Application pour l'Augmentation via Smartphones d'un Jeu de Carte Physiques

Lucie Chasan, Lorian Corbel, Hugo Jalenques, Chloé Pathé

Sujet proposé par : Jérémy Laviole

Mars 2020

Table des matières

1 Introduction

Jumathsji est un jeu de plateau permettant de faire des mathématiques en s'amusant. Créé par les élèves de l'école primaire de Saint-Michel de Castelnau et du collège Ausone de Bazas à l'occasion de la semaine des mathématiques de 2019, il a mobilisé l'ensemble des équipes pédagogiques des deux écoles, ainsi que le Lycée des Iris de Lormont lors de sa création. En effet, les différentes pièces qui le composent ont été créées pendant les cours des élèves afin de les impliquer un maximum dans le projet : les règles ont été rédigées en cours de français, les questions d'anglais pendant le cours du même nom, les pièces ont été conçues et imprimées en 3D en cours de Technologies et les dessins illustrants les différentes cartes du jeu ont été réalisés en Arts Plastiques.

Le scénario du jeu est le suivant : enfermés dans le plateau de jeu, les élèves doivent s'échapper en récupérant des badges correspondants à cinq "domaines" : la montagne des problèmes, la vallée des nombres, la plaine des 2D, l'espace 3D et "English Maths". Pour récupérer ces badges, l'élève doit répondre à des questions correspondant à ces domaines et écrites sur des cartes de jeu. Elles sont au nombre de trois cents. Les réponses se trouvent dans un livret à part et les élèves en difficultés peuvent disposer d'une aide (si la question le permet), elle aussi disponible dans un livret à part. Si l'élève répond correctement à la question, il peut avancer son pion sur le plateau. Dans le cas contraire, il perd une vie.

Figure 1 – Affiche de la semaine des mathématiques 2019 représentant le plateau de jeu

Ce jeu, dont les règles sont disponibles en ligne sur le site officiel du jeu [1], ne dispose cependant d'aucune composante technologique. Hors il serait particulièrement intéressant dans le cadre de ce projet pédagogique de permettre un éveil aux nouvelles technologies aux élèves, ainsi que de faciliter l'accès au jeu aux élèves qui pourraient souffrir de ne disposer que de la version papier du jeu.

Afin de répondre à cette attente, notre projet de fin d'études se propose de créer une application Android, ce système d'exploitation équipant près de 70% du parc des appareils mobiles en France, afin de pouvoir reconnaître les différentes cartes selon leur catégorie, et enfin de les afficher à l'écran.

Ce rapport étudiera d'abord les applications et les technologies déjà existantes, que ce soit du côté des jeux ou de la simple reconnaissance de texte, puis s'intéressera aux récits utilisateur, avant de se pencher sur l'application finale telle qu'elle a été prototypée, l'architecture du logiciel et son implémentation. Nous finirons enfin par étudier les tests réalisés.

2 État de l'art

2.1 Jeux utilisant la Réalité Augmentée

La plupart des jeux sous Android utilisant de la Réalité Augmentée dans leurs mécaniques de jeux utilisent Unity et parfois Vuforia, ces derniers permettant respectivement de créer le jeu et son interface pour l'un, et de rajouter la partie Réalité Augmenté pour l'autre.

2.1.1 Unity

Unity est un moteur de jeu développé par Unity Technologies et ayant une licence propriétaire. Très utilisé dans l'industrie du jeu video, il permet de réaliser un jeu en 2D ou en 3D, avec ses modèles, ses règles et son application. Le modèle économique de Unity permet aussi aux étudiants, aux particuliers, et aux petites entreprises gagnant moins de 200.000$, de l'utiliser gratuitement. Les entreprises plus grosses doivent, elles, payer une licence.

Un exemple de jeu sur téléphone ou tablette utilisant le moteur Unity et ayant des mécaniques de jeu orientées Réalité Augmentée est le jeu *Pokémon GO*. Le but de ce jeu est de capturer un maximum de *Pokémon*, des créatures imaginaires qu'il faut ensuite entraîner et faire combattre dans des lieux spécifiques (les arènes) ou contre ses amis. Afin de capturer et de combattre, le joueur doit se déplacer physiquement dans son environnement, l'application enregistrant les déplacements à l'aide des coordonnées GPS de l'appareil utilisé. La composante Réalité Augmenté intervient elle pendant la capture : si le joueur le souhaite, il peut activer l'appareil photo de l'appareil mobile et apparaîtra alors sur son écran une image de son environnement avec le pokémon incrusté de façon cohérente, comme s'il était réellement présent, comme on peut le voir en figure 2. Cette incrustation est cependant optionnelle et est gourmande en batterie.

Figure 2 – Exemple de capture de Pokemon utilisant la RA - *Pokémon GO*, 2016, *The Pokemon Company* [2]

Le principal problème de Unity est qu'étant un moteur de jeu, il est peu indiqué pour faire une simple application android. De plus, l'intégration d'une reconnaissance de caractères et d'un réseau de neurones pourrait poser problème. Enfin, notre client nous a demandé de ne pas utiliser cette technologie, pour des raisons qui lui sont personnelles.

2.1.2 Vuforia

Vuforia est un kit de développement propriétaire appartenant à PTC et permettant d'ajouter une composante réalité augmentée à des applications en JAVA, C++ ou les langages .NET à partir d'un module UNITY. Vuforia, associé à UNITY, permet de créer une composante Réalité Augmenté plus facilement qu'avec Unity seul. On remarquera aussi qu'il est possible de développer avec Vuforia sous Android sans pour autant utiliser Unity.

Vuforia, en plus de la composante AR, permet d'apporter à une application deux fonctionnalité qui nous intéressent : la transformation des cartes et la reconnaissance de caractères :

— Les transformations de cartes sont les transformations affines nécessaires pour permettre la reconnaissance de caractères. En effet, il n'est pas réaliste de croire que l'utilisateur utilisera toujours l'application dans des conditions optimale. On considère que les conditions sont optimales quand la lumière est suffisante pour analyser le texte sans soucis, qu'il n'existe aucune distorsion dans l'acquisition de l'image et que la carte est acquise dans le bon sens.
Il existe des méthodes dans Vuforia qui permettent de gérer ces deux derniers cas, c'est-à-dire remettre la carte dans le bon sens et effectuer les transformations affines nécessaire pour éliminer les distorsions.

— La reconnaissance de caractères est permise grâce à l'implémentation d'une liste de mots qui seront reconnus. Cette liste est soit fournie, et peut être complétée, soit être créée ex-nihilo. Ensuite, on peut utiliser la reconnaissance de caractères disponible dans la bibliothèque. On remarquera cependant que Vuforia ne reconnaît qu'une quantité limitée de caractères : les caractères de ponctuation point et virgule, le tiret et l'espace, ainsi que les caractères latins majuscules et minuscules. Vuforia ne reconnaît pas les chiffres et ne reconnaît pas non plus les caractères spéciaux de types lettres accentuées (la liste complète des caractères reconnus est disponible dans la documentation de Vuforia [3]).

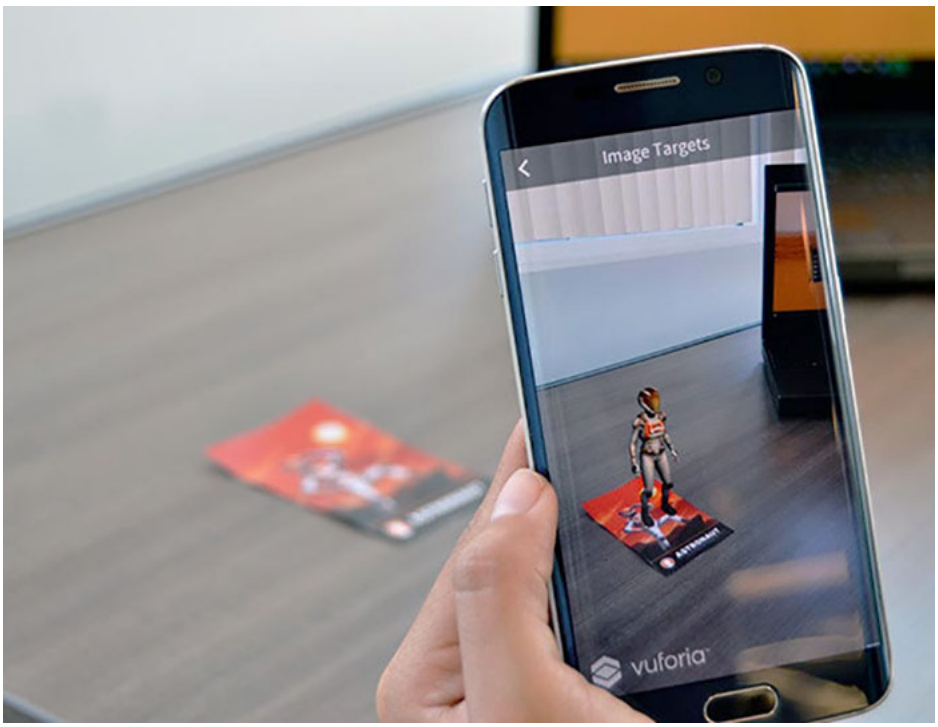

Figure 3 – Exemple d'application AR réalisée avec Vuforia - *Vuforia.com*

Vuforia est cependant une technologie que nous ne pouvons pas utiliser. En effet, en dehors du fait que le projet doit être le plus libre de droit possible, une licence Vuforia est facturée 504 dollars, soit 465 euros, par an pour les petites entreprises ou les étudiants. Bien que l'abonnement ne soit à payer qu'à la mise en service du produit, étant donné que l'application a vocation à être mise en service, cela ne semble pas un prix raisonnable pour ce projet.

