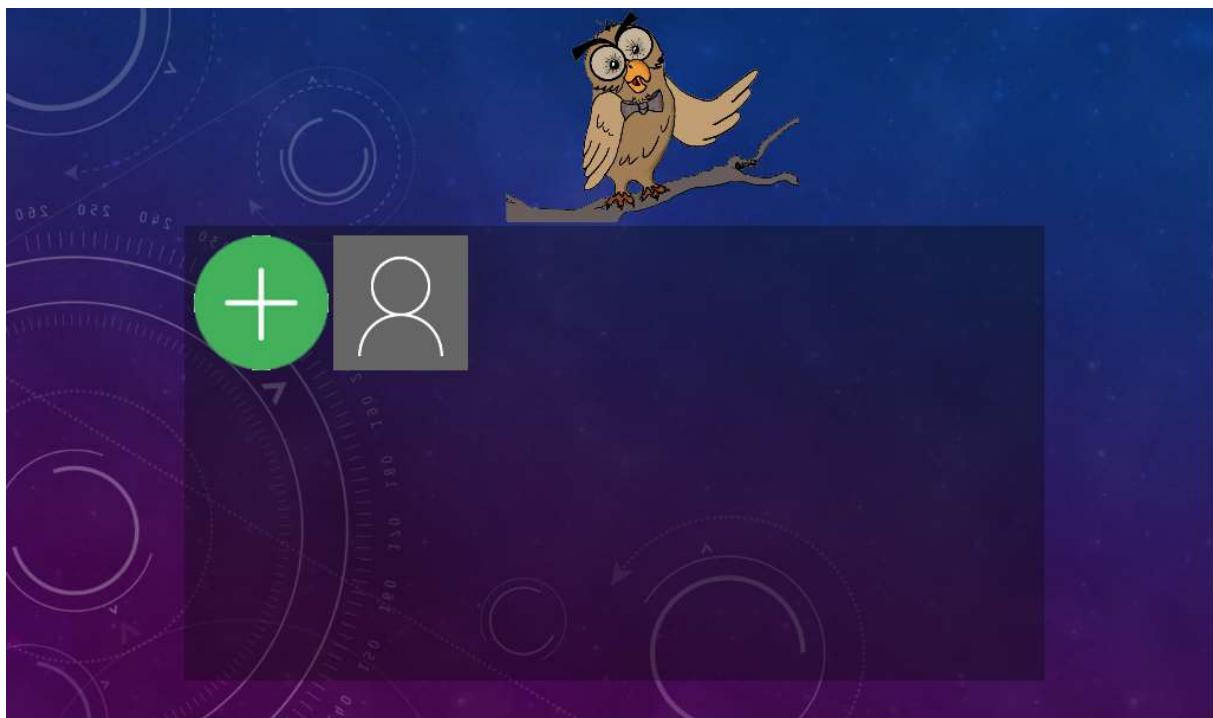


Figure 20: L'écran choix du joueur

De plus l'utilisation d'un script « global » spécifique permet à Godot de garder des informations en mémoire entre les scènes. Cela nous permet par exemple de conserver en

mémoire le nom du joueur sélectionné, son choix de mot et de jeu ainsi que son score que nous pouvons dès lors charger à partir de ses informations de joueurs, les modifier, puis les réécrire.

Ces informations seront toujours chargées en mémoire et nous pourrons faire appel au script ainsi qu'à ses composantes depuis n'importe quel autre endroit de l'application. L'énorme avantage est que ce script officie comme un « Singleton<sup>8</sup> » pour Godot. Rendant la conception de projet encore plus souple et optimisée.

