

UNIVERSITE DE HAUTE-ALSACE

Manuel développeur

Application Android Rallye Miage Mulhouse

Sujet proposé par M. Maillot et M. Cordier

Xavier FREYBURGER, Jean-Marc GROSS, Thomas KIRBIHLER, Franck PARRA, Gauthier SCAMPINI,
Mickaël BRENOIT



SOMMAIRE

0.	Préambule	2
1.	Environnement de développement	3
1.1.	Android Studio.....	3
1.2.	Genymotion.....	4
2.	Organisation des vues	5
3.	Structure d'un fichier XML propre à un circuit.....	6
3.1.	Les nœuds principaux.....	8
3.1.1.	Circuit	8
3.1.2.	Lieu	8
3.1.3.	Validation	9
3.1.4.	Question	10
3.1.5.	Réponse.....	11
3.2.	Les nœuds secondaires	11
3.2.1.	Description	11
3.2.2.	Information.....	12
4.	Structure du projet Android et fonctionnement global	13
4.1.	Classes liées aux données propres à un circuit	13
4.1.1.	Chargement d'un rallye dans la mémoire	13
4.2.	Organisation des classes.....	14
4.2.1.	Package « model ».....	14
4.2.2.	Package « services »	18
4.2.3.	Package « utilities »	19
4.2.4.	Les activités	20
4.3.	Fichiers de sauvegarde	29
4.4.	Communication avec le serveur	30
5.	Jouer à un rallye en local	31
6.	Publier son application sur Google Play Store	33
7.	Charte graphique.....	36
7.1.	Le nom	36
7.2.	Le choix des couleurs.....	36
7.3.	Le logotype	36
7.4.	Images et icônes	36

0. Préambule

Le projet “Rallye” a été initié en 2014 par M. MAILLOT et M. CORDIER, tous deux enseignants à la MIAGE de Mulhouse.

Une première partie du développement du site Web et de l’application Smartphone a été réalisée durant le mois de mai 2014 par Xavier FREYBURGER et Jean-Marc GROSS dans le cadre de leur formation en M1 MIAGE.

Le développement global a ensuite été repris l’année suivante, durant le mois de mai 2015 par Gauthier SCAMPINI (au niveau du site Web) et par Franck PARRA et Thomas KIRBIHLER (au niveau de l’application Smartphone), une nouvelle fois dans le cadre de leur formation en M1 MIAGE. Cela a notamment permis d’apporter un certain nombre de corrections, de modifications et d’améliorations à l’application, telles que :

1. l’amélioration de l’ergonomie sur la plupart des vues
2. une refonte du système de navigation à travers les différentes vues de l’application, notamment via un accès constant à la liste des étapes
3. la validation automatique d’une étape et tout ce qui y est relaté
4. corrections de bugs divers

Par la suite du début mai à fin juillet 2016, le développement de l’application Smartphone et du site web a été repris par Mickaël BRENOIT dans le cadre d’un stage proposé par l’Université de Haute-Alsace. Les apports effectués pendant cette période ont été la refonte complète du fonctionnement de l’application mobile. En effet, maintenant il est possible de faire un rallye sans connexion internet. De plus, les données sur la position des équipes sont envoyées de manière plus ou moins fréquentes si elles sont proches du lieu à trouver. Enfin, un système d’indice a été rajouté.

Il est fortement conseillé de consulter les informations liées aux principes généraux du fonctionnement d’un rallye dans la FAQ disponible sur le site.

Il est nécessaire d’avoir acquis les compétences minimales et nécessaires pour développer une application Smartphone (même simple). Ces notions nous ont en effet été enseignées durant notre formation en M1 MIAGE.

Veuillez aussi consulter au préalable le manuel utilisateur de l’application Smartphone pour comprendre plus profondément comment fonctionne et a été pensée l’application.

Enfin, une partie conséquente des fonctionnalités de l’application Smartphone sont très étroitement liées à celles du site Web. Veuillez donc consulter les documentations liées au site Web pour une meilleure compréhension des nombreuses interactions entre celui-ci et l’application Smartphone.

1. Environnement de développement

Le développement de l'application sur Smartphone utilise la technologie Android avec le langage Java. Les classes spécifiques à la programmation sur Smartphone sont fournies par le Source Development Kit (SDK) Android.



1.1. Android Studio

L'outil de développement *Android Studio* (version 2.1) est un outil spécialement dédié au développement d'application Android. Il s'avère donc être l'outil central utilisé tout au long du développement de l'application.

L'édition du code source, la compilation et toutes autres modifications peuvent être aisément réalisées grâce à cet outil. Ce dernier permet de créer tous les fichiers nécessaires, de modifier les sources, de tester le code, et bien sûr de créer à la volé les interfaces graphiques des différentes vues de l'application.

Les habitués de l'autre outil de développement phare pour Android, j'ai nommé Eclipse, ne devrait pas être dépayés. En comparaison avec ce dernier, nous avons en effet rencontré beaucoup moins de bugs pendant l'utilisation d'*Android Studio*, une fois l'outil correctement installé.

L'installation se réalise relativement simplement en suivant les instructions sur le site dédié <https://developer.android.com/sdk/index.html>. Veuillez notamment prendre en compte les indications liées aux versions de JDK, nécessaires au bon fonctionnement du logiciel.

Concernant l'importation d'un projet existant, suite à nos expériences dans la version d'*Android Studio* que nous avons utilisée, nous conseillons d'effectuer cette importation via la fonction "Files/Open..." (en sélectionnant le dossier correspondant au projet voulu), et non pas "Files/Import".

1.2. Genymotion

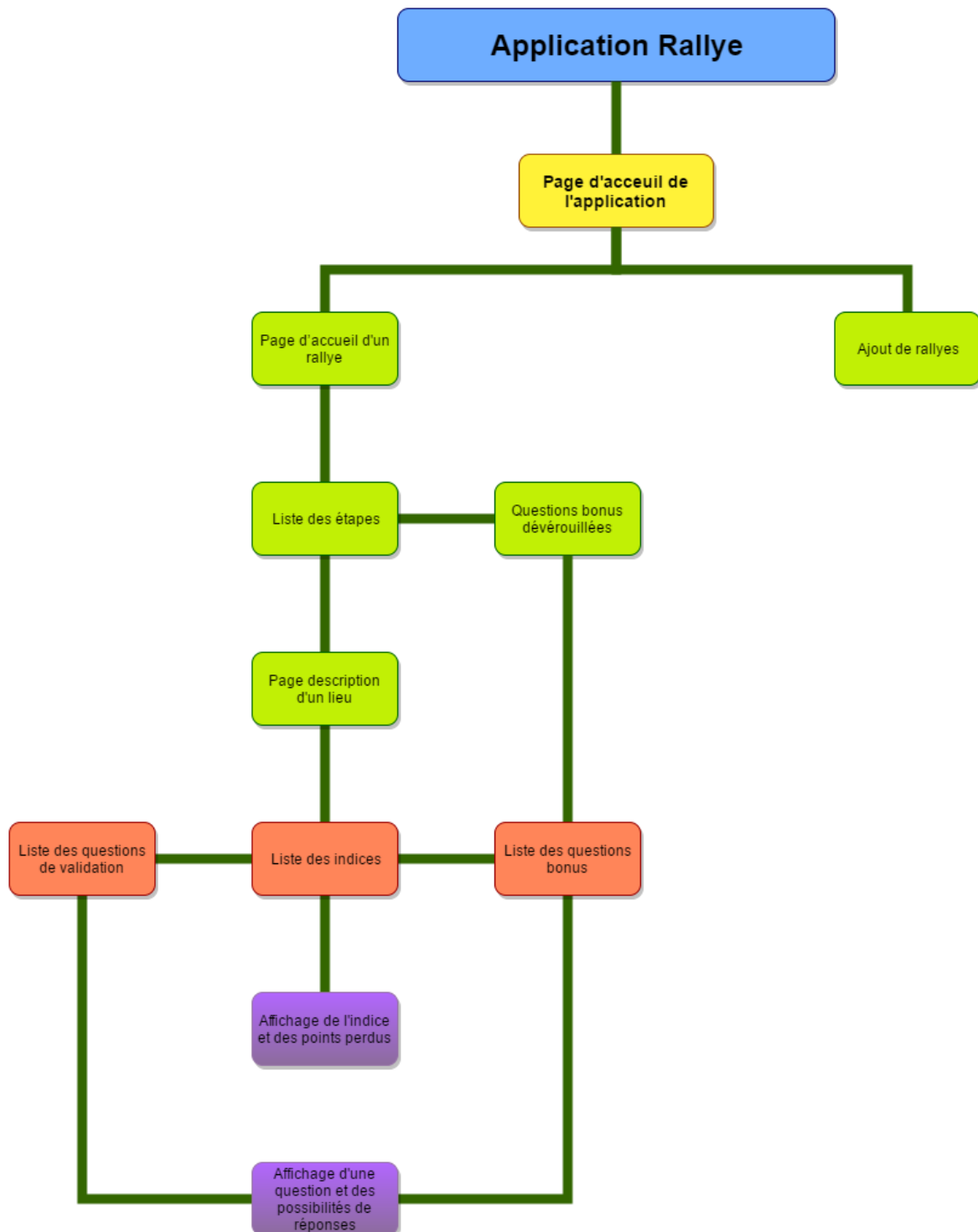
Si votre ordinateur dispose d'un processeur Intel, vous ne devriez pas rencontrer de problème quant à l'utilisation de l'un des émulateurs de Smartphone intégrés à Android Studio.