2.2 Reconnaissance de caractères

Il existe de très nombreuses bibliothèques de reconnaissance de caractères, par exemple Vuforia, comme cité précédement, mais aussi Google ML Kit et Tesseract.

2.2.1 Tesseract

Tesseract est un logiciel de reconnaissance de caractères d'abord développé par Hewlett Packard entre 1985 et 1994, et dont le code, mis à disposition libre de droit depuis 2005, est sponsorisé par Google.

Bien que Tesseract soit à l'origine un logiciel propriétaire, la mise à disposition gratuitement et libre de droit de son code source, disponible sur GitHub [4], permet son utilisation dans d'autres applications. Parmi les applications utilisant Tesseract et disponible sur téléphone Android, on peut citer Text Fairy. Text Fairy est une application de reconnaissance de texte qui permet de transformer un document papier en PDF, et ce dans plusieurs langues. Il suffit pour cela d'utiliser la caméra du téléphone et, une fois le document photographié, de sélectionner la zone à reconnaître. Comme Tesseract, Text Fairy est parfaitement capable de reconnaître des textes dans différentes langues, mais semble incapable de reconstituer correctement de simples équations, bien que les chiffres soient reconnus.

2.2.2 Google ML Kit

Google ML Kit est une interface de programmation d'application permettant de faire de l'apprentissage par ordinateur sur téléphone. Plusieurs modèles sont disponibles, dont la reconnaissance de visages, la lecture de code barres et la reconnaissance de caractères. Tous ces modèles se basent sur la reconnaissance d'images. Le grand avantage de cette interface est que la prédiction du modèle peut se faire en local de façon gratuite, et donc ne pas utiliser les services en ligne de Google, qui eux sont payant. Bien que des modèles préexistant soient disponible, il est bien sur possible de créer le sien et de l'entraîner. De plus, ML Kit est intégré à Firebase, une plateforme de développement d'applications mobile et web possédée par Google, et il est facile de l'intégrer à un projet Android depuis Android Studio.

2.3 Base de Données

Pour la base de données, deux moyens sont employés : le langage SQL, par l'utilisation de SQLite, et le format de représentation de données YAML.

2.3.1 SQL et SQLite

SQL est un langage normalisé servant à exploiter des bases de données relationnelles. Sa partie manipulation des données permet de chercher, ajouter enlever et modifier des données. Il est aussi possible de modifier l'organisation des données et de restreindre l'accès aux

données.

SQLite est une bibliothèque libre de droit qui permet de se passer d'un modèle de base de données client-serveur traditionnelle, le moteur de base de données est ici directement intégré à l'application et chaque base de donnée à son propre fichier. L'accès aux données est alors plus rapide et sécurisé qu'avec d'autres formats, textes par exemple.

Le problème de SQL et de SQLite est la création de la base de donnée en elle même. En effet, s'il est facile pour un développeur, débutant ou confirmé, d'utiliser ce langage, il est plus difficile à appréhender pour quelqu'un qui n'a jamais développé. Hors, si les utilisateurs de l'application sont des élèves de collège, il faut aussi prendre en compte le fait qu'un professeur veuille compléter la base de données fournies, ou en créer une nouvelle. Or il doit être possible pour un utilisateur autorisé voulant améliorer la base de donnée disponible de le faire sans grande difficulté.

2.3.2 YAML

YAML est un format de représentation de données par sérialisation. YAML essaie de décrire l'ensemble de ses données sous la forme d'une combinaison de liste, de tableaux et de scalaire. L'ensemble de ses combinaisons sont décrites par un langage simple et compréhensible par un humain, ce qui le rend facile à utiliser. De plus, sa syntaxe est facilement compréhensible.

Comme expliqué précédemment, il doit être possible de rajouter des données facilement, et ce peut importe le niveau de compétence en informatique de la personne. YAML permet tout à fait de faire cela.

2.3.3 Technologies choisies

Les technologies choisies répondent aux attentes suivantes : elles doivent être facile à utiliser, et ce même pour quelqu'un qui n'a pas beaucoup d'expérience en développement, elles doivent être au maximum libre de droit, ou au moins être mises à disposition du public gratuitement, utiliser les ressources locales du téléphone et enfin répondre aux attentes de notre client.

Dans cette optique, nous avons donc choisi d'utiliser les technologies suivantes :

— Android Studio seul pour la réalisation du prototype de l'application. En effet, comme évoqué plus haut, Unity n'est pas adapté à notre projet.
— Nous utiliserons ML Kit plutôt que Tesseract pour la reconnaissance des cartes, ML Kit permettant de pousser la reconnaissance aux images, et non seulement aux caractères, contrairement à Tesseract. En effet, il arrive que la seule chose différenciant deux cartes soit l'image qui accompagne le texte. Donc, si la reconnaissance de caractères est souvent suffisante et se justifie sur la plupart des cartes, elle n'est pas suffisante pour l'ensemble des cartes et doit parfois être complétée par une analyse de l'image.
— Enfin, pour la base de données nous avons choisi le format YAML, afin de stocker la représentation des cartes de manière simple et facile à comprendre, couplé à SQLite. Comme expliqué précédemment, il faut que, par la suite, un enseignant puisse rajouter des données dans la base de données, ou en créer une nouvelle, facilement. Étant donné qu'il serait inefficace de commencer avec une technologie pour en changer par

la suite, on préférera directement utiliser YAML. Cependant, on utilisera aussi SQLite pour faire les requêtes dans l'application même. En effet, YAML étant seulement un format, il ne comprend pas les méthodes nécessaires pour écrire les requêtes nécessaires à l'utilisation des données. SQLite permet de réaliser ces requêtes sur les tables contenues dans la base de données facilement et sans que nous ayons besoin de tout créer de nous-même.

3 Récits Utilisateur

Avant de parler des récits utilisateur et d'analyser les besoins, il faut d'abord bien identifier qui est l'utilisateur. Après avoir discuté avec le client, on peut définir notre utilisateur de la façon suivante : l'utilisateur est un ensemble d'élèves de CM1, de CM2 ou de 6ème, donc des enfants entre 9 à 12 ans, à qui on a mis à disposition un plateau de jeu JUMATHSJI, l'ensemble des pièces du jeu (les règles, les cartes question, les pions, etc...), ainsi qu'un appareil tactile de type tablette ou téléphone équipé de l'application à réaliser, et qu'on aurait potentiellement laissés autonomes dans un CDI pour jouer au jeu.

3.1 Analyse du récit utilisateur

Un récit utilisateur peut se résumer de la façon suivante : En tant qu'Elève de CM1/CM2/6ème, je veux pouvoir avoir accès aux cartes questions du jeu JUMATHSJI, à l'indice correspondant à la question s'il existe, et à la réponse à la question sur une tablette tactile lorsque je joue au jeu de mathématiques JUMATHSJI. On peut donc, de ces observations, conclure que :