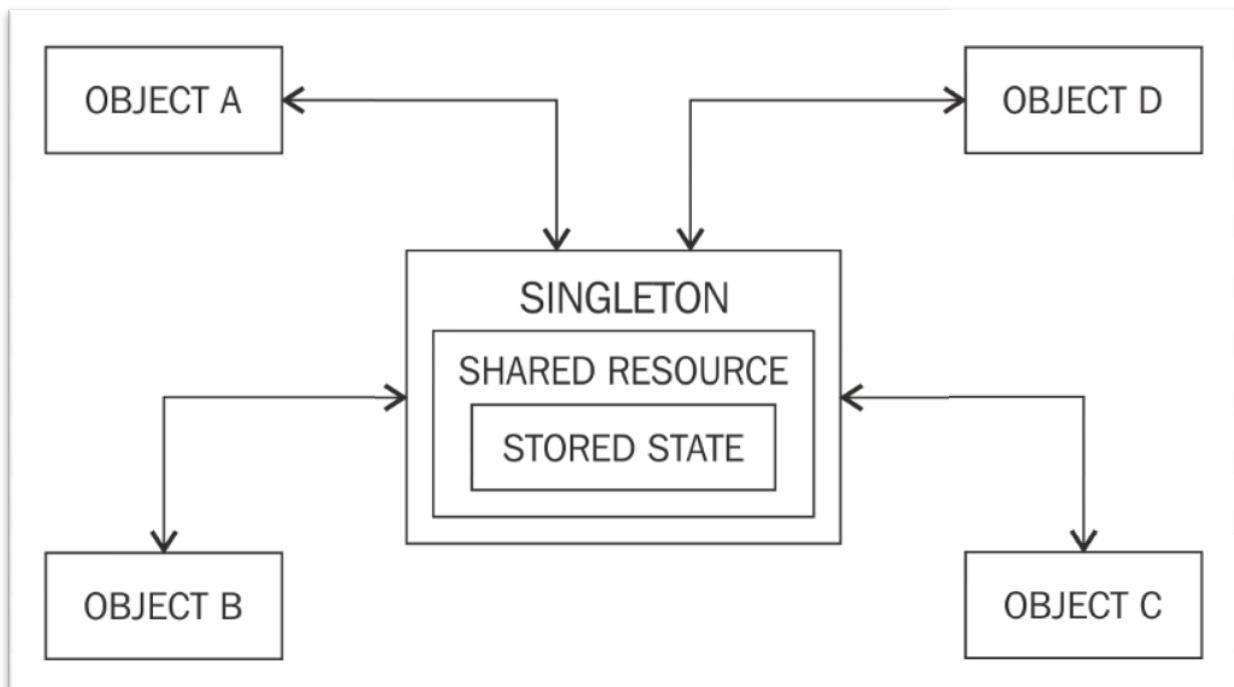


Figure 21: Schéma du Design Pattern Singleton

A ce stade du projet, j'ai réalisé que certains éléments étaient de trop. Des options comme le bouton pour quitter l'application ou encore les paramètres furent coupés du projet naturellement. Le placement de certains icônes ont également connu quelques changements.

Le but était de garder la pertinence des options, en offrant un épurement de fonctions superflues. Ces changements ce sont donc imposés d'eux-mêmes spontanément comme faisant partie de l'évolution naturelle du projet.

---

<sup>8</sup> Le singleton est un pattern de programmation dont l'objectif est de limiter l'instanciation d'une classe à un seul objet.

## 6.5. **Changement de design**

À commencer bien entendu par l'écran principal, et en contradiction avec mon mockup d'origine, je n'ai finalement jamais inclus de bouton « Suivant ». L'événement pouvait être déclenché dès lors que le joueur choisit son Avatar dans la liste. L'utilité du bouton étant de ce fait caduque, je n'avais plus vraiment de raison de l'inclure à cet endroit.

Il était également prévu au départ d'avoir un bouton pour quitter l'application. Je me suis vite rendu compte que l'application étant destinée à Android, la méthode la plus probable pour fermer l'application serait via la barre de navigation d'Android. Ici encore, l'utilité de l'icône étant remise en question et ne trouvant pas d'argument logique à son maintien, le choix a été fait d'épurer ce composant.

L'icône des paramètres a quant à elle subit un trajet plus complexe que les autres. Au départ, son utilité était de fournir à l'utilisateur un moyen de gérer les joueurs, mais aussi les séries de mots et de jeux.

Dans le cas des joueurs, cela se fera désormais via l'icône ajout de joueurs sur l'écran principale. L'ajout de séries de mots et de jeux étant désormais géré de manière différente de celle prévue initialement et ne requérant pas d'options dans l'application, le bouton paramètre était désormais désuet.

De plus, ce bouton étant initialement présent sur plusieurs écrans, son ablation permettrait d'alléger la présentation générale de l'application. Cela permettrait également de repenser d'autres icônes et d'utiliser l'espace à meilleur escient.

Peut-être le plus gros changement de design fût un ajout de ma part sur les écrans de choix de mots et de jeux. Pris dans la spirale créative du moment, j'ai eu l'idée de créer deux zones de part et d'autre de la mascotte de l'application, Mr Hibou.

L'une permettrait de conserver l'avatar du joueur sur l'écran tandis que l'autre servirait à voir le mot qu'il a choisi.

Pratique visuellement, cela m'a également permis de mettre à profit le Singleton. Lors du choix du joueur, le Singleton permet de stocker dans le script global les choix du joueur, puis d'afficher les données dans ces zones.

Cela s'est d'ailleurs avéré encore plus utile lorsqu'il a fallu passer au développement des écrans de jeux à proprement parler.