Cependant, il est très fortement conseillé pour les possesseurs d'un processeur AMD (ou simplement conseillée pour tout autre développeur) d'utiliser un des machines virtuelles fournies par l'éditeur Genymotion, donnant accès à des émulateurs particulièrement complets et fluides.

L'installation et l'association de Genymotion à Android Studio n'est pas spécialement compliquée à réaliser. Suite à l'inscription puis le téléchargement de Genymotion via le site Web dédié (<http://www.genymotion.com/>), et après installation de ce dernier sur votre ordinateur, il faut l'ajouter dans la liste des plug-in externes à Android Studio (Files → Settings → Plugins) puis cliquer sur l'icône approprié dans la barre des menus dans l'interface d'Android Studio.

2. Organisation des vues

Lorsque l'utilisateur lance l'application Rallye, la page d'accueil de l'application apparaît. Le schéma ci-dessous permet de décrire l'organisation des vues de l'application.



L'aperçu du contenu de chacune de ces vues et leur utilisation est consultable dans le manuel utilisateur de l'application.

3. Structure d'un fichier XML propre à un circuit

```
<?xml version="1.0" encoding="utf-8"?>
<circuit id=" " type=" ">
  <url> </url>
  <equipe> </equipe>
  <titre> </titre>
  <description>
    <text></text>
  </description>
  <difficulte></difficulte>
  <lieux>
    <lieu id="">
      <titre> </titre>
      <description>
        <text></text>
        <media src=" " type=" "/>
      </description>
      <validation type=" ">
        <coordonnees precision="">
          <latitude></latitude>
          <longitude></longitude>
        </coordonnees>
        <questions EstLibre=" ">
          <question id="">
            <intitule> </intitule>
            <reponses EstLibre="">
              <choix bonneReponse=""></choix>
              <choix bonneReponse=""></choix>
            </reponses>
          </question>
        </questions>
        <point></point>
      </validation>
      <indices>
        <indice id="">
          <libelle></libelle>
          <description>
            <text></text>
            <media src=" " type=""/>
          </description>
          <points></points>
        </indice>
      </indices>
    </lieu>
  </lieux>
</circuit>
```

```
<questions EstLibre="">
  <question id="">
    <intitule></intitule>
    <point></point>
    <reponses EstLibre="">
      <choix bonneReponse=""></choix>
      <choix bonneReponse=""></choix>
    </reponses>
  </question>
</questions>
<informations/>
</lieu>
</lieux>
</circuit>
```

Nous allons maintenant vous expliquer plus en détail les choix de cette organisation, et le contenu de chaque nœud.

3.1. Les nœuds principaux

3.1.1. Circuit

Le nœud Circuit est le nœud de base de notre structure. Il permet de décrire grâce à ses attributs et à ses descendants la totalité d'un rallye.

3.1.1.1. Attributs

Le nœud circuit possède deux attributs :

1. un attribut « id » servant à identifier de façon unique le rallye ;
2. un attribut « type » servant à spécifier le type du rallye. Ce dernier peut valoir « rallye » ou « circuit ». Dans le premier cas cela signifie que le fichier décrit un rallye classique, et dans l'autre cas, cela signifie qu'il décrit un circuit touristique. Dans le cadre d'une reprise de ce projet, cet attribut pourra directement servir à créer d'autres types de circuits.

3.1.1.2. Descendants

Le nœud circuit possède obligatoirement quatre descendants :

1. une balise « url » contenant l'adresse http du site web ;
2. une balise « equipe » contenant l'identifiant de l'équipe qui a servi à télécharger ce circuit
3. une balise « titre » servant à stocker le nom du rallye ;
4. un nœud « description » contenant la description du rallye ;
5. une balise « difficulté » contenant un entier entre 0 et 5 servant à décrire la difficulté du rallye ;
6. et enfin un nœud « lieux » contenant la description complète de toutes les énigmes.

Il est à noter que ces nœuds peuvent être des nœuds vides (par exemple <description />) mais ils doivent obligatoirement être présents pour que le fichier XML soit accepté par l'application Smartphone.

3.1.2. Lieu

Un lieu correspond à un lieu géographique à découvrir dans le cas du rallye, et doit donc contenir les informations nécessaires à proposer une énigme à l'utilisateur, ou un lieu

intéressant dans le cas d'un circuit touristique. Pour cela, nous avons décidé d'organiser un lieu de la façon suivante :

3.1.2.1. Attributs

Le nœud lieu possède un seul attribut : « id » servant d'identifiant unique à ce lieu dans le programme et le site web.

3.1.2.2. Descendants

Un lieu possède obligatoirement cinq descendants :

1. une balise « titre » servant à donner un nom à ce lieu. Ce nom fera souvent partie de l'énigme dans le cas d'un rallye ;
2. un nœud « description » servant à contenir l'énoncé de l'énigme ;
3. un nœud « validation » permettant de décrire la façon dont la validation de l'énigme devra être effectuée. Ce nœud peut être un nœud vide ce qui signifiera qu'il n'y a pas d'énigme à résoudre (cas d'un circuit touristique par exemple) ;
4. Un nœud « indices » contenant les indices. Ils peuvent posséder un texte et/ou des images et un nombre de points à retirer à l'étape s'il est dévoilé. Ce nœud peut être vide s'il n'y a pas d'indices à dévoiler (cas d'un circuit touristique par exemple) ;
5. un nœud « questions » contenant la description complète des questions bonus liées à ce lieu. Ce nœud peut être vide ce qui signifiera qu'il ne possède pas de question bonus ;
6. et enfin un nœud « informations » qui peut être utilisé par un circuit touristique pour contenir des informations subsidiaires que l'on souhaiterait transmettre à l'utilisateur.

3.1.3. Validation

Le nœud validation a pour but de décrire précisément de quelle façon l'application devra valider ou non de la résolution de l'énigme par l'utilisateur. Nous avons choisi de gérer deux types de validations : une validation par question, et une validation par position GPS. Aucune des deux n'est obligatoire, et l'on peut utiliser les deux en même temps si nécessaire. Pour cela on utilise l'attribut « type » de la validation qui peut soit contenir :

1. « position » signifiant que la validation s'effectue par positionnement GPS du Smartphone ;
2. « question » signifiant que la validation s'effectue par réponses à des questions ;

3. « position,question » signifiant que la validation s'effectue par la combinaison des deux précédents.

Dans le cas du développement d'un nouveau type de validation, il sera nécessaire de le spécifier au moyen de cet attribut « type ».

3.1.3.1. La validation par position GPS

Dans le cas où l'attribut « type » de la validation contient l'argument « position », le nœud « coordonnees » doit obligatoirement être présent dans la validation. Ce nœud a pour but de décrire les coordonnées GPS du lieu à découvrir ainsi que le rayon autour du lieu dans lequel on considère que l'utilisateur est au bon endroit.

3.1.3.1.1. Attribut

Ainsi le nœud « coordonnees » utilise un attribut « precision » contenant un entier, précisant la distance (en mètres), à laquelle l'utilisateur peut se trouver pour considérer qu'il a réussi à résoudre l'énigme.

3.1.3.1.2. Descendants

Le nœud « coordonnees » possède deux descendants « latitude » et « longitude » contenant respectivement la latitude et la longitude du lieu à découvrir en degrés.

3.1.3.2. La validation par questions

Si l'attribut « type » du nœud validation contient l'argument « question », alors le nœud devra contenir un nœud « questions » contenant les différentes questions auxquelles l'utilisateur devra répondre.

3.1.4. Question

Le nœud « question » a pour but de contenir les informations nécessaires à la description d'une question et des réponses possibles.

3.1.4.1. Attributs

Le nœud « question » contient :

1. l'attribut « id » permettant d'identifier de façon unique une question, que ce soit dans l'application, ou dans le site web.
2. l'attribut « estLibre » permettant de savoir si la question est à réponse libre ou non.

3.1.4.2. Descendants

Le nœud question possède obligatoirement deux descendants :

1. une balise « intitulé » contenant l'intitulé de la question ;
2. un nœud « reponses » pouvant, soit contenir les réponses si la question est de type choix parmi propositions, soit être un nœud vide si la question est de type « réponse libre ».