- Il n'y a pas un utilisateur unique, mais plusieurs utilisateurs utilisant le même médium, la même application. Cela signifie qu'il faut pouvoir réinitialiser la partie centrale de l'application (la rechercher de carte) de façon rapide et intuitive.
- Les utilisateurs sont des enfants. Il faut donc, en dehors de l'intuitivité de l'application, que cette dernière soit la moins frustrante possible. Il ne faut donc pas que les différentes fonctionnalités soient cachées dans des menus, ou que les boutons soient trop rapprochés les uns des autres, de telle sorte qu'un appui au mauvais endroit renvoie sur le menu d'accueil, par exemple.
- Toujours par rapport à l'âge des utilisateurs, et comme déjà évoqué, il faut prendre en compte le fait qu'il est impossible de prédire que les conditions d'utilisations seront optimales :
  1. Il est possible que, lors de la capture d'image, la caméra tremble.
  2. Il est aussi possible que la carte ne soit pas parfaitement alignée par rapport à la caméra, que cela soit volontaire ou pas.
- Pour finir sur l'âge des utilisateurs, il faut faire bien attention à ne pas les mettre dans une position où le respect de leur vie privé serait remis en cause. En effet, les enfants sont particulièrement sensibles à la publicité et aux manipulations qui lui sont liées. L'application ne doit donc renvoyer aucune information vers l'extérieur. Elle peut par contre recevoir des données de l'extérieur.
- Enfin, il faut prendre en compte le fait que l'application sera utilisée principalement sur les appareils tactiles équipant les CDI, dans des écoles qui ne sont pas forcément équipées d'une bonne connection internet. Il est donc logique de supposer que :
  1. Les appareils disponibles sont vieux et ne disposent donc pas des dernières versions d'Android. Il faut donc que l'application soit utilisable par ces vieilles versions, et donc que les technologies utilisées soient aussi disponibles pour ces versions.
  2. Il n'est pas possible de stocker la base de données contenant les cartes, les indices et les réponses en ligne. Il faut que la base de données soit téléchargée avec l'application, ou qu'elle soit téléchargeable par la suite et stockée dans le téléphone. Il n'est pas non plus possible d'utiliser des services en ligne, tel que les prédictions d'un modèle d'apprentissage par ordinateur en ligne.

3.2 Étude d'une carte

Avant de définir les besoins de notre application, il semble important de caractériser une carte. La figure 4 ci-dessous nous servira d'exemple.

Figure 4 – Exemple de carte à reconnaître

Comme on peut le voir sur l'exemple 4, une carte est constituée des éléments suivants, toujours disposés de la même façon :

1. Un code unique, et spécifique à la carte, qui décrit le type de question, le niveau de l'utilisateur qui doit y répondre et le numéro de la question. Dans notre exemple, ce code est *e3dcm1q_1*. On peut l'analyser de la façon suivante : e3d indique qu'il s'agit d'une question d'*Espaces 3D*, cm1 qu'il s'agit d'une question pour un élève de CM1 et q_1 que c'est la question numéro 1.
2. Un titre définissant le type de question. La couleur du titre correspond au type de question et correspond à la couleur de la question sur le plateau de jeu. Cette couleur se retrouve aussi sur les codes de réponse et d'aide que nous verront par la suite. Dans notre exemple, le titre est *Espaces 3D*.
3. Un texte énoncé, qui sera utilisé pour analyser la carte. Ce texte n'est pas unique et ne peut donc définir seul une carte. De plus, ce texte peut être constitué de plusieurs lignes. Dans la figure 4, le texte est *Combien y-a-t-il de cubes dans cette forme ?*
4. Une image, de façon optionnelle. Cette image est unique et peut donc permettre à elle seule d'identifier une carte. Cependant, toute carte n'a pas d'image.
5. Le code réponse, situé en bas à gauche, dans la même couleur que le titre. Ce code correspond à une réponse inscrite dans le livre de règles. Ici, le code est *R1*.

6. Le code aide, qui permet à un élève ayant des difficultés à répondre d'avoir un indice sur la réponse. Là encore, il est de la même couleur que le titre et correspond à une entrée dans le livre de règles. A noter qu'il n'existe pas toujours d'aide pour une question. Dans notre exemple, l'indice est le *A1*.

3.3 Étude des besoins

Le projet était initialement découpé en quatre parties (dites "lots" dans le sujet) :

1. L'interface de l'application
2. L'identification des cartes
3. La reconnaissance de cartes
4. La gestion du jeu de données de cartes

L'intérêt principal du PFE était la veille technologique d'outils de reconnaissance de contenu afin de remplir les contraintes intrinsèques aux conditions d'utilisation spécifiques de l'application (matériel peu performant, utilisation hors-ligne). Les lots 1 et 4, l'application Android et la gestion des cartes, n'étaient pas comprises dans le sujet du PFE car réservées pour le sujet de stage en découlant. Cela étant, des versions minimales de ces deux lots ont été réalisées afin de fournir un environnement de tests et de démonstration suffisant.

Les besoins de l'utilisateur peuvent être classés en deux catégories : les besoins de l'application et les tests nécessaires pour garantir le bon fonctionnement de l'application.

3.3.1 Application

Le projet doit répondre aux besoins suivants :

1. Un prototype d'application, qui permet de lancer les autres fonctionnalités. Ce prototype n'a pas besoin d'être graphiquement plaisant, mais il doit cependant être sans bugs et fonctionnel.
2. Un ensemble de fonctions permettant d'identifier une carte du jeu JUMATHSJI à partir du texte de son énoncé, et si le texte ne suffit pas, comme dans le cas où plusieurs cartes aurait le même texte, de reconnaître l'image qui accompagne toujours la carte dans ces cas.
3. Permettre de trouver une carte par un identifiant unique dans les cas où la reconnaissance de carte serait infructueuse.
4. Afficher la carte sur l'écran de l'appareil utilisé. Comme évoqué précédemment, une carte doit contenir son titre, son image et les boutons affichant l'aide s'ils existent, sa réponse, et son identifiant unique. En plus de ces caractéristiques, l'écran doit afficher un bouton permettant de relancer la recherche de carte par OCR, un champ permettant d'entrer le code de la carte si celle trouvée n'est pas la bonne, ainsi qu'un bouton permettant de revenir au menu principal.
5. Un début de base de données définissant les caractéristiques nécessaires d'une carte, et ce afin de pouvoir compléter la base de données par la suite.

La figure 5 montre un prototype papier de l'application. On peut y voir le nom de l'application - Jumathsji -, une loupe indiquant l'option de recherche, un bouton home indiquant la possibilité de revenir au menu principal, l'identifiant, le titre, le texte de l'énoncé, l'image, un bouton de réponse à gauche, un bouton d'aide à droite et un bouton caméra au milieu indiquant comment relancer la recherche de carte.

FIGURE 5 – Prototype papier de l'application - *Réalisé à l'aide de proto.io*

3.3.2 Tests utilisateurs

L'application doit répondre aux impératifs suivants :

1. La rapidité d'exécution : les cartes doivent être trouvées rapidement par l'OCR, ou dans le cas où la carte n'est pas trouvée, la recherche de carte par son identifiant doit être rapide. Il est bien sûr normal qu'en fonction de l'appareil utilisé la vitesse d'exécution varie. Cependant, il faut que, même sur un appareil ancien, l'application soit rapide. Il faut donc tester la vitesse d'exécution des méthodes de recherche de cartes de façon répétée et ce sur plusieurs appareils.
2. L'application doit être résistante à un enfant appuyant partout. Seuls les endroits définis sur la dalle tactile doivent donc réagir. Un "Monkey Test" est donc primordial. De plus, s'il existe des endroits dans l'application où la place a été réservée pour déclencher d'autres fonctionnalités de l'application, ses boutons, ou textes, ne doivent pas déclencher de bugs.
3. La complétude de la base de données. En effet, il faut vérifier que chacune des cartes décrite dans la base de données contienne bien les données nécessaires. En l'occurrence, il faut impérativement vérifier que l'ensemble des caractéristiques d'une carte sont remplis.
4. Il faut enfin tester les taux de ressemblance des textes des différentes cartes. En effet, il est nécessaire de savoir combien de cartes peuvent poser problème du fait qu'elles ont un texte identique ou très semblable.

4 Application Finale

Afin de répondre aux besoins des utilisateurs définis dans les Récits Utilisateurs de la partie 3, nous avons réalisé une application la plus épurée, ainsi que la moins frustrante possible. La figure 6 présente les quatre écrans principaux qui la composent.

(a) Écran au lancement

(b) Écran après téléchargement des données

(c) Écran pendant la reconnaissance de carte

(d) Écran à l'affichage d'une carte

FIGURE 6 – Les quatre écrans principaux de l'application

Cette interface comprend plusieurs éléments essentiels. Dans la figure 6a, on peut voir un bouton "Mettre à jour Jumathsji". Il est nécessaire d'appuyer sur ce bouton pour télécharger les données de la base de données après avoir téléchargé l'application. Une fois que les données ont été téléchargées, ou mises à jour si on re-télécharge les données, on obtient l'écran de la figure 6b. Une fois les données téléchargées, on peut cliquer sur le bouton caméra en bas à droite. Ce bouton permet de lancer la reconnaissance de carte.

La figure 6c montre l'utilisation de la caméra lorsque la reconnaissance de cartes est lancée. La reconnaissance a été faite sur un écran d'ordinateur mais cette particularité ne change en rien le fonctionnement de l'application.