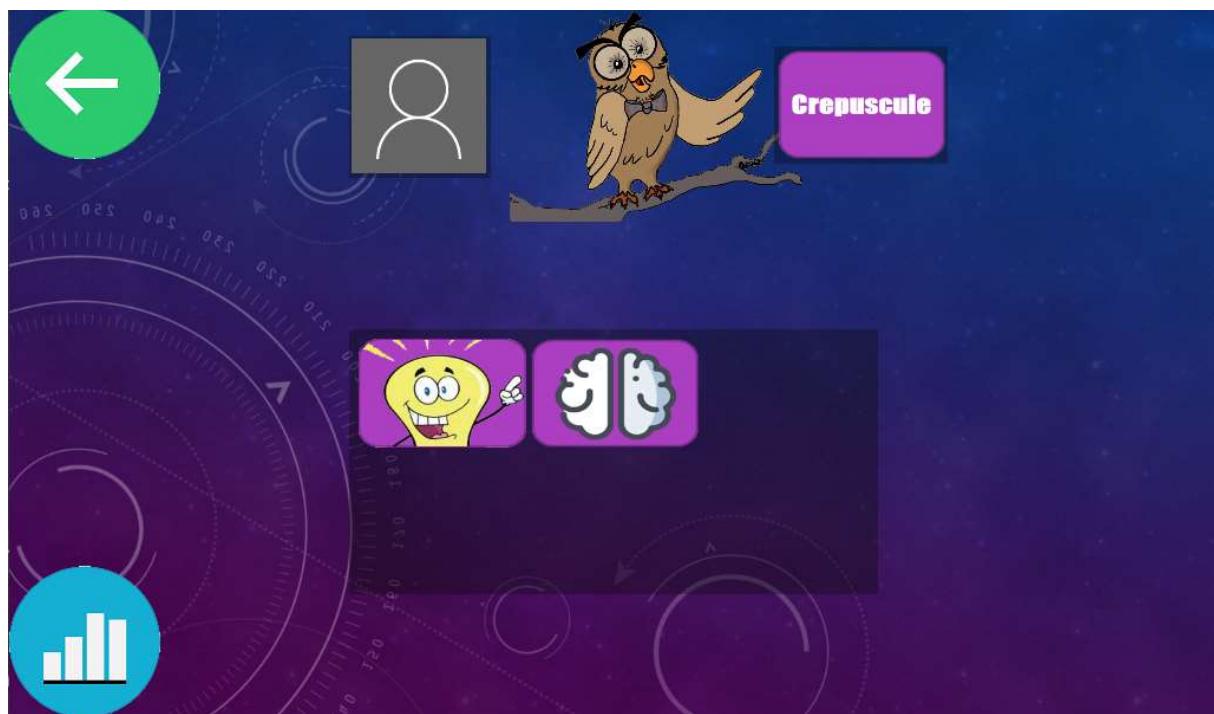


Figure 22: Changement de design bénéfique

D'autres changements mineurs ont aussi eu lieu, tel que le déplacement de certains icônes dans les écrans de jeux, l'ajout d'un icône « refresh » lors de ceux-ci afin de rafraîchir la série sans pour autant quitter le jeu et l'ablation de l'icône « home » devenu désuet par la force des choses ainsi que le design inhérent de l'application.

Bien qu'il restait alors certaines adaptations à faire sur mes écrans de sélection, les fonctionnalités étaient en place et l'application prenait forme. Il était désormais temps de valider le fonctionnement des jeux. Pour ce faire j'ai choisi de commencer par le jeu « phare » de l'application : L'ampoule.

## 6.6. L'ampoule

Le concept de jeu est simple : Six images sont montrées au joueur. Ce dernier, en fonction du mot qu'il a sélectionné, doit choisir les bonnes réponses.

Lorsqu'il clique sur une image, si celle-ci est correcte, un V vert se rajoute sur l'image et un son est joué en récompense. Si le joueur clique sur une mauvaise image, un son d'erreur est joué et une croix rouge apparaît sur l'image en question.

Afin de réaliser ce jeu, il me fallait donc les images correctes correspondant au mot, mais également de mauvaises images servant d'erreur.

Les ressources étant fournies par le promoteur, j'entrepris de les intégrer au projet. Cependant pour les images d'erreurs et afin d'optimiser l'utilisation des ressources, il n'était pas obligatoire d'en inclure dans le projet.

Car en effet, si le joueur choisit un mot spécifique pour jouer, cela implique également que toute autre image provenant d'un autre dossier de mots contient donc des images erronées pour le mot choisi. Il ne me restait donc plus qu'à déterminer un dossier au hasard pouvant servir de réserve d'images d'erreur pour la série en cours.

J'ai donc pu déterminer le dossier d'erreurs facilement, il m'a suffi pour cela d'émettre comme condition que le dossier choisi aléatoirement par ma fonction devait impérativement être différent de celui choisi pour la série de mot.

Pour l'algorithme du jeu, nous utiliserons ici encore le signal « ready() ». Nous allons cependant appeler d'autres fonctions lors de son exécution afin de faciliter le développement et l'ajout de nouvelles fonctions dépendant de nos besoins.

```

▼ func _on_ampoulecontainer_ready():
  ▷ dircount(subd) #Count how many element there are in a folder
  ▷ errordir() #Choose a dir for wrong answer at random
  ▷ randadd() #Will add an answer at random

```

Figure 23: Structure du code du jeu de l'Ampoule

Chaque fonction est ici définie sous la fonction principale et appelée au besoin. Afin de déterminer les éléments à sélectionner ainsi que le dossier d'erreurs, nous comptons d'abord les éléments de nos dossiers, desquels nous soustrayons les éléments indésirables comme le « . » et « .. ».

Nous choisissons ensuite un dossier d'erreurs et bouclons dans cette fonction tant que notre choix n'est pas valable. Après quoi, nous recomptons de nouveau le contenu du dossier afin d'en sélectionner des éléments aléatoirement.

Notre dernière fonction est ici particulière dans le sens où nous stockons nos réponses choisies dans un « array<sup>9</sup> ».

Les réponses stockées dans l'array sont cependant prévisibles, les trois premières étant les bonnes réponses et les trois suivantes les mauvaises réponses. La fonction utilisée nous permet donc de choisir un élément de l'array de manière aléatoire et lancera elle-même une sous-fonction d'ajout qui s'occupera de récupérer les informations nécessaires avant d'appeler elle-même une autre sous-fonction d'instanciation.

```

  ↘ func randadd(): #Will add an answer at random
    >| add()
  ↘ func add():
    >| instantiate()
  ↘ func instantiate(): #Will instantiate the element in the scene
  
```

Figure 24: Appel des sous-fonctions de l'ampoule

Une fois mes éléments instanciés correctement, j'entrepris de préparer les scripts à appliquer à ces derniers.

Puisque chaque réponse aurait une exécution lorsqu'elle serait sélectionnée, un script leur a été ajouté. Ce script nous permit de déclencher les événements de choix de réponse.

Le script vérifie d'abord quelle réponse est choisie et dans le cas d'une correspondance avec le mot de la série, récompense le joueur en adaptant la texture de son choix avec un V vert avant de jouer le son de récompense.

Le script fera de même pour les mauvaises réponses, à la différence qu'il substituera le V vert avec la croix rouge et la récompense auditive avec le son d'erreur.

Il existe sûrement d'autres méthodes plus optimisées pour arriver au résultat escompté. Beaucoup de fonctions de code existent et certaines opérations que j'effectue ici moi-même en créant mes propres fonctions existent déjà dans Godot.