3.1.5. Réponse

Le nœud « reponse » a pour but de décrire les différentes réponses et de spécifier laquelle ou lesquelles sont des bonnes réponses.

Pour cela il contient un descendant « choix » contenant l'intitulé de la réponse, et possédant un attribut de type booléen « bonneReponse » spécifiant par vrai si cette réponse est juste, et par faux le cas contraire.

Dans le cas d'une question libre, ce nœud ne contient une seule et unique occurrence d'un descendant « choix », dont le texte est la bonne réponse à la question libre.

3.2. Les nœuds secondaires

3.2.1. Description

Le nœud « description » fournit une description dont le contenu peut être relativement libre, comme pourrait le vouloir un organisateur de rallye souhaitant créer une énigme en se basant sur d'autres types de médias que le simple texte.

Pour l'instant seul le texte simple et le média de type image sont gérés par notre application. Mais d'autres types de médias tels que les vidéos ou l'audio pourrait être ajoutés en gardant notre structure dans le cas de la poursuite de notre projet.

3.2.1.1. Le texte

Pour ajouter un contenu de type texte, une simple balise « texte » est utilisée avec comme contenu le texte à afficher.

3.2.1.2. Les médias

Le nœud « media » a pour but de contenir un média externe qui sera chargé par l'application. Pour l'instant seul les images sont gérées mais la conception de ce nœud s'est voulue évolutive.

Ainsi si ce nœud décrit une image, l'attribut « type » devra le spécifier avec l'argument « image », et l'attribut « src » devra alors contenir la source de l'image (une adresse web).

3.2.1.3. Utilisation

Les nœuds « texte » et « media » peuvent être utilisés successivement autant de fois que l'organisateur du rallye le souhaite pour créer des descriptions véritablement personnalisées.

3.2.2. Information

Le nœud « informations » n'est utilisé que dans le cas d'un circuit touristique, et remplace les questions bonus par un contenu plus riche.

En effet les informations ne contiennent pas uniquement des intitulés textuels comme les questions, mais un nœud « description » permettant une plus grande liberté de contenu et un affichage plus adapté que celui des questions-réponses.

4. Structure du projet Android et fonctionnement global

Tout le code au sein du projet Android est accompagné de commentaires explicites expliquant clairement le principe des différentes fonctions utilisées, des variables ou encore des algorithmes implémentés.

De plus, une convention de nommage a été rajoutée par la suite. Les commentaires ainsi que le nom des méthodes sont écrits en anglais pour un code compréhensible par le plus grand nombre.

Pour les noms des méthodes étant composées de plusieurs mots, il faut savoir que la première lettre du premier mot est en minuscule et à chaque nouveau mot nous le commençons par une majuscule. Exemple : « goodExample ».

4.1. Classes liées aux données propres à un circuit

Les différentes classes utilisées ici ayant la vocation à être sauvegardées à la volée, elles implémentent toutes l'interface « Serializable », afin de pouvoir les écrire, ainsi que les lire très simplement.

4.1.1. Chargement d'un rallye dans la mémoire

Afin de charger efficacement le fichier XML décrivant un rallye dans la mémoire du Smartphone, nous avons créé l'équivalent des nœuds XML en classes. Le principe étant très proche, nous évoquerons uniquement les spécificités de chaque classe, les bases étant décrites dans la description du fichier XML à la section **Erreur ! Source du renvoi introuvable...**

4.2. Organisation des classes

4.2.1. Package « model »

4.2.1.1. Circuit

La classe « Circuit » permet de stocker toutes les informations d'un circuit : son id, son type, son titre, sa date de fin et sa difficulté.

La description est stockée dans un tableau de Description afin de stocker dans l'ordre toutes les descriptions chargées lors du parsing de l'XML.

Les lieux sont stockés dans un tableau de Lieu dans le même ordre que dans le fichier XML.

Enfin le circuit possède un attribut userId contenant l'identifiant de l'équipe qui a été utilisé lors du téléchargement de l'XML, et un attribut adresse contenant l'adresse du site web sur lequel les données seront envoyées.

Pour ce qui est des méthodes :

1. « nbPoints » : retourne un entier. Cette méthode retourne le nombre de points gagnés des étapes de chaque lieu.
2. « nbBonusPoints » retourne un entier. Cette méthode retourne le nombre de points gagnés par rapport aux questions bonus de chaque lieu.
3. « toString » redéfinition de la méthode toString. Cette méthode retourne une chaîne de caractères avec la date de fin du circuit si ce dernier est terminé. Pour chaque lieu, on retourne les points de validation de l'étape, des questions bonus et ceux obtenus au total.
4. « nbPlaces » retourne un entier. Cette méthode retourne le nombre de lieux qu'est composé le rallye.
5. « getNbRallyPoints » retourne un entier. Cette méthode retourne la somme des nombre de points maximum de chaque étape que l'on puisse faire.

4.2.1.2. Coordinates

La classe « Coordinates » contient simplement la latitude, la longitude et la précision. Ce dernier attribut permet de savoir à quelle distance en mètres autour du point spécifié par les coordonnées GPS la position est considérée comme valide.

4.2.1.3. Description

La classe « Description » permet de stocker en mémoire une description. Celle-ci peut être de type texte ou être un média.

Ainsi, la classe contient un booléen spécifiant si l'instance contient un media ou non, le contenu du media qui sera le texte dans le cas d'une description textuelle, et l'adresse html dans le cas d'une description de type média. Enfin le type du média est aussi stocké afin de prévoir l'évolutivité de l'application.

Dans cette classe se trouve une classe asynchrone privée appelée « GrabMedia », elle permet de télécharger le média dans un dossier privé du téléphone créé par l'application en rendant son accès impossible pour l'utilisateur. Cela permet l'affichage du média sans passer par la connexion internet.

4.2.1.4. Clue

La classe « Clue » permet de stocker toutes les informations d'un indice : son id, son libellé, les points à enlever à l'étape, sa date de consultation et un booléen pour savoir s'il a été déverrouillé.

La description est stockée dans un tableau de Description afin de stocker dans l'ordre toutes les descriptions chargées lors du parsing de l'XML.

4.2.1.5. Information

La classe « Information » permet de représenter une information complémentaire en mémoire (pour les circuits touristiques).

Pour cela, la classe contient l'intitulé de l'information, sa date de consultation, ainsi qu'un tableau de Description servant de contenu.

4.2.1.6. Place

La classe « Place » permet de stocker toutes les informations relatives à un lieu : son id, son titre, sa date de validation, sa date d'envoi au serveur, le nombre de tentatives (questions de validation) et un booléen pour savoir s'il est en cours ou non.

La description de l'énigme est stockée dans un tableau de Description similaire à celui du circuit.

La méthode de validation est décrite grâce à une instance de la classe Validation.

Les questions bonus sont décrites dans un tableau de Question.

Les indices présent ou non selon le bon vouloir des rédacteurs sont stockés dans un tableau d'Indices.

Et enfin dans le cas d'un circuit touristique, les informations complémentaires sont stockées dans un tableau d'Information.

Pour ce qui est des méthodes :

1. « codeDeverrouillage(int idLieu, String titreLieu) » retourne une chaîne de caractères. Cette méthode permet grâce à l'id et titre du lieu de générer un code de déverrouillage pour le lieu si jamais l'équipe s'est trompée deux fois aux questions de validation.
2. « toLog » retourne une chaîne de caractères. Cette méthode retourne la date de validation d'un lieu et la date d'envoi au serveur des données concernant ce même lieu.
3. « getNbStagePoints » retourne un entier. Cette méthode retourne le nombre de points restants de l'étape, c'est-à-dire qu'une étape peut contenir des indices qui ont été dévoilés. Ainsi, les points accordés pour la validation de l'étape sont réduits.
4. « getNbBonusPoints » retourne un entier. Cette méthode retourne le nombre de points gagnés grâce aux questions bonus répondues correctement sur ce lieu.
5. « getNbBonusPoints(int q) » retourne un entier. Cette méthode retourne le nombre de points gagnés si la réponse à la question passée en paramètre est correcte.
6. « nbClues » retourne un entier. Cette méthode retourne le nombre d'indices que contient ce lieu.
7. « getTotalNbBonusPoints() » retourne un entier. Cette méthode retourne le nombre de points total des toutes les questions bonus associées à ce lieu.

4.2.1.7. Question

La classe « Question » permet de stocker une question en mémoire. Pour cela la classe contient l'id de la question, son intitulé, sa valeur en points (uniquement si c'est une question bonus), et enfin un tableau de Reponse contenant si nécessaire les choix de réponses possible (vide dans le cas d'une question libre).

Par la suite, d'autres attributs ont été rajoutés tels qu'un tableau de booléen pour savoir quelles réponses à cocher l'utilisateur, une date pour savoir quand cela a été répondu et une chaîne de caractères pour stocker la réponse libre de l'équipe.