Comme on peut le voir dans le bandeau sur la partie haute de l'application, il existe deux zones interactives : la première, une loupe, doit permettre d'ouvrir une zone de texte où l'identifiant de la carte pourra être inscrit (cette fonctionnalité n'a pas été implémentée à cause de la mise en suspend des projets) ; la deuxième permet de revenir sur le menu de départ, soit la figure 6a.

Enfin, la figure 6d montre l'écran obtenu quand une carte est reconnue. Toujours dans la partie haute de l'application, le même bandeau que vu au paragraphe précédent, avec les mêmes fonctionnalités : c'est à dire la recherche par identifiant, qui est implémentée pour cette partie, et le retour à l'écran d'accueil. En dessous, l'identifiant de la carte, ici *e3dcm1q_1*, puis son titre : *Espaces 3D*, son énoncé : *Combien y-a-t-il de cubes dans cette forme* et son image. Enfin, en bas, un bouton *R1* permettant d'afficher la réponse, un bouton caméra pour relancer la reconnaissance de carte, et enfin un bouton *A1*, présent uniquement si la question possède une aide, qui permet d'afficher cette dernière.

On peut remarquer que la présentation de la carte est semblable à celle de la carte physique présentée précédemment.

Les bandeaux blancs, en haut et en bas de chaque capture d'écran correspondent à la barre d'état du téléphone (en haut) et à la barre de navigation (en bas). Elles dépendent du téléphone et ne sont pas comprises dans notre application.

5 Architecture du logiciel

Comme expliqué précédemment, ce projet est à l'origine divisé en 4 lots, et bien que notre objectif soit seulement d'implémenter la reconnaissance de carte et leur identification, afin d'obtenir un prototype de l'application finale fonctionnel il a aussi fallu mettre en place une application android "basique" et un début de gestion des cartes. Il découle de ces différents lots 4 packages dans notre architecture : **DB**, **Download**, **OCR** et **UI**.

5.1 Package UI : Application Android

Pour commencer, la package **UI** représente l'interface utilisateur de notre application android. L'ensemble des layouts et des activités de ce package n'a pas vocation à être définitif mais est là afin de permettre la liaison entre les autres packages. On trouve ainsi dans ce package les classes **MainActivity** et **SecondActivity**. **MainActivity** permet la liaison avec le package **Download** et **SecondActivity** avec le package **OCR** et **DB**.

5.2 Package DB et Download : Gestion des Cartes

Ensuite, les packages **DB** et **Download** permettent tous deux la gestion des cartes. Le package **Download** est seulement composé de la classe **UpdateDBTask**. Puisque les cartes présentes dans le jeu Jumathsji sont susceptibles d'être modifiées il faut que les données soient disponibles en ligne. Mais l'une de nos principales contraintes était que l'application puisse être utilisée principalement hors ligne. De ce fait, la classe **UpdateDBTask** permet de télécharger ou de mettre à jour la base de données du jeu sur le stockage externe du téléphone (voir le diagramme de séquence figurant à la figure 11 en annexes).

Le package **DB** regroupe toutes les classes relatives à la base de données. La principale classe **AppDataBase** permet les accès à cette base de données. Seulement ces accès nécessitent d'être effectués de manière asynchrones, de ce fait des classes asynchrones ont été nécessaires, comme par exemple **CardSearchTask** présent dans ce package qui permet de trouver une carte dans la base de données selon son numéro. On retrouve aussi ici les classes permettant de représenter les cartes physiques Jumathsji sous forme d'objets dans notre application. On a ainsi, par exemple, la classe **CardWithLines** qui permet de combiner les deux classes **Card** et **Line** puisque le texte d'une carte physique est stocké sous forme de lignes pour la reconnaissance. Un exemple d'utilisation sur le diagramme de séquence figure 12 se trouve en annexes .

5.3 Package OCR : Reconnaissance et Identification des Cartes

Enfin, le package **OCR** regroupe l'ensemble des classes permettant de reconnaître et d'identifier la carte de l'utilisateur. Ce package est composé de 4 classes. Tout d'abord, la classe **TextRecognitionActivity** s'occupe de l'acquisition des images venant de la caméra du téléphone. C'est aussi cette classe qui s'occupera de transmettre le résultat de l'identification vers l'UI à la fin de la reconnaissance. La classe **TextAnalyser** permet d'extraire le texte ligne à ligne capturé via la caméra précédemment. Pour cela, on utilise l'OCR fourni par les services Firebase de Google dans *ML Kit* [5]. A partir du texte détecté, la classe **SearchTask** effectue des requêtes dans la base de données de manière asynchrone afin d'identifier à quelle carte le texte appartient. Cependant, il est possible que plusieurs cartes aient exactement le même texte, dans ces cas là, c'est l'image présente sur la carte qui permet de les différencier. Mais un OCR n'est pas capable de le faire. Pour cela, la classe **TfInterpreter** permet d'utiliser un classifieur CNN lorsque c'est nécessaire. Notre CNN est basé sur le modèle

MobileNet [6], et nous utilisons TensorFlow Lite fournit aussi dans le ML Kit de Firebase. Le diagramme de séquence figure 13 en annexe résume tout cela .

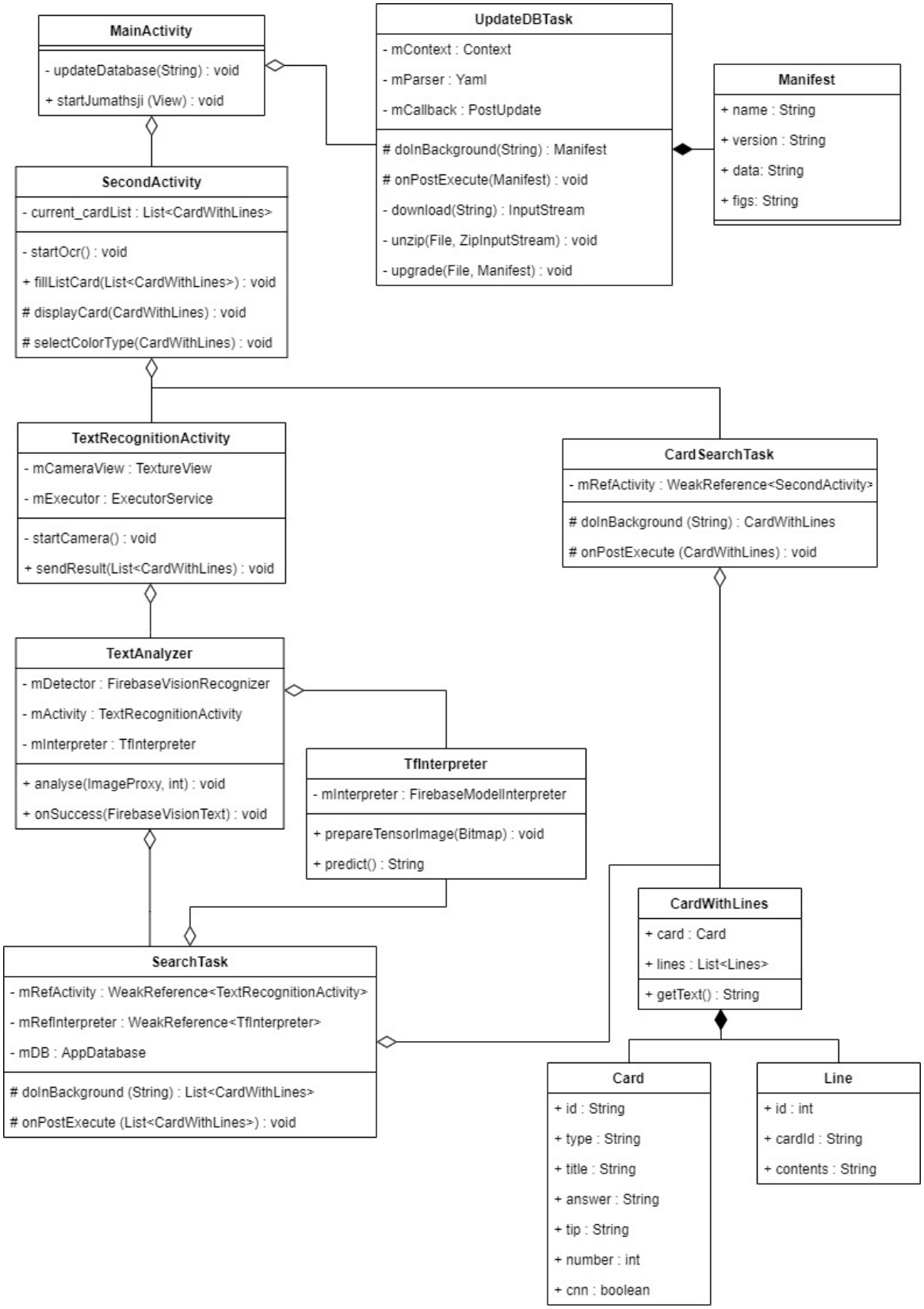

FIGURE 7 – Diagramme de Classe résumant l'architecture. Chacunes des classes ayant -**Task** dans leur nom sont les classes qui accèdent à la base de donnée via **AppDataBase**.