Je découvre cependant le moteur au fur et à mesure du projet, c'est pourquoi certains aspects de ce dernier peuvent sembler peu polis pour les yeux avertis. Cela se verra encore lors de la mise en place du jeu suivant, le memory qui, même s'il applique certains principes

---

<sup>9</sup> Un array est un tableau de structure de données permettant le stockage de plusieurs variables en un seul élément.

déjà vu, offre pas mal de différences par rapport aux autres scènes que nous avons réalisées jusqu'à présent.

## 6.7. Le memory

Le memory consiste en une série de réponses face cachée et pouvant être retournées afin de voir à quelle image elle correspond. Il faut ensuite trouver l'autre image correspondante dans la série, toujours sans les voir.

En cas de mauvaise réponse : les images retournent face cachée. En cas de bonne réponse, ces images restent découvertes et sont désactivées pour le reste de la partie. Le but est de reformer les paires d'images jusqu'à complétion de la série.

Il s'agit donc encore une fois de déterminer les images à générer dans une grille donnée afin de servir comme élément du jeu.

Chaque image est ainsi générée avec comme attribut cette fois une image constante servant de verso des images. L'image réelle étant elle attribuée au recto, lors de la sélection, nous inversons ce recto et ce verso afin de laisser l'image à l'écran.

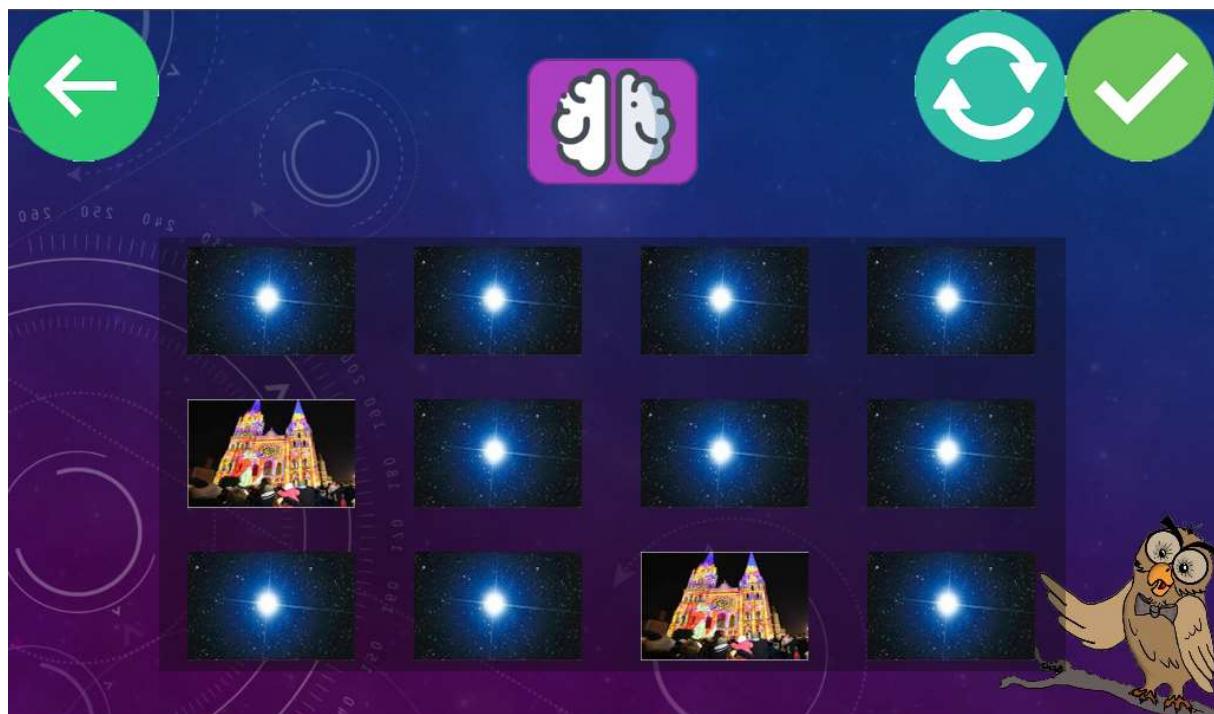


Figure 25: Le jeu du memory

Étant donné que notre script ne se lance que lorsque l'on sélectionne une image et qu'il est lié à cette image, le script qui se lancera à la sélection de la deuxième sera différent puisque lié à un autre objet de notre scène. Il nous fallait donc une solution pour conserver en variable le choix du joueur fait précédemment.

Nous utilisons la force du singleton à cet effet. Il nous permet de stocker la première image choisie afin de pouvoir la comparer à la seconde pour vérifier si le joueur a su faire une paire ou non. En cas de paires, nous laissons simplement les images à découvert et jouons le son de récompense. En cas de faute, chaque image est retournée afin de les dissimuler à nouveau.

## 6.8. La gestion d'utilisateur

La gestion d'utilisateur devra permettre au professeur utilisant l'application dans le cadre de leurs recherches d'ajouter des joueurs représentés par les enfants qui joueront au jeu de l'application.