Pour ce qui est des méthodes :

1. « isMultipleChoiceQuestion » retourne un booléen. Cette méthode retourne vraie si une question a deux réponses minimums donc elle est à choix multiples.

4.2.1.8. Response

La classe « Response » permet de stocker les choix de réponse en mémoire. Pour cela, la classe contient l'intitulé d'une réponse, d'un booléen pour savoir ce que l'utilisateur a répondu, ainsi qu'un booléen permettant de spécifier si la réponse est juste ou fausse.

4.2.1.9. Validation

La classe « Validation » permet de décrire complètement la méthode de validation d'un lieu :

1. Un booléen « position » permet de spécifier l'activation ou non de la localisation GPS;
2. Un booléen « question » permet de spécifier la présence ou non de questions de validation;
3. un entier « point » permet de stocker la valeur en points de la résolution de l'énigme ;
4. un entier « remainingPoints » permet de stocker les points restants à l'étape lorsqu'un indice est dévoilé.
5. Un booléen « validationByQuestions » et « validationByGPS » qui permet de savoir comment l'étape a été validée.

Si nécessaire, les coordonnées GPS sont stockées dans une instance de la classe Coordinates.

Enfin si nécessaire, les questions de validations sont stockées dans un tableau de Question.

Pour ce qui est des méthodes :

1. « allJust » retourne un booléen. Cette méthode retourne vraie si toutes les réponses de l'utilisateur pour cette validation sont justes.

4.2.2. Package « services »

4.2.2.1. `sendingGPSData`

La classe « `sendingGPSData` » est un service. Un service est comme une activité, il possède un cycle de vie ainsi qu'un contexte qui contient des informations spécifiques sur l'application et qui constitue une interface de communication avec le restant du système. En revanche, à l'opposé des activités, les services ne possèdent pas d'interface graphique : c'est pourquoi on les utilise pour effectuer des travaux d'arrière-plan.

Elle a en son sein plusieurs constantes de temps et de distances qui permettent de définir des zones. Ces zones sont de plus en plus petites lorsque nous nous rapprochons de l'endroit à trouver, permettant d'envoyer des données plus ou moins fréquemment économisant ainsi la batterie du téléphone.

Le service est lancé depuis l'activité « `DescriptionPlace` » en lui transmettant le circuit et le numéro de lieu associé. Ainsi, un « listener » est créé grâce à la position du lieu. Ce « listener » est appelé tous les X temps définit par la zone dans laquelle nous nous trouvons. Si jamais nous nous trouvons dans une zone qui est plus proche de la validation que la précédente, nous mettons notre « listener » à jour grâce à une méthode appelée « `newUpdateGPS` ».

Les principales méthodes employées dans cette classe sont :

1. « `requestLocationUpdates` », méthode d'android. Nous avons en premier paramètre le « provider » c'est-à-dire le service que le téléphone utilise pour se géolocaliser. En second paramètre, le temps d'intervalle d'appel du « listener ». En troisième paramètre, la distance minimal que l'utilisateur doit parcourir en X temps (définit selon le deuxième paramètre). Et enfin, l'appel au « listener ».

Nous allons expliquer certaines méthodes du « listener » utilisées :

- 1.1. « `onProviderDisabled` » qui lance une activité appelée « `MyDialog` » (voir partie ???)
- 1.2. « `onLocationChanged` » appelée suivant certaines périodes de temps définit par les paramètres du « listener ». On test si la validation se fait par géolocalisation. Si c'est le cas, on fait le test pour savoir dans quelle zone se trouve l'équipe. Plus cette dernière est proche du lieu à trouver, plus nous réduisons l'intervalle du temps d'appel du « listener ». Un autre test est lancé, si la distance du GPS par rapport au lieu est inférieure à la précision du lieu indiqué dans le fichier XML, alors l'étape est validée par géolocalisation. Ce qui nous amène à (peu importe où nous nous trouvons dans l'application) être

automatiquement redirigé vers l'activité « DescriptionPlace » avec notre vue qui est mise-à-jour.

2. « newUpdatesGPS(long time, float distance) », elle prend en paramètre un temps et une distance. Si le temps ou la distance a été modifié par rapport au précédent appel du « listener », nous l'arrêtons et le mettons à jour avec les nouvelles valeurs.
3. « stopGPS », arrête l'exécution du « listener » mais pas le service.
4. « uploadGPS(Location location) », exécute une tâche asynchrone qui s'occupe d'envoyer les données de la position (longitude et latitude) de l'équipe à l'adresse indiquée.

4.2.3. Package « utilities »

4.2.3.1. Files

Cette classe permet de sauvegarder et de charger le circuit depuis un fichier nommé « CIRCUIT_circuitId_userId ». Ce fichier est ce qui sert d'avancement dans le rallye, permettant de savoir où en est l'équipe.

Pour ce qui est des méthodes utilisées :

1. « load(Context, String, String) » retourne un objet de type « Circuit ». Les paramètres sont nombres de trois. Le premier étant le contexte de la vue depuis laquelle cette méthode est appelée. Le second paramètre est l'identifiant du circuit et le troisième paramètre est l'identifiant de l'équipe.
2. « save(Context, Circuit), permet de sauvegarder un circuit dans le fichier précédemment cité grâce au contexte de la vue et à l'objet « Circuit » modifié.
3. « getValuesToSendFromCircuit(Circuit, int) » retourne une ArrayList de NameValuePair. Elle prend en paramètre un circuit qui contient les différentes informations à envoyer au serveur et le numéro du lieu pour spécifier de quel lieu sont les informations à envoyer.
4. « sendToUrl(ArrayList<NameValuePair>) », envoie les données au serveur retourné par la méthode décrite dans le 3. à l'adresse spécifiée.

4.2.3.2. SendCircuitToUrl

Est une classe réalisant une tâche asynchrone qui consiste à envoyer tous les lieux du circuit au serveur. Elle appelle en son sein la précédente méthode que nous avons décrite « `getValuesToSendFromCircuit` ». Dans cette classe nous avons une « `progressDialog` » qui est en fait utilisée lorsque nous cliquons sur le bouton « Terminer le rallye ». Elle nous montre la progression d'envoi des données au serveur par le biais d'une barre de chargement. Si tout fonctionne parfaitement nous sommes redirigés vers la vue « `DescriptionRally` » avec le bouton « Liste des étapes » qui est affiché.

4.2.3.3. Strings

Est une classe qui permet de découper une chaîne de caractères qui rencontre des « / » ou « . » par exemple, notamment les urls comme ceux utilisés pour les images.

La méthode « `getLastNameOfPath(String)` » permet de récupérer le dernier élément de la chaîne de caractères passé en paramètre.

La méthode « `getParentPath(String)` » permet de récupérer la chaîne de caractères passé en paramètre en enlevant le dernier élément de cette dernière désigné par le « `Pattern` ».

4.2.4. Les activités

4.2.4.1. AddRally

La classe « `AddRally` » permet de récupérer un rallye depuis une URL.

Une tâche asynchrone télécharge le contenu XML. Des méthodes ont été prévues pour vérifier l'activation de la connexion Internet du Smartphone et l'existence de la page web.

Une fois que le contenu XML est récupéré, la fonction « `parseXML` » se charge de traduire les balise XML en une classe `Circuit`.

Par ce qui est des méthodes utilisées :

1. « `onClickSubmitButton` » est appelée lorsque l'on clique le bouton « Ajouter ». Elle affiche la liste des rallyes que l'utilisateur a précédemment entrée. Si nous entrons un rallye deux fois (c'est-à-dire le même code), une fenêtre de dialogue s'affiche

nous demandant si nous voulons l'écraser. L'écrasement provoque la suppression de l'avancement du rallye.

2. « `downloadXML(String)` » est appelée pour vérifier qu'un code est bien entré (ce qui est passé en paramètre) et qu'internet est disponible. Si ces deux conditions sont remplies, alors nous exécutons la tâche asynchrone pour télécharger le contenu XML.
3. « `checkInternetConnection` » retourne un booléen. Cette méthode retourne « `true` » si une connexion internet est disponible sinon « `false` ».
4. « `parseXML(String, boolean)` », retourne un booléen. Elle a en premier paramètre la chaîne de caractères qui représente le code de l'équipe. Elle a en second paramètre un booléen qui est « `true` » si le code vient du dossier « `Download` » du téléphone ou « `false` » s'il vient de l'url spécifié dans le code. Cette méthode permet donc de parser un fichier XML. Elle renvoie « `true` » si tout s'est bien passé sinon « `false` » avec des messages d'erreurs explicites qui permet de savoir ce qu'il s'est passé.
5. `getDescription(Element, ArrayList)` est utilisé pour créer un objet description suivant si l'élément passé en paramètre est de type « `texte` » ou de type « `media` ».