6 Implémentation

6.1 Application Android

La base de l'application a été réalisée en Android. Comme toute application Android, elle est divisée en plusieurs écrans, qui constituent des **Activités**. Chaque activité est composée de plusieurs fichiers : un ou plusieurs fichiers xml, qui permettent de définir l'apparence de l'application quand l'activité est appelée, ainsi que les fonctionnalités qui pourront être utilisées lors de l'utilisation, telles que les zones qui afficheront la caméra ou les zones dites "on-click" qui permettront d'interagir avec l'application. L'autre partie de l'application est constituée de fichiers java qui contiennent la logique nécessaire au fonctionnement de l'application.

Notre application est divisée en trois activités :
— La première, dite **"Main Activity"**, est la première activité que rencontre un utilisateur quand il lance l'application. Elle est composée d'un menu coulissant qui permet de prévoir la place pour les informations de contact, ainsi que de deux boutons qui permettent de lancer soit le téléchargement, soit la reconnaissance de carte.
— La **Second Activity** procède à l'affichage des cartes trouvées par l'OCR ou le CNN. Elle lance directement une **TextRecognitionActivity** en attendant un résultat de celle-ci via la méthode **onActivityResult**. Quand elle lui fournit une carte, elle l'affiche selon le format déterminé. De cet écran on peut aussi entrer un identifiant directement dans le champ de recherche en haut de l'écran. Si celui-ci correspond à une carte de la base de données, celle-ci sera affichée à la place de l'ancienne carte.
— La troisième activité, la **TextRecognitionActivity** affiche la caméra pour que l'utilisateur procède à la reconnaissance de cartes. Quand une carte est trouvée par l'OCR ou le CNN, un bundle contenant les informations de la carte est transmis à la seconde activité via la méthode **sendResult**.

6.2 Reconnaissance

L'implémentation principale de l'application est la reconnaissance de cartes en elle même, c'est-à-dire la façon dont les cartes sont reconnues et identifiées avec certitude.

Il y a trois étapes pour la reconnaissance :
— La première est l'extraction du texte d'une carte à l'aide d'un OCR, dans notre cas, le ML Kit fourni dans les services Firebase de Google. Il nous a permis très facilement d'extraire les blocs de lignes d'une carte, fidèles au caractère près.
— La deuxième partie consiste à trouver la carte à partir de l'ensemble des lignes extraites par l'OCR. Pour cela, notre algorithme vérifie que chaque ligne extraite existe dans la base de données, et si elle existe, combien d'identifiants de carte sont associés à cette ligne. Puisqu'une ligne peut être présente sur plusieurs cartes, nous classons dans un tableau associatif l'identifiant d'une carte et son score d'occurrence. Après vérification de toute les lignes, la carte avec le score strictement le plus élevé est retenue comme résultat de la reconnaissance. Malheureusement certaines cartes peuvent avoir exactement le même texte en tout point et seulement avoir une figure différente.
— La troisième, et dernière, partie est optionnelle, et est uniquement prévue pour ce genre de cas. Pour le résoudre, nous utilisons un réseau de neurones convolutif basé sur MobileNet, que nous avons préalablement entraîné uniquement sur les cartes qui posent ce problème et qui comportent donc généralement une figure différente. Ces

cartes sont identifiées dans la base de données par un champs *cnn* mis à **on**. Le résultat de la prédiction du CNN nous permet de trancher facilement sur ce problème.

Pour les cartes qui auraient exactement le même score, bien que leurs textes soient différents, on peut expliquer l'erreur par un défaut dans la lecture de l'OCR, par exemple le balaiement progressif de la carte par l'OCR. Dans ce cas, c'est-à-dire si la reconnaissance de carte obtient le même score pour deux cartes différentes, aucune des deux n'est choisie : le résultat est annulé et l'acquisition des lignes reprend depuis le début.

6.3 Classifieur CNN

Nous n'avons pas eu le temps de produire un script python avec Keras et TensorFlow permettant d'entraîner notre classifieur sur un jeu de données variables.

En effet, et comme évoqué précédemment, il doit être possible de rajouter des cartes dans la base de données. Hors, si ces cartes rentrent dans un des cas de conflit cités précédemment, l'application doit être capable de reconnaître et de différencier ces différentes cartes. Or notre classifieur n'est entraîné que pour un nombre limité de données et le ré-entraîner demande le travail d'un développeur.

Pour le moment, le modèle présent dans l'application est un prototype entrainé avec **Teachable Machine**[7] et qui reconnaît quatre classes, les cartes de la famille *e3dcm1* qui contiennent des figures.

6.4 Bugs connus

Il est possible, dans le cas de la rotation de la caméra pendant la reconnaissance de carte (c'est à dire pendant l'aperçu de la caméra) de provoquer un plantage de l'application. Cependant, étant donné la vitesse d'exécution de la reconnaissance, ce bug n'est pas considéré comme étant critique.

Il est aussi possible de provoquer un retour à l'écran de départ dans un cas précis. Pour reproduire ce bug, il faut arriver à accéder à la carte par défaut de l'application (voir figure 8). Un autre bug nous permettait d'y accéder facilement à partir de l'écran de reconnaissance de carte, mais ce premier bug ayant été résolu, il n'y a plus de moyen d'accéder à cette carte par défaut depuis l'application. Une fois sur cette carte par défaut, si on touche la flèche qui permet normalement de faire défiler les cartes, on provoque un écran blanc, avant de revenir à l'écran principal.

Cela ne peut pas être défini comme étant un comportement normal. Cependant, on remarquera que ce cas n'arrive que depuis la carte par défaut. En effet, la carte par défaut est une visualisation nécessaire de tous les éléments présent pendant l'activité de visualisation. On ne peut pas la supprimer, et on peut définir que la gestion d'une erreur impliquant une carte vide existe, sinon l'application planterai complètement, au lieu de renvoyer sur l'écran d'accueil. La solution semble donc de gérer l'apparition de la carte et non pas le contenu de la carte.

Cependant, il est impossible de rencontrer ce bug maintenant, puisque qu'il est impossible d'en remplir les conditions de reproduction.

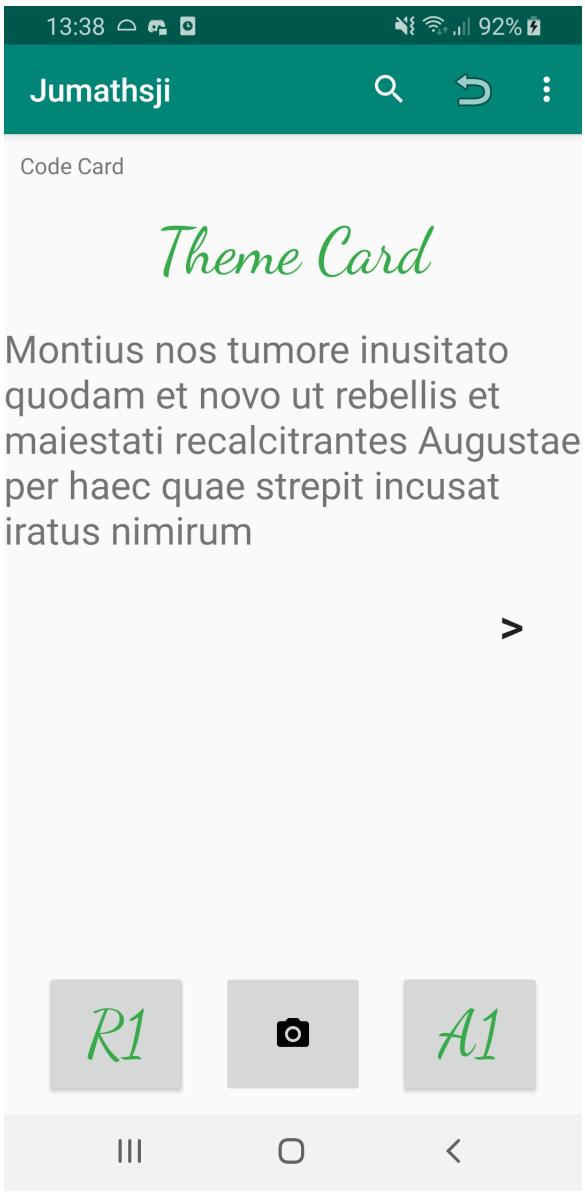

Figure 8 – Affichage par défaut d'une carte

7 Tests Effectués

7.1 Test du singe

Comme évoqué dans la partie sur les récits utilisateur, l'application doit être résistante à un enfant d'une dizaine d'année ayant pour but de la faire planter. Pour tester cette résistance, un test du singe semble tout indiqué.