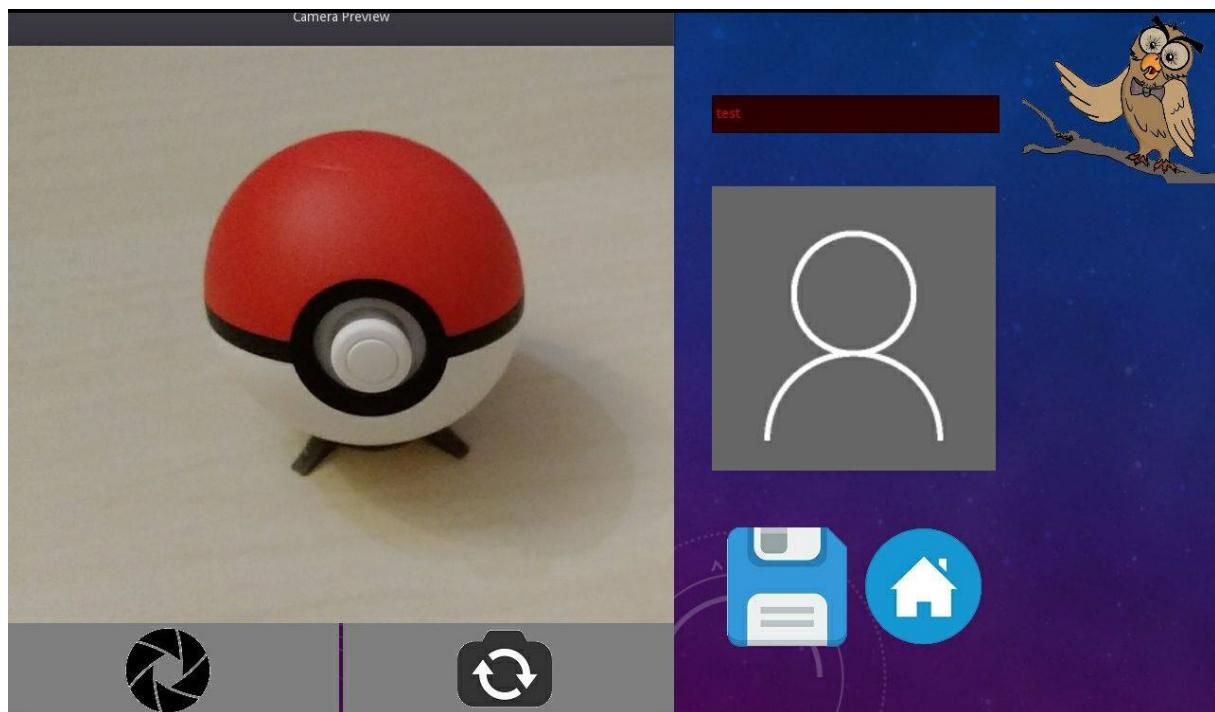


Figure 26: Création de joueurs

A ces fins, le processus de création permettra de choisir le nom du joueur ainsi que de prendre une photo via la caméra du périphérique utilisé.

Nous utilisons donc un écran dédié qui nous permet de visualiser le feed de la caméra, mais aussi d'avoir une prévisualisation du résultat lorsque nous cliquerons sur le bouton pour prendre la photo.

Une vérification est faite sur le nom entrée, ce dernier ne peut être vide ni correspondre à un nom déjà présent dans les données joueurs.

Une fois le nom choisis et la photo prise, nous avons le choix de sauvegarder le profil nouvellement établi via le bouton de sauvegarde situé sous la prévisualisation ou d'abandonner la création via le bouton home. La procédure de sauvegarde enregistrera une scène avec comme texture l'image prise plus tôt et portant le nom du joueur.

## 6.9. Les Scores

Sur proposition de ma part, il fût prévu d'ajouter un système de score permettant de mieux analyser les résultats des élèves lors de leurs sessions de jeux.

Lorsqu'un enfant valide sa session de jeux, l'application vérifie l'existence d'un fichier de score au nom du joueur et le crée le cas échéant. Nous pouvons ensuite inscrire dans ce fichier le score de l'élève et le récupérer à tout moment afin de le mettre à jour suite à nouvelle session de jeux, ou tout simplement pour l'afficher sur l'écran individuel du joueur.

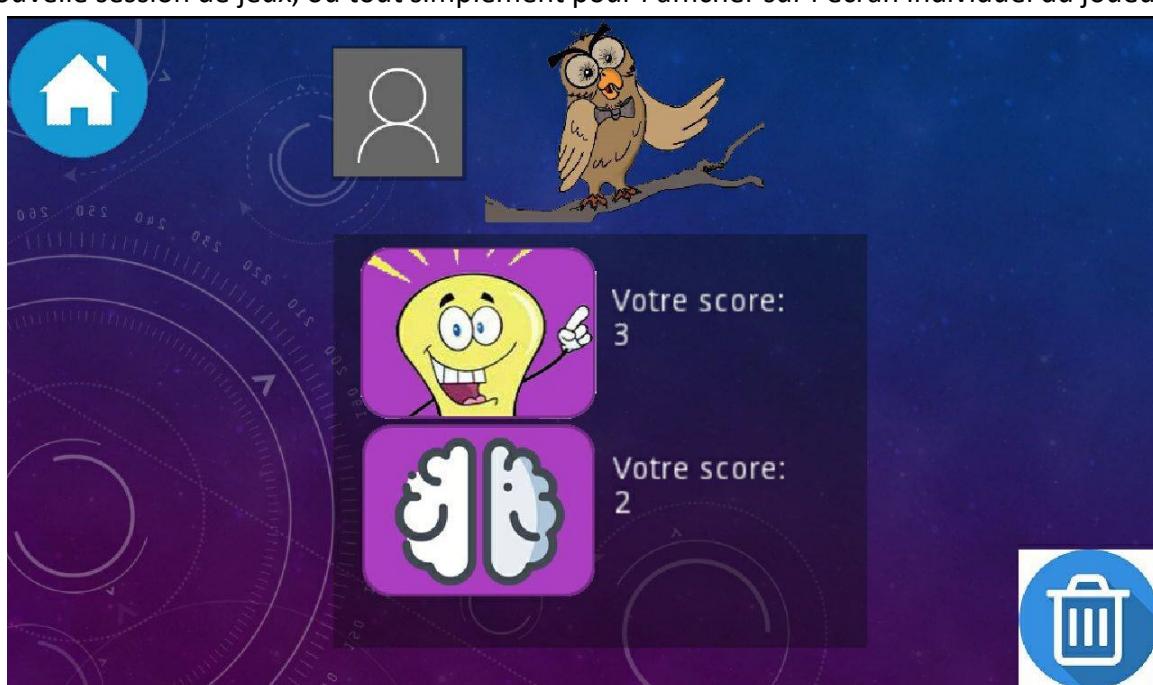


Figure 27: Scores et gestion joueurs

## 7. Problèmes et mésaventures

Même si l'évolution du projet présenté ici laisse entendre une évolution linéaire et logique, il n'en fût rien dans les faits. Comme tout projet, certaines limitations se sont imposées, des erreurs sont intervenues et tout ne s'est pas déroulé dans l'ordre que je présente ici.