Enfin, on se retrouve encore avec une classe privée nommée « `GrabUrl` », qui représente une tâche asynchrone permettant d'obtenir le fichier XML grâce à une url.

4.2.4.2. **CheckableRelativeLayout**

Est une classe qui permet de rendre un « `RelativeLayout` », « `checkable` ». Dans l'application « `Rallye` » vous verrez que par moment la liste d'items n'est pas seulement composée d'intitulés et « `checkbox` » mais d'« `ImageView` » en plus. De base, avec les « `layouts` » d'origines d'Android, il n'est possible que de mettre un « `TextView` » et une « `Checkbox` » d'où l'élément « `CheckedTextView` ». Cette classe a pour but d'offrir aux développeurs la possibilité de pouvoir personnaliser ces « `RelativeLayout` ».

4.2.4.3. **ClueAdapter**

Est une classe qui est en fait un « `adapter` ». Cette classe permet la personnalisation de la liste des indices. Comme pour les étapes, seul le premier indice est disponible. Les indices disponibles sont indiqués grâce à la présence d'une flèche bleue et leur intitulé est de couleur noire. Les indices qui sont révélés sont indiqués par la présence d'une coche verte. Enfin, pour les indices bloqués, c'est-à-dire ceux dont le précédent indice n'a pas encore été dévoilé, sont indiqués par une image représentant un cadenas. Seule la méthode « `updateClueList(ArrayList<Clue>)` » est présente. Elle permet de mettre à jour la liste des indices qui sont fournis dans l'« `adapter` ».

4.2.4.4. DisplayClues

Est une classe qui permet d'afficher les indices (textes/images) et de retirer un certain pourcentage des points lorsque s'ils sont dévoilés. L'appel de cette vue se fait lorsque l'utilisateur clique sur le bouton « Indices » et ensuite clique sur un indice disponible dans la liste. Lorsqu'un indice est dévoilé, ce dernier est accompagné d'une coche verte dans la liste des indices pour prévenir qu'il a été consulté.

Pour ce qui est des méthodes utilisées :

1. « `displayImg(String, LinearLayout)` », permet d'afficher dans le layout passé en second paramètre, l'image qui se trouve dans le dossier privé du téléphone grâce au lien passé en premier paramètre.
2. « `displayTxt(String, LinearLayout)` », permet d'afficher dans le layout passé en second paramètre le texte qui est passé en premier paramètre.
3. « `round(double)` » retourne un entier. Cette méthode retourne un entier de manière intelligente. C'est-à-dire, par exemple, si 25.4 est passé en paramètre on effectue une troncature. On renvoie 25. Si 25.5 est passé en paramètre on effectue un arrondi. On renvoie alors 26.
4. « `restartService()` », permet de redémarrer le service car le circuit a été mis à jour puisque des points ont été retirés. On a besoin de renvoyer le circuit.
5. « `isMyServiceRunning(Class<?>)` », retourne un booléen. Cette méthode retourne vraie si le service passé en paramètre (Exemple : `MonService.class`) est en activé.

4.2.4.5. DisplayInformations

Cette classe a pour but d'afficher à l'écran des informations. Cela étant l'équivalent des questions bonus mais pour les circuits touristiques.

Le principe est d'afficher à l'écran une succession de textes de description, et d'images ; tous deux provenant de la classe `Description`.

Les images sont affichées dans des « `ImageView` » afin de profiter du chargement dynamique du contenu, et les textes dans des « `TextView` ».

Pour ce qui est des méthodes utilisées :

1. « `displayImg(String, LinearLayout)` », permet d'afficher dans le « layout » passé en second paramètre, l'image qui se trouve dans le dossier privé du téléphone grâce au lien passé en premier paramètre.

2. « `displayTxt(String, LinearLayout)` », permet d'afficher dans le « layout » passé en second paramètre le texte qui est passé en premier paramètre.

4.2.4.6. `DisplayQuestionsResponses`

Cette classe a pour but d'afficher à l'écran l'intitulé d'une question ainsi que les possibilités de réponse.

Il peut y avoir des questions libres, des questions à choix unique et des questions à choix multiple.

Si la question est à choix unique, la `ListView` des réponses est donc vide, et l'utilisateur peut saisir sa réponse dans une zone de texte.

Sinon la `ListView` contient tous les choix de réponses. L'utilisateur peut y répondre en cochant les cases voulues.

Lors du clic sur le bouton « Valider » ou du bouton « retour en arrière », la ou les réponses sont transmises à la vue précédente (`DisplayQuestionList`) et la vue courante se termine.

Pour ce qui est des méthodes utilisées :

1. « `exit()` » permet d'enregistrer les réponses de l'utilisateur que ce soit pour les questions de validation ou les questions bonus. Cette méthode est appelée lorsque le bouton « Valider » ou le bouton « retour en arrière » du Smartphone est pressé.
2. « `getQuestion` » retourne une question. Cette méthode permet de récupérer la question associée à ce lieu et à l'index de la liste où l'utilisateur a cliqué.

4.2.4.7. `DisplayQuestionsList`

Cette classe a pour but d'afficher à l'écran les intitulés des questions dans une « `ListView` ». Lorsque l'utilisateur clique sur une question, une nouvelle instance de la classe « `DisplayQuestionReponses` » est créée et s'affiche alors à l'écran.

Cette classe sert notamment à afficher la liste des informations et des indices aussi. Respectivement lorsque nous cliquons dessus, nous arrivons sur la classe « `DisplayInformations` » et « `DisplayClues` ».

Il est à noter que chaque type de liste (questions, information et indices) possède son propre « adapter » qui permettent la personnalisation de ces listes. Ces derniers seront vus plus tard dans ce manuel.

Pour ce qui est des méthodes utilisées :

1. « cancelAnswers » est appelé lorsque le bouton « Annuler » est appuyé. Cette méthode permet d'annuler les réponses que l'utilisateur a répondues précédemment.

4.2.4.8. DisplayQuestionsUnlocked

Cette classe a pour but d'afficher à l'écran la liste des lieux possédant des questions bonus, et pour lesquels l'énigme a été validée par l'utilisateur.

Lors d'un clic sur un lieu de la ListView, la liste des questions bonus de ce lieu s'affiche alors à l'écran via une instance de la classe DisplayQuestionList.

4.2.4.9. InformationAdapter

Est une classe qui est en fait un « adapter ». Cette classe permet la personnalisation de la liste des informations. Les informations qui n'ont pas encore été consultées sont accompagnées d'une flèche bleue dans la liste des informations. Dans le cas contraire, elles sont accompagnées d'une coche verte. Seule la méthode « updateInformationList(ArrayList<Information>) » est présente. Elle permet de mettre à jour la liste des informations qui sont fournis dans l' « adapter ».

4.2.4.10. ListOfStages

La classe « ListOfStages » affiche à l'écran la liste des étapes d'un circuit. Elle permet d'accéder à un lieu donné via le chargement d'une instance de la classe « PlaceDescription ».

Dans le cas d'un rallye, la liste des étapes accessibles est établie en fonction du dernier lieu déverrouillée par l'utilisateur.

Cette classe permet aussi d'accéder aux questions déverrouillées via le chargement d'une instance de la classe « DisplayQuestionUnlocked ». Lorsque l'utilisateur décide de terminer

un rallye, l'ensemble des réponses aux questions bonus de chaque étape sont envoyées au serveur via une méthode spécifique de cette classe.

Pour ce qui est des méthodes utilisées :

1. « `updateButtonsDisplayed` » permet d'afficher les boutons correspondant à l'état dans lequel se trouve le rallye. Des boutons tels que « Terminer le rallye » ou « Questions bonus déverrouillées »
2. « `displayPlace` » permet d'afficher le lieu choisi par l'utilisateur
3. « `onClickListStagesBonusQuestions` » permet d'afficher la liste des étapes ayant des questions bonus.
4. « `onClickListeEtapesTerminerRallye` » permet de terminer le rallye. Le bouton est activé seulement si l'équipe a réussi à aller au bout du rallye, c'est-à-dire à avoir validé toutes les étapes. Une fenêtre de dialogue est affichée pour demander à l'utilisateur s'il est sûr de son choix. Une fois le bouton « oui » cliqué, toutes les étapes et leurs questions bonus sont envoyées au serveur. Nous pouvons le progrès de l'envoi par le biais d'une barre de progression.
5. « `EtapeListAdapter` » n'est pas une méthode mais une classe interne. Elle étend d'un `adapter` et comme tous les autres « `adapters` », elle permet de rendre la liste des étapes personnalisable. Si un lieu est en cours, une flèche bleue est affichée. Si un lieu est validé, une coche verte est affichée. Si un lieu est bloqué, un cadenas est affiché.

4.2.4.11. `MainActivity`

La classe `MainActivity` est la classe de base de l'application. Celle-ci est chargée en premier lorsque l'application démarre.