Ce test à été effectué en trois temps : d'abord sur l'écran d'accueil de l'application, puis sur l'écran de reconnaissance de carte, et enfin sur l'écran affichage de carte.

Bien qu'il soit possible de réaliser un test du singe à partir de la SDK de Android, nous avons réalisé le notre sur un appareil android en conditions réelles.

7.1.1 Écran d'accueil

Sur l'écran d'accueil, on cherche à toucher autant d'éléments affichés à l'écran que possible. Pour cela, on touche toute la dalle tactile.

On remarque que toucher l'onglet à gauche de *Home* déclenche l'ouverture du menu coulissant qui doit par la suite contenir les informations de contact. Ce comportement est attendu. On remarquera ensuite que toucher n'importe lequel des champs disponible ne déclenche aucune activité, ni ne fait planter l'application. Toucher a de très nombreuses reprise ce menu le fait bouger et le fait parfois disparaître s'il est poussé jusqu'à la bordure gauche du téléphone. Là encore, ce comportement est attendu.

Ensuite, on peut se concentrer sur un appui répéter de la touche *Mettre à jour Jumathsji*. Le premier appui au lancement de l'application fait apparaître un texte avec la composition de la base de donnée téléchargée. Un appui sur ce texte ne déclenche rien. Un appui répété sur le bouton de mise à jour ne déclenche pas non plus de bug, ni de ralentissement.

Un appui sur l'onglet de paramètre en haut à droite permet de faire apparaître un menu contenant un seul objet : *Settings*. C'est de nouveau un place-holder. Une fois qu'une touche sur l'objet a été effectué, celui ci disparaît. C'est ce qui est attendu. Une répétition de cette séquence d'action (onglet de paramètre puis settings) continue de faire apparaître, puis disparaître l'objet settings, ce qui est le comportement attendu.

Enfin, un simple appui sur l'onglet caméra en bas à droite déclenche la reconnaissance de carte, ce qui est normal.

7.1.2 Écran de reconnaissance

Une fois arrivé sur l'écran de reconnaissance, on remarque qu'il est possible de rester indéfiniment sur cet écran si aucune carte n'est présentée devant la caméra. Si ce comportement n'est pas anormal en soit, il pourrait être intéressant cependant d'utiliser un minuteur qui indiquerait où entrer l'identifiant de la carte si ce minuteur arrivait à zéro.

Ensuite, si on reste dans ce cas de reconnaissance infinie, on remarque que chaque action définie dans le cadre de l'application se déroule selon les attentes. Les seules parties de l'écran réagissant sont la loupe déclenchant la recherche par identifiant, qui n'est pas active car non reliée à la fonction de recherche par manque de temps, la flèche de retour à l'écran

d'accueil qui a bien cette fonction, et le menu des options qui à le même comportement que sur l'écran d'accueil.

Nous avions un bug qui se présentait de la manière suivante : une fois arrivé sur l'écran de reconnaissance, faire arrière à l'aide du bouton du téléphone et tomber sur la carte par défaut de l'application. Cela été un bug en lui même. Mais il pouvait en entrainer un autre si la flèche permettant de faire défiler les cartes était appuyé. Dans ce cas, un écran blanc apparaissait avant que l'application ne revienne à l'écran d'accueil.

Ce problème a été réglé en modifiant le comportement du bouton de retour. En effet, nous n'en contrôlions pas le comportement au départ, laissant cela au téléphone. Dans un cas normal, le téléphone revient à l'activité précédente, dans l'état où elle à été quittée. Or, dans le cas dans lequel nous étions, l'application ne revenait pas sur l'écran d'accueil, pourtant défini dans le manifeste en tant que tel, mais emmenait l'utilisateur sur l'activité de visualisation.

Ne pouvant pas enlever l'activité de visualisation par défaut, la visualisation des cartes se basant dessus, il nous a suffit de contrôler le comportement du bouton de retour pour régler le problème.

Reste le problème de la liste de cartes qui renvoie un écran blanc. Ce bug là n'a pas été réglé.

On remarquera que même si la reconnaissance est déclenchée depuis plusieurs minutes, il n'y a pas de problème lors de la reconnaissance de la carte.

7.1.3 Écran de visualisation

Une fois sur l'écran de validation, on peut reprendre notre test.

Si on appuie de façon répétée sur les boutons *Réponse* ou *Aide*, on trouve que le comportement est semblable et normal : la fenêtre d'affichage apparaît à la pression et disparaît lors d'une seconde pression n'importe où sur l'écran.

Appuyer sur la loupe déclenche bien la zone de saisie de l'identifiant de la carte. Entrer un identifiant erroné ou une série aléatoire de caractère ne fait pas planter l'application. Entrer un code qui existe fait apparaître la carte.

Les zones du menu se comportent comme pour les autres écrans. Quand à appuyer sur le bouton caméra, il relance la reconnaissance de carte et donc l'écran correspondant, ce qui est un comportement normal.

Appuyer sur le bouton de retour de l'appareil renvoie bien sur l'écran d'accueil de l'application.

7.1.4 Conclusion du test

En conclusion, nous pouvons dire que dans les cas étudiés, notre application se comporte de façon attendue et qu'elle devrait donc se comporter normalement, même si utilisée par un élève déterminé à la faire planter.

A noter que ce test nous a permis de trouver le bug décrit plus haut.

7.2 Base de Données

Les Instrumented Test d'Android Studio permettent de tester des fonctionnalités de l'application en se mettant directement dans le contexte de celle-ci. Cela est nécessaire lorsqu'on veut utiliser les méthodes de la base de données. En effet, l'instance de la base de données est propre au contexte dans laquelle elle est initialisée, et le contexte requit est celui de l'application. Les tests unitaires sur les classes de la base de données sont situés dans la classe DataBaseUnitTest. Ils vérifient que l'insertion et l'extraction d'élèments, ainsi que la recherche par identifiant, fonctionnent correctement.

Concernant l'intégrité des élèments de la base données, un test d'unicité des identifiants a été mis en place dans la classe **DatabaseTest**. La vérification se fait directement sur la base de données téléchargée et construite avec les méthodes de l'application. De cette façon on s'assure du même coup du bon fonctionnement de celles-ci. Le test procède alors aux étapes suivantes :

— Partant de la base de données complète, une liste de toutes les cartes est extraite.
— Une liste qui contiendra tous les identifiants passés en revue est initialisée sans éléments.
— Pour toutes les cartes, on vérifie que leur identifiant n'est pas présent dans la liste d'identifiants, et si tel n'est pas le cas on l'ajoute dans celle-ci. Cette condition est vérifiée par une assertion, qui, si elle n'est pas valide, écrit l'identifiant dupliqué en message d'erreur dans la console.

D'autres tests de vérification du jeu de données ont été demandés, mais n'ont pas pu être réalisés dans les temps (tests de complétude décrits dans la partie "Test utilisateurs" de la section "Récits Utilisateur").

7.3 Tests en conditions réelles

L'application a pu être testée sur plusieurs appareils. Voici les temps de reconnaissance d'une carte sur trois téléphones et deux tablettes.

Table 1 – Temps de reconnaissance d'une carte par modèle d'appareil
(temps estimé par un utilisateur muni d'un chronomètre, avec une exposition à la lumière et une orientation de la carte optimales)

modèle	Pixel 3a	Samsung Galaxy S8	CUBOT R11	Galaxy Tab S5e	Galaxy Tab A 2019
temps	<1s	<1s	1-5s	<1s	<1s

Le CUBOT R11 a été notre appareil de test pendant la phase de développement. Il a fournit un seuil de performance minimum a atteindre pour l'application.

Android Studio fournit aussi un outil de profiling. On peut relever que l'application demande un maximum de 400 MO de RAM au démarrage, puis descend à 100 MO en moyenne en cours d'utilisation, n'allant plus au dessus de 128 MO.

7.4 Similarité de la base de données

Comme expliqué auparavant, il est intéressant de tester la similarité de notre base de données de façon à savoir exactement combien de cartes peuvent être trop similaires pour

la reconnaissance de caractères, et précisément lesquelles. Pour cela, nous avons deux options : utiliser une comparaison naïve des textes, qui compare mot à mot chaque texte, qui calcule un pourcentage de mots en communs et qui retourne ce pourcentage ; ou alors faire une comparaison plus poussée avec une similarité cosinus.

Étant donné qu'on ne cherche pas à analyser un texte long, et qu'on ne cherche pas non plus à en comprendre le sens, mais juste à faire une comparaison, on se concentrera sur la comparaison naïve mot-à-mot.