Sans aller jusqu'à définir un ordre d'événements, certains de ces problèmes ont été suffisants que pour ralentir drastiquement mon avancement. Que ce soit au début du projet ou au beau milieu du développement, ces expériences m'ont permis de mieux comprendre les outils que j'utilisais afin d'en tirer le meilleur parti. Ceci m'a également permis de livrer au travers de ce rapport une compréhension du sujet plus pointue.

### 7.1. *GameMaker, la version perdue*

Etant détenteur d'une licence GameMaker, j'avais envisagé d'utiliser ce moteur pour le développement de l'application. Il présentait pas mal d'avantages, le plus important étant la base de connaissances que j'avais déjà acquise sur le moteur par le passé.

Je n'ai cependant appris que bien après avoir commencé le projet que ma version n'était plus en ordre vis-à-vis du Play Store. Tout mon écran était déjà prêt et le jeu de l'ampoule en était au balbutiement. Je m'apprêtais à scripter le choix aléatoire et dynamique des images en jeux lorsque je suis tombé par hasard sur la procédure d'export Android sur laquelle étaient mentionnées les informations concernant les réglementations pour le Play Store.

System requirements			
	<b>DESKTOP</b>	This is a permanent GameMaker Studio 2 licence which enables exports to Windows, Mac and Ubuntu.	\$99 Permanent licence
	<b>WEB</b>	This is a permanent GameMaker Studio 2 licence which enables exports to HTML5.	\$149 Permanent licence
	<b>UWP</b>	This is a permanent GameMaker Studio 2 licence which enables exports to Universal Windows Platform, including the Xbox One Creators Program.	\$199 Permanent licence
	<b>MOBILE</b>	This is a permanent GameMaker Studio 2 licence which enables exports to Android, Fire and iOS platforms.	\$199 Permanent licence

Figure 28: Plan tarifaire de GameMaker Studio 2.0

Au stade où j'en étais, impossible de transférer le projet, il me fallait le recommencer depuis le début. Non seulement cela, mais le fonctionnement de GameMaker ne se retrouve dans aucun autre moteur, chacun ayant sa propre méthodologie. Il me faudrait donc recommencer mon projet, mais également apprendre une nouvelle manière d'approcher mon projet.

## 7.2. *Les ressources*

J'ai en effet bien profité de cette expérience pour développer le point 5.5 du fait de son implication et de son importance. Quoi de plus rageant que de passer énormément de temps à développer et fignoler une partie du projet, partie qui fonctionne à merveille lors de vos test « in-engine » sur PC, mais plus du tout lors de l'exportation sur Android ?

Vous préparez donc l'export vers Android afin de voir votre petit bijou en action et là c'est le drame : plus rien ne fonctionne. La moitié des images n'apparaît pas. Les écrans sont presque vides. Les éléments variables, listes de joueurs, de mots, de jeux, plus rien ne s'affiche.

S'en est suivi pour moi la période la plus longue du projet. Je l'ai mentionné déjà, il s'agit de l'adaptation de tout mon code au format multiplateforme. Et pour cela il me fallut comprendre concrètement comment Godot allait placer les ressources lors de l'export et donc comment elles seraient organisées sous Android.

Référencer un fichier directement ne posait pas de soucis, il me fallait par contre adapter toute la partie dynamique que j'avais déjà passé un temps fou à établir correctement. Heureusement, comme j'avais fait ce travail consciencieusement dès le départ, son adaptation en fut facilitée. Mais je dis bien facilitée, cela ne veut en rien dire que ce ne fut pas un challenge à part entière.

J'étais par contre loin de me douter que ce ne serait pas la dernière fois.

## 7.3. *La récursivité, saturation des ressources système et boucle infinie*

Afin de moduler mon code au mieux, j'ai parfois fait le choix de créer des sous-fonctions afin de les ré-exécuter en cas de nécessité.

L'une des idées était que pour la sélection des images dans les jeux, l'application disposerait d'une fonction récursive. Cette fonction se rappellerait elle-même jusqu'à obtention d'un résultat correct.

Ici encore, les différences entre nos deux systèmes, Windows pour le développement et Android pour le résultat final, se fit sentir de manière significative.

```
func adder(dadd, arr):
    for i in range():
        if i == "" or i == "." or i == ".." or i in arr:
            adder(dadd, arr)
        else:
            instantiate(dadd, n)
```

Figure 29: Exemple de récursivité

Là où tout se passait pour le mieux sur Windows, l'application s'arrêtait sous Android de par la saturation de la mémoire. J'ai donc dû redoubler de prudence dans mon approche car même si cela fonctionnait en théorie, les limitations du contexte pratique ne permettaient pas d'adopter complètement ce mode de fonctionnement.

## 7.4. Les ressources 2 le retour : Attack of the node

Même en sachant tout cela, même en l'appliquant rigoureusement dans la suite du projet, je n'avais apparemment pas compris la leçon. Certaines subtilités du système m'échappaient encore et Godot ne rata pas l'occasion de me le faire comprendre. De manière brutale.