Cette classe permet d'afficher dans une `ListView` tous les rallyes déjà ajoutés à l'application, de lancer un rallye en le sélectionnant et en cliquant sur le bouton « Ouvrir », d'ajouter de nouveaux rallye en cliquant sur le bouton « Ajouter Rallye » et enfin de supprimer des rallyes grâce au bouton « Supprimer ».

Il est à noter que cette classe utilise la sérialisation pour sauvegarder dans un fichier tous les rallyes qui ont été ajoutés à l'application.

Pour ce qui est des méthodes utilisées :

1. « `save` » permet d'enregistrer la liste des circuits dans un dossier caché du téléphone.
2. « `load` » permet de charger la liste des circuits depuis un dossier caché du téléphone.
3. « `initButtons` » permet d'initialiser la vue. Les boutons sont grisés ou non suivant s'il y a des circuits présents dans la liste des circuits.

4. « onClickAddButton » permet de rediriger l'utilisateur vers la vue « addRally » lorsque l'icône en forme de croix verte est pressé.
5. « onClickDeleteButton » permet de supprimer un rallye de la liste des circuits avec le fichier qui lui est associé grâce à la méthode « deleteProgress » qui est appelée.
6. « deleteProgress » permet de supprimer le fichier « CIRCUIT_IDCIRCUIT_IDEQUIPE » qui est associé au circuit de la liste pressé.
7. « onClickOpenButton » permet de lancer le rallye. Si le circuit sélectionné n'a pas de fichier associé, on commence un nouveau rallye. Dans le cas contraire, on lance le rallye avec le fichier associé.

4.2.4.12. MyDialogGPS

Est une classe qui permet d'afficher une fenêtre de dialogue pour prévenir que l'utilisateur n'a pas son GPS d'activé. Cette classe est appelée depuis le service « SendingGPSData » dans la méthode « onProviderDisabled ». Une classe a été créée ainsi car on ne peut pas afficher une fenêtre de ce type dans un service.

4.2.4.13. PlaceDescription

La classe « PlaceDescription » affiche à l'écran l'énigme correspondant à un lieu. Elle permet de gérer toutes les possibilités de validation que ce soit par positionnement GPS ou par questions de validation.

De plus cette classe permet d'atteindre la liste des questions bonus grâce au bouton «Questions bonus ».

Il est à noter que, dans cette vue, la réussite ou non de la validation est directement affichée à l'écran pour que l'utilisateur soit mis au courant. Suite à une validation effective de l'étape en cours, une méthode de cette classe est appelée afin de transmettre en arrière-plan les données appropriées de validation au serveur.

Pour ce qui est des méthodes utilisées :

1. « testValidationIsFinished » permet de valider une étape que ce soit par questions de validation, par géolocalisation ou par code de déverrouillage du côté application Smartphone. De plus, si nous nous trompons lors des questions de validation, le nombre de tentatives restantes est indiqué. Si toutes les tentatives ont été épuisées, on bloque l'étape en remplaçant le bouton des questions, obligeant ainsi l'équipe a appelée l'organisateur.

2. « actions_validationStageEstablishedBy_GPSorQuestionV » permet de mettre à jour le statut de l'étape en indiquant de comment cette dernière a été validée et combien de points elle a rapporté. Elle permet aussi lorsqu'une étape vient d'être validée d'afficher une fenêtre de dialogue pour le prévenir de sa réussite (également lorsque le circuit est terminé).
3. « actions_validationStageEstablishedBy_Code » permet de mettre à jour le statut de l'étape en indiquant qu'elle a été débloquée par code de déverrouillage donc elle ne rapporte aucun points.
4. « actions_validationStageEstablished_noValidationNeeded » permet de valider une étape qui ne nécessite aucune validation.
5. « displayImg(String, LinearLayout) », permet d'afficher dans le « layout » passé en second paramètre, l'image qui se trouve dans le dossier privé du téléphone grâce au lien passé en premier paramètre.
6. « displayTxt(String, LinearLayout) », permet d'afficher dans le « layout » passé en second paramètre le texte qui est passé en premier paramètre.
7. « onClickListOfStages » appelle la méthode « displayListOfStages »
8. « displayListOfStages » permet d'afficher la liste des étapes.
9. « onClickButtonPreviousPlace » permet d'afficher le lieu précédent par la pression du bouton à gauche de celui de la liste des étapes.
10. « onClickButtonNextPlace » permet d'afficher le lieu suivant par la pression du bouton à droite de celui de la liste des étapes.
11. « onClickButtonPlaceQuestions » appelée lorsque nous cliquons sur le bouton « Questions de validation », une fenêtre de dialogue est affichée pour prévenir l'utilisateur qu'il est préférable qu'il essaye de valider ce lieu par géolocalisation. Si jamais l'étape a été verrouillée suite à des tentatives ratées par l'utilisateur, le bouton « Questions de validation » est remplacé. Enfin, il y a aussi un test pour savoir si le code de déverrouillage entré est le bon ou non. L'utilisateur est prévenu grâce à une fenêtre de dialogue.
12. « onClickButtonBonusPlace » permet d'afficher la liste des questions bonus que propose ce lieu.
13. « onClickButtonCluePlace » permet d'afficher la liste des indices que propose ce lieu.
14. « validationStageSmartphone » appelée lorsqu'une étape vient d'être validée par questions de validation, géolocalisation ou code de déverrouillage.
15. « isMyServiceRunning(Class<?>) », retourne un booléen. Cette méthode retourne vraie si le service passé en paramètre (Exemple : MonService.class) est en activé.

4.2.4.14. QuestionAdapter

Est une classe qui est en fait un « adapter ». Cette classe permet la personnalisation de la liste des questions. En effet, lorsqu'une question n'a pas encore été répondue, on affiche une flèche bleue. Dans le cas contraire, une coche verte est affichée.

Mais ce n'est pas tout cela fonctionne aussi pour les questions bonus. De plus lorsque le rallye est terminé, nous affichons dans la liste des questions bonus lesquelles sont justes (coche verte) et lesquelles sont fausses (croix rouge).

4.2.4.15. RallyDescription

La classe « RallyDescription » affiche à l'écran la description du rallye, et permet de le commencer, le reprendre, ou simplement revoir les différentes étapes d'un rallye terminé (sans possibilité de modification dans ce cas).

Même si le bouton dédié est caché, il est à noter que cette classe peut réinitialiser l'avancement d'un rallye. Pour cela les valeurs de la classe Avancement correspondante sont remises à leurs valeurs initiales, puis l'avancement est sérialisé dans le fichier habituel pour écraser les données existantes. En outre, le serveur est prévenu de la nécessité de réinitialiser les données concernant cette équipe, via une tâche asynchrone.

En temps de développement, réactivez le bouton « Réinitialiser » pour aisément réinitialiser un rallye si l'on rencontre un problème lors de tests.

Remarque : la dernière partie en italique et surlignée n'a pas été testée lors du stage. Mais vu que la classe Avancement a été supprimée, cette partie ne doit pas fonctionner mais ce n'est pas grave. En effet, le bouton et les méthodes qui y sont associées peuvent être supprimées ou mis en commentaires.

Pour ce qui est des méthodes utilisées :

1. « onClickDisplayStages » appelle la méthode « displayListOfStages »
2. « displayListOfStages » permet d'afficher la liste des étapes lorsque le bouton est appuyé (« Commencer », « Reprendre » et « Liste des étapes » selon l'avancée du rallye).
3. « displayImg(String, LinearLayout) », permet d'afficher dans le layout passé en second paramètre, l'image qui se trouve dans le dossier privé du téléphone grâce au lien passé en premier paramètre.
4. « displayTxt(String, LinearLayout) », permet d'afficher dans le layout passé en second paramètre le texte qui est passé en premier paramètre.

5. « `onClickButtonReinitializeProgress` » permet de réinitialiser un rallye lorsque le bouton est appuyé. Cette méthode ainsi que le bouton qui y est associé sont à utiliser dans le cadre du développement de l'application pour les tests. Attention, ceci n'a pas été testé lors du stage.
6. « `checkInternetConnection` » retourne un booléen. Cette méthode retourne « `true` » si une connexion internet est disponible sinon « `false` ».
7. « `isMyServiceRunning(Class<?>)` », retourne un booléen. Cette méthode retourne vraie si le service passé en paramètre (Exemple : `MonService.class`) est en activé.

4.2.4.16. **ResponseAdapter**

Est une classe qui est en fait un « adapter ». Cette classe permet la personnalisation de la liste des questions. En effet, elle permet lorsque le rallye est terminé et que nous voulons voir les réponses pour les questions bonus, la possibilité de voir les erreurs commises. A côté des cases à cocher où l'utilisateur a appuyé, une image représentant une coche verte peut être visible signifiant que cette réponse est bonne.