Le test s'effectue de la façon suivante :

— On commence par récupérer les textes des deux cartes à comparer sous forme de tableaux. Chaque chaîne de caractère correspondant à un mot a donc son indice dans son tableau respectif. Les caractères de ponctuations, points et virgules par exemple, ne sont pas comptés comme des mots à part et sont comptés comme faisant partie du mot qui les précèdent. Cela veut dire que, par exemple, les mots "sine", "sine." et "sine," sont considérés comme étant trois mots différents. Dans le contexte de notre application, la différence n'est pas importante, puisque la ponctuation fait partie des caractéristiques texte.
— On cherche ensuite quel est le texte le plus court et cela dans le but de ne pas effectuer de dépassements de tableau. On aurait pu augmenter la taille du tableau le plus court (c'est à dire le texte le plus court) en considérant que les indices supplémentaires contiennent des chaînes de caractères nulles. Cependant, cela ne change rien au résultat final.
— Ensuite, on fait une comparaison indice par indice des deux tableaux. Si deux chaînes de caractères ayant le même indice sont égales, on incrémente un compteur. Ce compteur correspond au nombre total de mots identiques et placés au même endroit dans les deux textes.
— Une fois le plus court tableau complètement parcouru, on regarde le taux de similarité en divisant le compteur final par le nombre maximum de mots dans un texte. Le résultat de ce calcul est renvoyé sous forme de pourcentage.
— Si le taux de similarité est plus élevé qu'un plafond défini à l'avance (85% et 90% pour les tests effectués), on retourne les identifiants des cartes qui sont trop similaires et qui pourraient poser problème.

Pour tester la méthode, et afin de garantir la véracité des tests effectués sur la base de données, des tests unitaires on été effectués.

En premier lieu, on vérifie que les textes ne sont pas vides. Puis on passe à des comparaisons à l'aide de *Lorem ipsum*. Les textes de tests sont les suivants :

1. *"Lorem ipsum dolor sit amet."*, une phrase complète de cinq mots avec un point à la fin du mot *amet*. Cette phrase nous servira de référence.
2. *"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent eu."*, deux phrases pour un total de dix mots. Les cinqs premiers sont strictement identiques.
3. *"Lorem ipsum dolor amet sit."*, une phrase semblable à la première, mais avec les deux derniers mots inversés.
4. *"Lorme ipsum dolor sit amet, consectetur adipiscing elit. Praesent eu."*, deux phrases très semblables aux phrases du deuxième exemple. On remarquera cependant que le premier mot, *"Lorem"* est mal orthographié dans ce quatrième exemple. Il doit donc compter comme un mot différent.

5. *"Lorem ipsum ipsum sit amet."*, une phrase de cinq mots presque égale au premier exemple, si on exclu le troisième mot *ipsum*, au lieu de *dolor*.

Lors des tests, on s'attend à ce que comparer chaque cas à lui même renvoie une similarité de 100%. Par contre, pour les autres cas, la similarité ne peut jamais être de 100%. Par exemple, en comparant l'exemple 1 à l'exemple 3, on s'attend à une similarité de 60%. En effet, ces deux textes partagent trois mots se situant dans les mêmes positions dans leur texte respectifs. On obtient donc le calcul $(3/5) * 100 = 60$. En comparant les exemples 2 et 4, on s'aperçoit que seul un mot diffère à cause de deux lettres inversées. On s'attend donc à ce que le résultat soit 90%.

Comme on peut le voir dans notre implémentation, notre méthode passe les tests. On peut donc maintenant comparer les cartes dans la base de données.

Étant donné que notre base de données n'est pas complète, nos tests ne porteront que sur la similarité des cartes à l'intérieur du même fichier YAML. On ne comparera pas deux cartes qui se situeraient dans deux fichiers différents.

Nous avons commencé par tester le fichier *"e3dcm1.yaml"*, d'abord pour 85% de similarité puis 90%.

En effectuant le test sur ce premier fichier et pour un taux de similarité maximum de 85%, et comme présenté en 9, on s'aperçoit qu'il existe 5 cas dans lesquels la similarité des cartes est au dessus du taux maximum.

```
Verifying cards are not above 85.0% similar
Starting card similarity verification of the database
Cards directory successfully opened
Verifying file : e3dcm1.yaml
e3dcm1q_11 e3dcm1q_18
e3dcm1q_5 e3dcm1q_18
e3dcm1q_13 e3dcm1q_20
e3dcm1q_2 e3dcm1q_3
e3dcm1q_1 e3dcm1q_3
There is (are) 5 similarity(ies)
```

Figure 9 – Résultat de la comparaison des cartes de e3dcm1.yaml pour une similarité maximum de 85%

On peut donc voir que, si une similarité supérieure à 85% est considérée comme problématique, on a cinq cas de similarité. Par exemple, on remarquera que *e3dcm1q_18* est très similaire à *e3dcm1q_5* et *e3dcm1q_11*. Par contre, ces deux cartes ne sont pas similaires entre elles.

Maintenant, si on fait le test pour une similarité de 90%, on obtient le résultat présenté en 10.

```
Verifying cards are not above 90.0% similar
Starting card similarity verification of the database
Cards directory successfully opened
Verifying file : e3dcm1.yaml
e3dcm1q_2 e3dcm1q_3
There is (are) 1 similarity(ies)
```

Figure 10 – Résultat de la comparaison des cartes de e3dcm1.yaml pour une similarité maximum de 90%

Avec une similarité maximale de 90%, il ne reste plus que deux cartes similaires, les cartes *e3dcm1_2* et *e3dcm1_3*. En regardant le texte de ces deux cartes, on remarque que pour les deux cartes, le texte est identique : *"Combien y-a-t-il de cubes dans cette pyramide ?"*. On peut donc conclure qu'avoir recours à l'identification par image va être nécessaire pour ces cartes là.

Bien sûr le taux de similarité maximum toléré par l'application est à la discrétion du propriétaire de l'application.

8 Conclusion

Comme nous l'avons vu dans ce projet, notre application remplit les demandes principales du client :

- L'application se présente sous la forme du prototype d'une application Android permettant de télécharger les fichiers YAML composant notre base de données.
- Ces fichiers contiennent une partie des cartes permettant de jouer au jeu JUMATHSJI, en particuliers les cartes "Espaces 3D" pour CM1 et "English Maths" pour CM2.
- Les cartes physiques peuvent être reconnues à l'aide d'un système de reconnaissance optique de caractère qui extrait les blocs de lignes avant de rechercher la carte.
- Dans le cas où un texte serait présent dans plusieurs cartes, l'application lance un réseau de neurones qui permet de prédire quelle carte est la bonne et de trancher le problème.
- Afficher la carte trouvée avec son titre, son texte et son image si celle-ci existe. Cette affichage, et les images, correspondant en tout point aux cartes réelles.
- Afficher une aide, si celle ci-existe, ainsi que la réponse à la question, à la demande.
- Chercher une carte en fonction de son identifiant unique.
- Si plusieurs cartes ont été trouvées et que l'application n'a pas été capable de les départager, l'utilisateur à la possibilité de choisir parmi les cartes trouvées.
- Relancer la recherche de carte par l'OCR après qu'une carte ait été trouvée.

On notera aussi qu'il est possible de chercher à quel point des cartes sont similaires en comparant leur texte énoncé et ce dans l'optique de pouvoir anticiper la nécessité d'avoir recours à un second moyen d'identification pour certaines cartes.

On finira par noter qu'un début de base de données existe, et ce afin de définir quels paramètres doivent être inclus afin que chaque carte soit considérée comme complète.

Parmi les pistes possibles pour l'amélioration de l'application, on voir trois axes principaux :

- Finir les fonctionnalités manquantes. On peut penser à la fonction de recherche par identifiant qui n'a pas été implémenté du fait de la suspension des projets, par exemple.
- L'amélioration de l'interface. En effet, l'application présentée ici n'est qu'un prototype qui n'est pas destiné à être utilisé par le public. En particulier, on peut pointer l'aspect peu accueillant de l'interface, qui n'a pas été travaillé vu son aspect peu prioritaire. On notera aussi le nombre d'actions nécessaire pour lancer la reconnaissance au lancement de l'application.
- L'amélioration de la base de données. Cet axe là peut être divisé en deux sous-axes :
  1. Le premier est l'insertion des réponses et des aides dans la base de données déjà existante. En effet, les textes composants les aides et les réponses ne nous ont pas été fournis.
  2. Le deuxième est la création d'un outil permettant de créer facilement de nouvelles cartes sans passer par le langage YAML. En effet, malgré la simplicité de YAML, il serait plus agréable pour un utilisateur étant enseignant et souhaitant créer ses propres cartes (de géographie ou d'histoire, par exemple) d'utiliser une interface où les champs nécessaires à la création de carte seraient déjà indiqués. On pourrait réaliser cette interface sous forme de site web, par exemple, ce qui supprimerait un possible problème de compatibilité matérielle. Bien entendu, il faudrait aussi qu'il soit possible d'envoyer les jeux de données ainsi créés sur le serveur où se

trouve le reste des données.