J'ai donc pris du recul et consulté les ressources à ma disposition : documentation officielle dans le moteur, sur le site officiel, diverses chaines Youtube, forum officiel.

Je comprenais le fonctionnement de Godot, mais pas encore assez pour le maîtriser suffisamment. Certains concepts de gestion des ressources m'échappaient encore et je ne maîtrisais pas non plus tout à fait la structure des scènes et des nœuds qui allait de pair avec ces ressources.

Je ne prétendrai pas savoir dans tous les détails les tenants et aboutissants de Godot et de la gestion d'un projet Android. Mais j'ai définitivement gagné en compréhension lors de la réalisation de cette application.

## 7.1. **Le module Caméra**

J'aborde ici le plus gros problème du projet une fois porté sous Godot. Le problème du module caméra se posa déjà dès le début. Comment faire pour utiliser un périphérique qui n'existe pas sur l'environnement où l'on développe ? Il peut tout à fait exister une caméra sur Windows, mais elle ne sera pas la même que sur Android.

Cette simple question soulève énormément d'implications et met en relief l'approche à adopter pour un projet multiplateforme. Si tout peut sembler fonctionner sur un système, comment garantir que tout fonctionnera de manière similaire sur un autre ? Et le cas échéant, comment faire pour que le code soit fonctionnel sur tous les systèmes ?

Fort heureusement, comme spécifié dans la section dédiée au moteur, je savais qu'il existait une communauté open-source et je n'étais très certainement pas la seule personne à me poser la question. Une simple recherche confirma qu'un plugin existait bien rendant la compatibilité de la caméra Android avec Godot tout à fait possible.

J'avais cependant énormément de pain sur ma planche et étais encore loin de me douter des retards que j'allais encourir de par son implémentation.

L'implémentation n'est pas difficile et la documentation Godot officielle suffit à comprendre la procédure. C'est ici le plugin lui-même qui posa problème.

Afin d'être utilisable dans Godot, l'auteur du plugin a dû en coder les propriétés et méthodes en java. Mais bien qu'il soit assez récent, certains morceaux de code java du plugin utilisant les fonctionnalités de Gradle étaient désormais désuets. Il m'aura fallut de nombreuses heures de recherche infructueuses avant de tomber sur la solution à ce problème.

J'arrive désormais à afficher la camera dans l'application. Cependant la manière dont le plugin fut réalisé est en total irrespect de la hiérarchie de Godot. La prévisualisation de la camera est donc superposée sur tout le reste et se trouve « hors » du champ d'action de notre code.

Prendre une photo devient dès lors très compliqué car si l'idée de base est de faire une capture d'écran et de post-traiter le résultat, cela devient impossible dans le cas présent puisque la camera se trouve en dehors de notre champ d'action.

## 8. Conclusions et améliorations

Concernant notre application, l'objectif fût non seulement d'établir nos fonctions principales, mais aussi de préparer le terrain pour le maintien futur de l'application au-delà du cadre de ce projet d'études.

Il ne fait nul doute que le projet en l'état actuel sera déjà d'une grande aide pour le public même s'il existe nombre d'améliorations que nous pouvons y apporter.

### 8.1. Bilan

En l'état, nous avons réussi à accomplir la majeure partie de nos objectifs permettant de rendre l'application viable. Nous avons une application compatible Android pouvant être uploadée sur le Play Store. Notre application contient deux jeux de base sur un maximum de trois prévus à l'origine: l'ampoule et le memory.

Nous avons réalisé un écran d'accueil, un écran de choix du thème ainsi que du jeu. Nos deux jeux sont jouables et permettent de conserver les scores du joueur.

Nous pouvons ajouter des joueurs à loisir. Nous disposons d'un écran score afin de consulter leurs points mais aussi de supprimer les joueurs.

L'application dans son ensemble fût réalisée avec possibilité de maintien facilité comme principe de base. Elle peut donc être facilement mise à jour pour satisfaire tout besoin futur.

### 8.2. Points forts du projet

Godot fût un moteur très agréable à utiliser. Bien qu'il demande une certaine adaptation de par sa nouveauté par rapport aux autres moteurs du marché, il offre de nombreuses possibilités et un système robuste pour tout type de projet.

J'ai encore récemment découvert qu'il offrait la possibilité de faire du Scripting visuel permettant vraiment à quiconque de pouvoir s'adonner aux joies de la programmation. Il ne troque cependant pas cet ajout au profit d'une diminution des possibilités qu'il offre.

C'est justement sa versatilité et sa légèreté qui en font un moteur digne des plus grands. Pour ma part, après avoir vécu ce projet avec ce moteur, je peux clairement dire que Godot est un moteur qui vaut la peine qu'on s'y intéresse.

Notre application quant à elle offre désormais toutes ses fonctions de base. La souplesse ajoutée à sa gestion des séries de mots et de jeux, permettra par la suite d'ajouter de nouvelles séries ou de nouveaux jeux facilement.

Les séries de mots peuvent facilement s'ajouter au projet. Les jeux seront quant à eux la partie qui demandera le plus d'effort car même si l'accès aux jeux via son icône ne prendra que quelques secondes à implémenter, il faudra d'abord préparer le jeu lui-même, ce qui est au final un processus naturel dans le développement d'une telle application.

Le plus gros point fort du projet cependant reste l'expérience acquise lors de sa réalisation. Dans le cadre du développement d'une application, il permet de bien se rendre compte de la manière dont il convient de gérer les ressources de projet, mais aussi de l'importance des bonnes pratiques de programmation par utilisation de certains design pattern.