Enfin, l'adapter nous permet surtout de rendre les items de la liste désactivés et non la liste elle-même grâce aux méthodes « `areAllItemsEnabled` » et « `isEnabled` ». Comme cela l'utilisateur peut consulter toutes les réponses sans les modifier.

4.2.4.17. **SplashScreen**

Est une classe qui affiche le logo de l'application pendant trois secondes.

4.3. Fichiers de sauvegarde

Comme vous avez pu le lire jusque-là, les fichiers de sauvegarde sont beaucoup utilisés dans notre application.

Afin d'éviter qu'ils puissent être modifiés par l'utilisateur, ils se trouvent dans le dossier de l'application (difficilement accessible) et ils sont automatiquement verrouillés en écriture pour toute application externe.

Le fichier servant à stocker toutes les descriptions de rallyes se nomme « `saveCircuits` » et est unique pour l'application.

Les fichiers servant à stocker les avancements pour chaque rallye se nomment de la façon suivante :

CIRCUIT<id du circuit><id de l'équipe>

Ainsi on peut avoir uniquement un seul avancement par circuit et par code d'équipe. De ce fait on ne peut pas maintenir deux avancements différents sur un même circuit et sur un même Smartphone.

4.4. Communication avec le serveur

La communication avec le serveur doit obligatoirement se faire dans des tâches asynchrones afin d'éviter que l'application ne « freeze » pendant un téléchargement ou un envoi de données.

La récupération de données utilise simplement la récupération du code source d'une page qui est présente dans le protocole http.

L'envoi de données utilise le principe de POST du protocole http, ainsi même en « écoutant » les échanges de données il n'est pas directement possible de savoir quelles données ont été transmises au serveur (comme avec l'utilisation de GET).

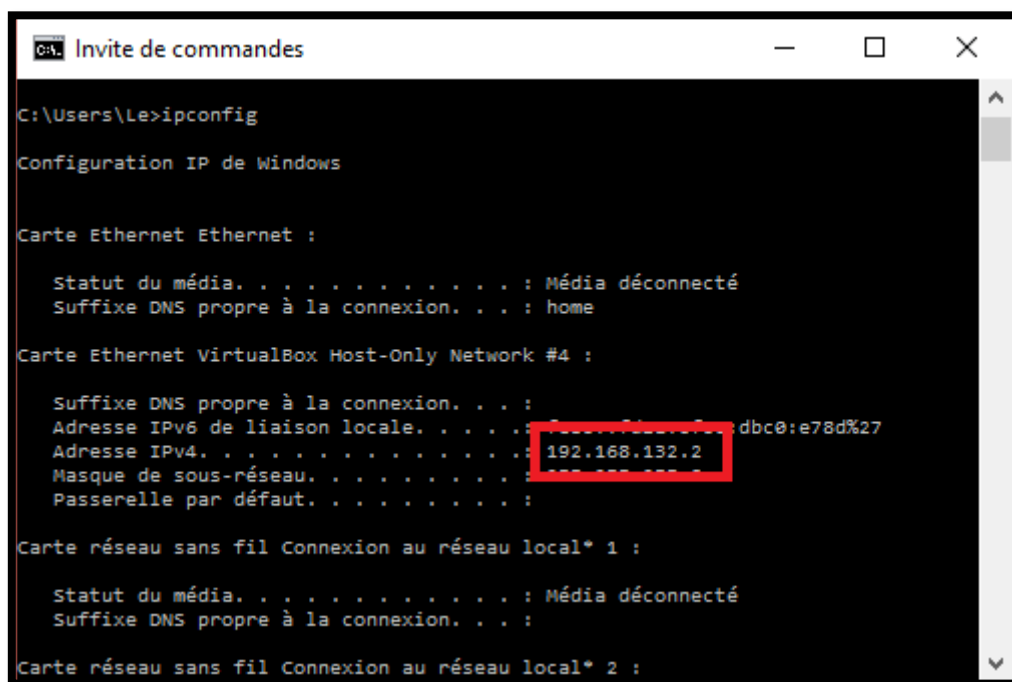
5. Jouer à un rallye en local

Dans un premier temps du côté de l'application android, il faut modifier l'url dans tous les fichiers qui servent à envoyer des données au serveur. Vous trouverez un lien à modifier dans les fichiers suivants :

- Files.java
- SendCircuitToUrl.java
- sendingGPSTData.java (méthode uploadGPS)
- AddRallye.java

Dans ces quatre fichiers, changer le lien <http://www.rallye.miage.uha.fr/index.php/> en http://192.168.132.2/rallye_web/SITERALLYE/index.php/.

Pourquoi **192.168.132.2** ? Car cela est l'adresse de ma machine virtuelle GENYMOTION. Il suffit de taper la commande « ipconfig » dans l'invite de commandes pour l'obtenir.



```

C:\Users\Le>ipconfig

Configuration IP de Windows

Carte Ethernet Ethernet :

    Statut du média. . . . . : Média déconnecté
    Suffixe DNS propre à la connexion. . . : home

Carte Ethernet VirtualBox Host-Only Network #4 :

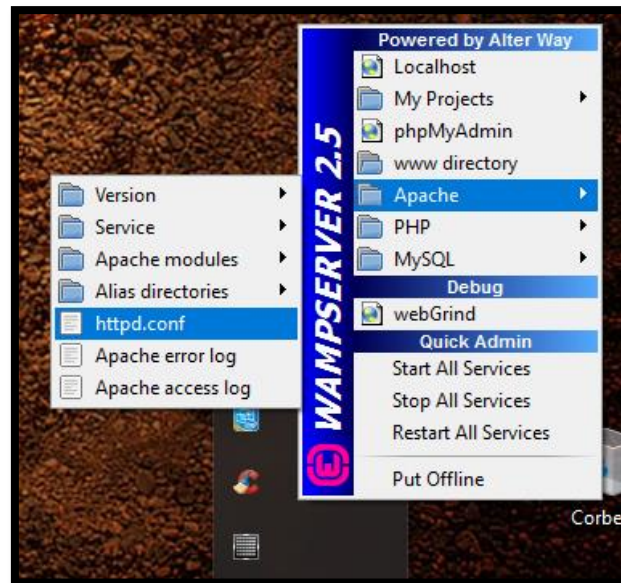
    Suffixe DNS propre à la connexion. . . :
    Adresse IPv6 de liaison locale. . . . : fe80::57d8:1101:1d9c:ebc0:ef78d%27
    Adresse IPv4. . . . . : 192.168.132.2
    Masque de sous-réseau. . . . . : 255.255.255.0
    Passerelle par défaut. . . . . :

Carte réseau sans fil Connexion au réseau local* 1 :

    Statut du média. . . . . : Média déconnecté
    Suffixe DNS propre à la connexion. . . :

Carte réseau sans fil Connexion au réseau local* 2 :
  
```

Pour finir il faut aller dans le fichier de configuration « httpd.conf » d'Apache et modifier certaines lignes pour autoriser le serveur à recevoir des données d'une source inconnue.



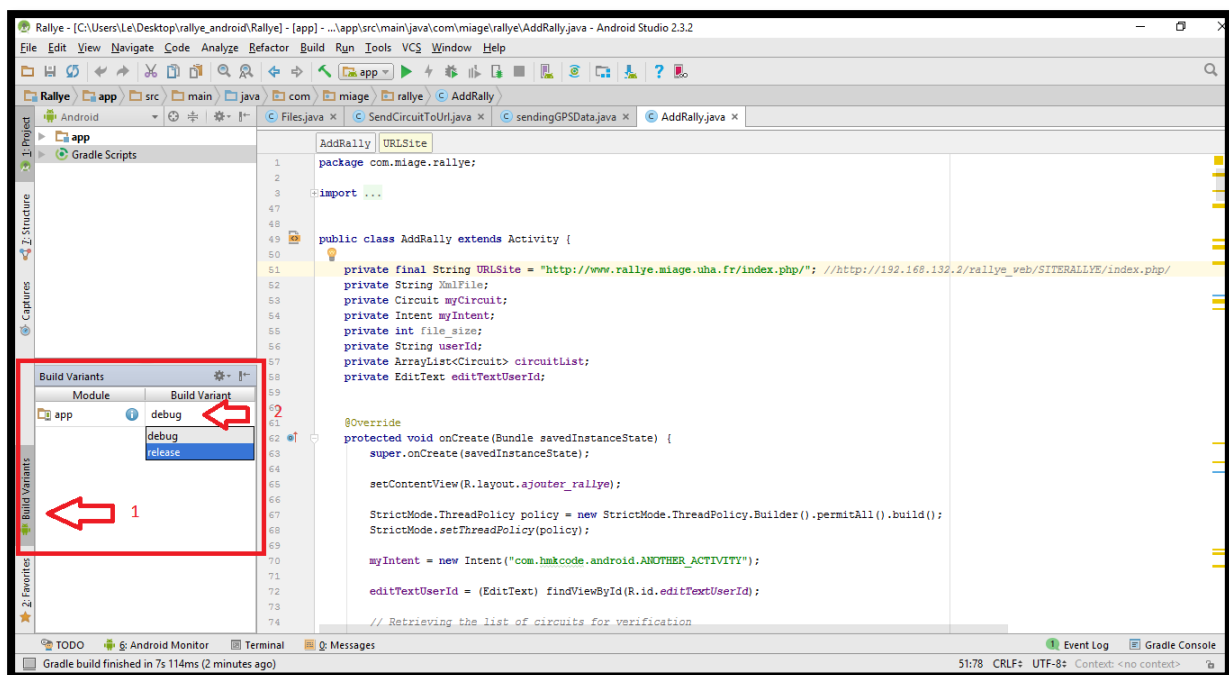
Une fois le fichier trouvé, il faut modifier les lignes « AllowOverride none » et « Require all denied » en « AllowOverride all » et « Require all granted » qui sont présentes entre les balises <Directory></Directory> et <Directory "c:/wamp/www/"></Directory>

6. Publier son application sur Google Play Store

Dans un premier temps, il faut avoir un compte Google. Si jamais ce n'est pas le cas, vous pouvez en créer sur l'adresse suivante : <https://accounts.google.com>.