Beaucoup de ces améliorations sont cependant hors de notre sujet et seront poursuivies lors du stage faisant suite à ce PFE.

9 Annexes

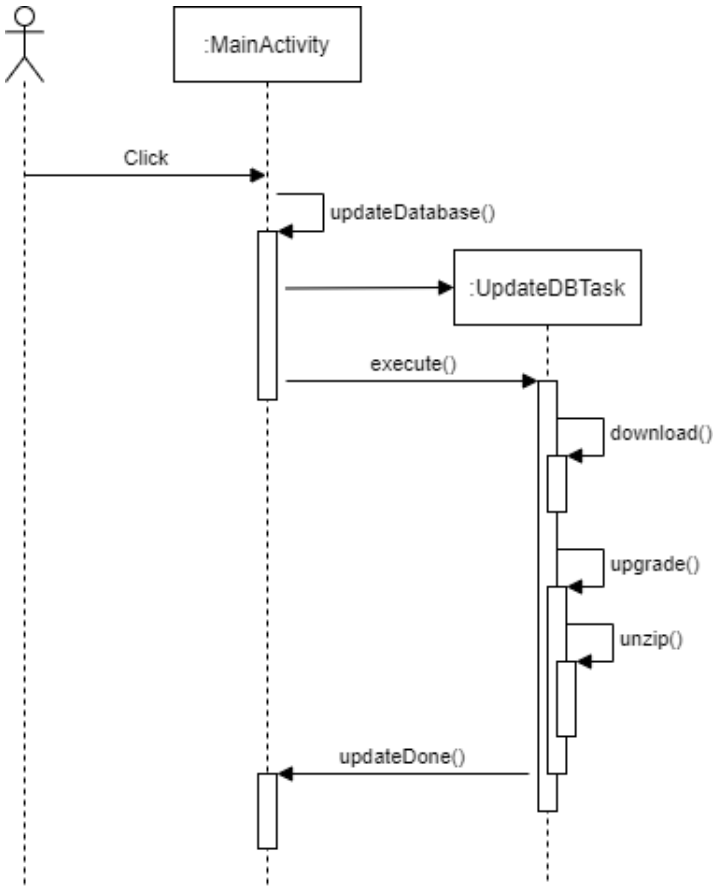

Figure 11 – Diagramme de Séquence : Mise à jour de la Base de Donnée

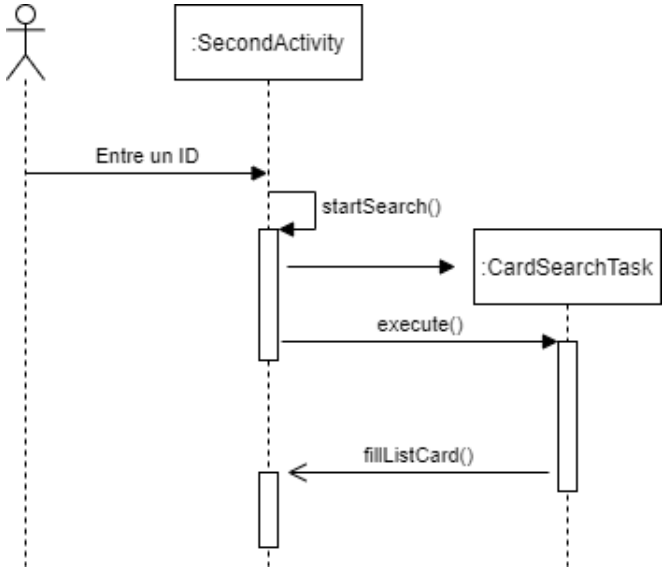

Figure 12 – Diagramme de Séquence : Identification d'une carte par son numéro

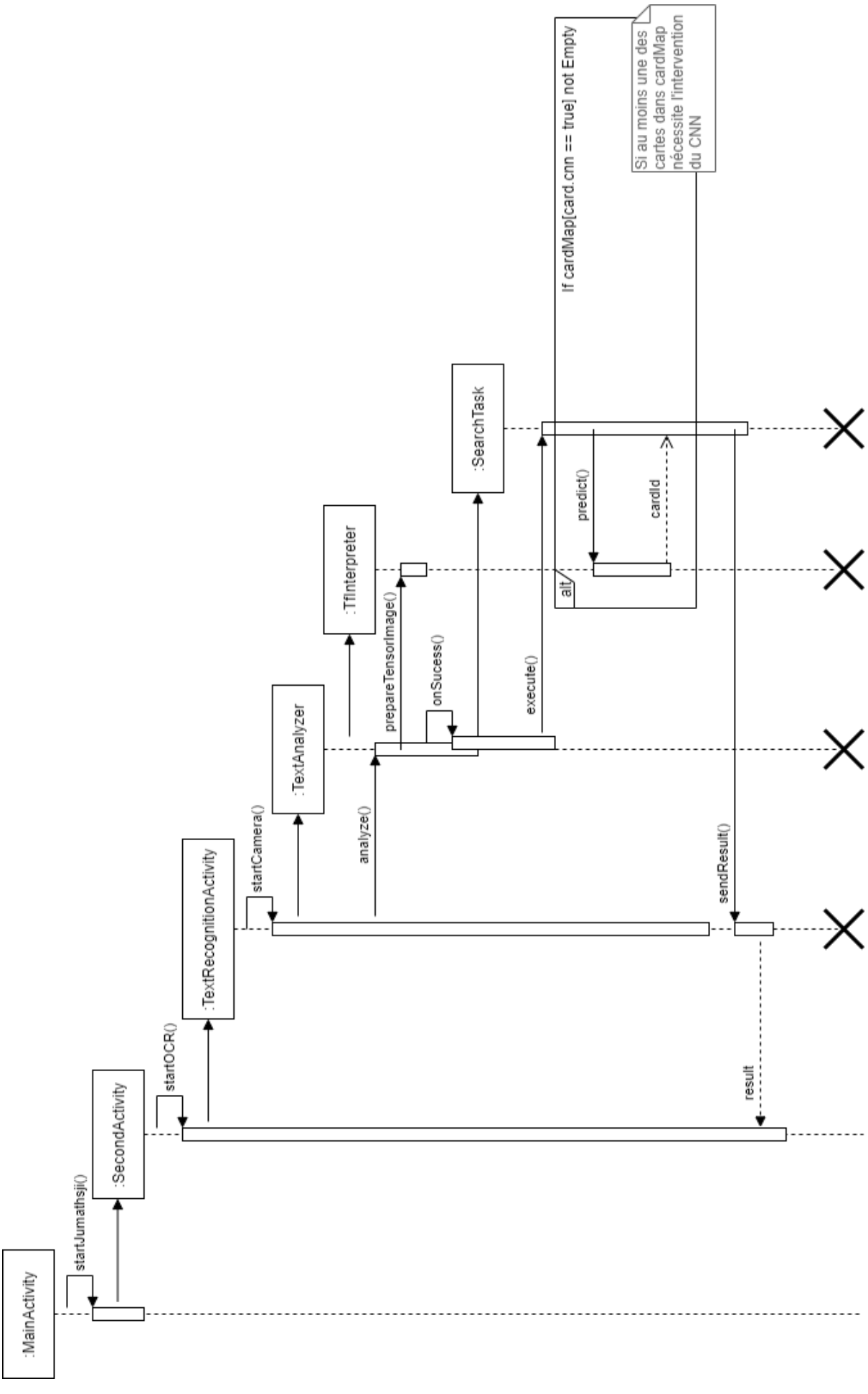

FIGURE 13 – Diagramme de Séquence : Reconnaissance et Identification d'une carte avec la caméra

Références

[1] “Jumathsji, le complément numérique du jeu plateau et réseau.” [Online]. Available : https ://jumathsji.glideapp.io/

[2] “Ar+ est désormais disponible dans pokémon go !” [Online]. Available : https ://pokemongolive.com/fr/post/arplus/

[3] “Text recognition.” [Online]. Available : https ://library.vuforia.com/articles/Training/Text-Recognition-Guide

[4] “tesseract-ocr.” [Online]. Available : https ://github.com/tesseract-ocr/tesseract

[5] “Ml kit for firebase.” [Online]. Available : https ://firebase.google.com/docs/ml-kit/

[6] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets : Efficient convolutional neural networks for mobile vision applications,” 2017.

[7] “Teachable machine.” [Online]. Available : https ://teachablemachine.withgoogle.com/