### 8.3. *Limitations*

Tout choix est bien sûr tout à fait débattable. Dans le cas de notre projet la question se posa d'elle-même concernant les données utilisateur. Faillait-il utiliser une base de données pour l'ajout de série de mots ou de jeux ?

L'utilité d'une base de données serait restée fort limitée dans le cas présent. Disposer d'un serveur sur lequel nous mettrions toutes les ressources offertes par les jeux aurait été une solution plus onéreuse pour un projet de ce type. Il était donc plus pertinent de les conserver en local dans l'application même.

Les joueurs étant ajoutés eux-mêmes par chaque utilisateur de l'application, l'utilisation d'une base de données était là aussi peu pertinente. Nous n'aurions concrètement stocké que les avatars de ces derniers ainsi que leurs scores, sachant que le score était une partie optionnelle pour le client qui n'est présente que de par ma suggestion à ce dernier.

Dans le cas d'une base de données locale, le fonctionnement aurait reposé sur un fichier json<sup>10</sup> crypté par l'application afin d'en sécuriser les informations.

Il n'était donc pas pertinent d'implémenter de telles mesures étant donné qu'aucune information réellement sensible n'en aurait fait l'objet (Score et image).

---

<sup>10</sup> JavaScript Objet Notation : Format de données textuelles dérivé du Javascript.

Nous ne traitons en effet absolument pas avec des mots de passe. Nos joueurs ne sont qu'un ensemble d'images et de scores à des fins de recherche. De ce fait, seul le professeur ayant mené la recherche sur un élève afin d'en relever le score trouverait cette information pertinente.

De même, aucune ressource du jeu n'est sensible. Elles ne sont constituées que d'un ensemble d'images basées sur un thème précis et fournies aux joueurs sous forme de jeux simples à des fins d'études par un professeur compétent ayant lui-même demandé au joueurs d'utiliser l'application. L'intérêt de s'emparer et/ou de sécuriser ces données étaient donc également inexistantes.

## **8.4. Évolution future**

Les possibilités de maintien de l'application par l'ajout des jeux et des séries de mots afin de diversifier la sélection proposée seront le point culminant du projet.

Le client avait en effet envisagé plusieurs jeux possibles à ajouter, environ 5 au total et chacun différent du précédent, ce qui pour des raisons évidentes de gestion de temps ont du être réduites à 2 pour le projet (Voir 3 si le temps le permettait, ce qui ne fut pas le cas).

Il n'est également pas impossible que ce projet soit également utilisé dans le cadre de séances logopédiques. Ce type d'activités étant capital dans l'établissement d'un diagnostic concernant les difficultés d'apprentissage que certains enfants peuvent rencontrer dans leur développement.

Il existe donc un réel intérêt à faire perdurer ce projet sur le long terme. Son adaptation se fera au besoin du contexte dans lequel il se positionnera au fil du temps.

## **8.5. Conclusion finale**

Ce projet fût une aventure des plus enrichissantes. J'ai non seulement eu l'occasion de mettre en pratique nombre de concepts vus lors de mes études mais j'ai également pu en apprendre plus sur la conception d'application Android par le biais de moteur de développement.

Sa réalisation ne fût pas sans encombre. Mon parcours a constamment été ponctué par les divers soucis qui se posent lors de tout projet de ce type. Erreurs de code, recherches extensives, périodes de crunch. Le plus surprenant est que je persiste à survivre à tout cela sans une seule goutte de café.

Bien qu'il puisse paraître simpliste, ce projet a su montrer à quel point il était néanmoins complet et complexe. Preuve s'il en faut que les projets de développement ne doivent jamais être pris à la légère.

Qui plus est, fort de cette expérience, tous les autres projets que j'envisageais m'apparaissent désormais avec plus de clarté. Comme si un chemin venait de s'illuminer dans mon esprit, me permettant de voir plus clairement quels sont mes objectifs et comment les atteindre.

Le fait d'utiliser un moteur permet aussi de désacraliser le processus de création d'un jeu. Car même si l'on peut comprendre la logistique de programmation ou la non-linéarité de l'orienté objet par des cours magistraux, le fait d'avoir un projet concret réalisé de A à Z permet de mieux appréhender les choses. Chaque aspect doit en effet faire l'objet d'une étude concrète et doit pouvoir se justifier dans le contexte.

J'en retire malgré tous les soucis rencontrés une certaine gratification, notamment, puisque l'application sera disponible sur le Play Store. Même si le projet demandera du maintien, le sentiment d'accomplissement demeure.

En conclusion, ce projet m'aura permis d'affirmer les acquis de mes trois années d'études mais aussi de poser les bases de mes projets futurs. Il m'a permis de mettre en relief les parties manquantes de mes connaissances, tout en consolidant les bases déjà présentes.

## 9. Bibliographie

GDQuest, [en ligne]

<https://www.youtube.com/channel/UCxboW7x0jZqFdvMdCFKTMsQ>

Endrit Special Productions, Fast Help YouTube video series [enligne]

<https://www.youtube.com/channel/UChoaAUkZdYXXjCCQaxBcJhQ>

Gamefromscratch, [enligne]

<https://www.youtube.com/user/gamefromscratch>

Godot Engine, [en ligne]

[https://www.youtube.com/channel/UCKIDvfZD1ZhY4\\_hhbotf7wA.](https://www.youtube.com/channel/UCKIDvfZD1ZhY4_hhbotf7wA)

GDScript.com, [en ligne]

<https://gdscript.com/>

Godot Docs, [en ligne]

<https://docs.godotengine.org/fr/stable/>

Github, [en ligne]

<https://github.com/funabab/godot-camera-plugin-demo/pull/2/files>