Une fois votre compte Google créé, connectez-vous à ce dernier et allez sur le lien suivant : <https://play.google.com/apps/publish>. Ensuite, s'il vous est demandé d'associer votre compte Google à « Developer Console » faites-le. Enfin, il faut accepter le contrat développeur et payer 25\$ (une seule fois et valable pour toutes les applications futures).

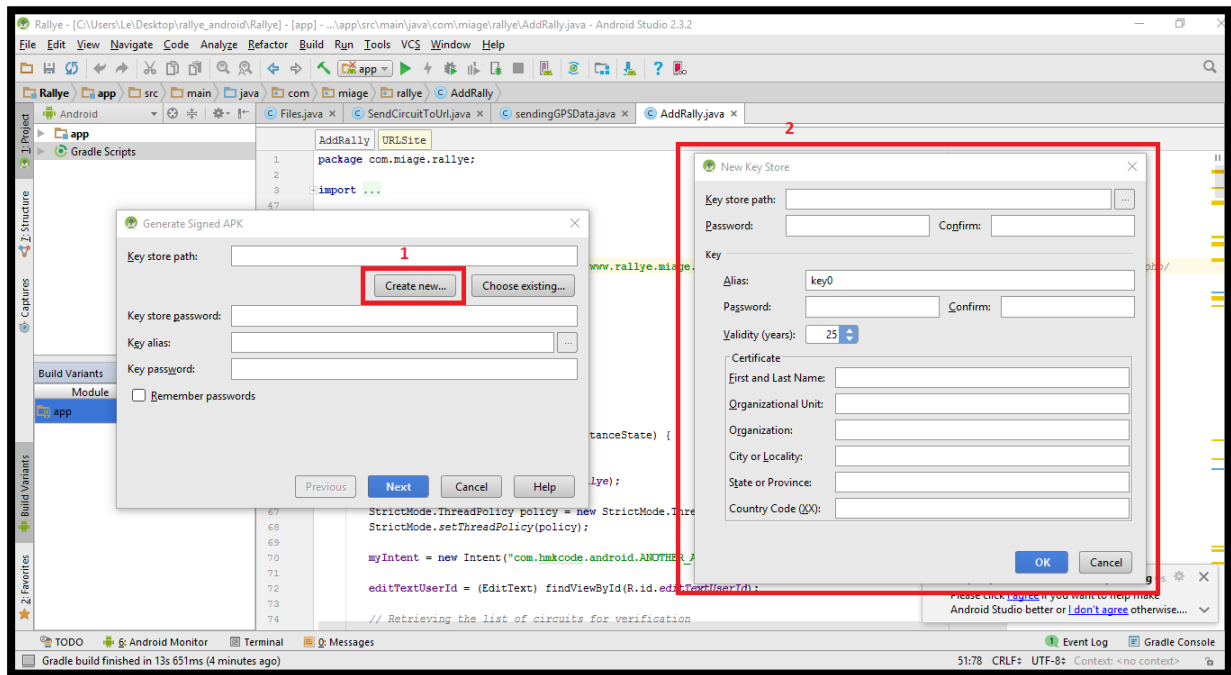
Dans Android Studio, il faut maintenant construire l'APK c'est-à-dire l'exécutable de notre application. Dans un premier temps, allez en bas à gauche de votre logiciel, cliquez sur « Build Variants » et changez « debug » en release « release ».



Ensuite, allez dans l'onglet menu « Build » et cliquez sur « Generate Signed APK ». Une nouvelle fenêtre s'ouvre « Generate Signed APK », si vous avez déjà un « key store path » appuyez sur le bouton « Choose existing... » sinon cliquez sur « Create new ... » et une nouvelle fenêtre apparaîtra « New Key Store ». Il y a plusieurs champs à renseigner tels que :

- Key store path : dossier où sera stockée la clé
- Password – Confirm : un mot de passe
- Alias : nom de la clé
- Password – Confirm : un mot de passe (peut être le même que le précédent)

- Validity (years) : validité de l'application. Un minimum de 25 ans est recommandé par Android.
- First and Last Name : prénom et nom
- Organizational Unit : le service de l'entreprise qui a développé l'application
- Organization : le nom de l'entreprise
- City or Locality : nom de la ville
- State or Province : le nom de l'état, région ou département
- Country Code (XX) : le code du pays (par exemple, FR pour France)



Enfin, il ne reste plus qu'à cliquer sur « Ok », « Next » et « Finish ». Votre « APK » sera générée dans le dossier « app/app-release.apk » de votre application.

Dernière étape, il faut retourner sur <https://play.google.com/apps/publish> et uploadez votre APK. Pour cela cliquez sur « Upload your first APK to Production ». Droppez votre APK dans la zone prévue à cet effet. Allez dans l'onglet « Store Listing » et mettez les informations suivantes : description de votre application, des screenshots de votre application, une icône (512*512), une vidéo promotionnelle (lien youtube) ... Pour « Privacy policy » laissez en blanc et cocher la case qui se trouve à côté si on n'utilise pas de données privées qui concernent les utilisateurs.

Vous pouvez sauvegarder vos modifications à tout moment en appuyant sur « Save Draft » en haut de la page.

Ensuite allez dans l'onglet « Content Rating » : un questionnaire vous sera posé pour savoir quels éléments les utilisateurs peuvent accéder avec votre application.

Ensuite allez dans l'onglet « Pricing and distribution », il y aura alors d'autres informations à compléter. Dans un premier temps, vous devez indiquer si votre application est « free » (gratuite) ou « paid » (payante). Si votre application est payante il faudra créer un « merchant account » (cela permet de déterminer comment Google va vous payer). Vous aurez alors aussi à choisir dans quels pays votre application sera disponible, est-ce que votre application contient des pubs ou non. Enfin, dans « Consent », vous devez cocher les consentements obligatoires pour pouvoir publier votre application.

Dans l'onglet « in-app products » et « Services & APIs », il faut indiquer si vous utilisez des applications, services ou APIs au sein de votre application.

Il ne reste plus qu'à sauvegarder les dernières modifications et publier l'application.

Documentations supplémentaires :

- <https://openclassrooms.com/courses/creez-des-applications-pour-android/publier-et-rentabiliser-une-application>
- <http://mathias-seguy.developpez.com/tutoriels/android/signer-deployer-application-android/>
- <http://doc.pcsoft.fr/fr-FR/?9000118>
- https://www.embarcadero.com/starthere/xe5/mobdevsetup/android/fr/preparing_an_android_application_for_deployment.html

7. Charte graphique

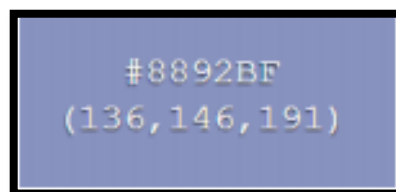
7.1. Le nom

Comme le projet se nomme Rallye, nous avons nommé notre application smartphone « Rallye Miage Mulhouse ».

7.2. Le choix des couleurs

Pour respecter une certaine identité au sein de nos créations, nous avons utilisé une charte graphique commune pour le site web et l'application Smartphone.

Notre charte graphique est relativement simple dans le cadre de l'application : nous avons utilisé la couleur illustrée ci-contre pour les barres de titres et les différents boutons de l'application.



Les textes sont en noir et la couleur d'arrière-plan est le blanc.

7.3. Le logotype



Comme notre application n'est pas destinée à être utilisée directement, nous nous sommes limités à un logo basique, mais clair et évocateur, utilisant uniquement le nom de notre application.

7.4. Images et icônes

Quelques icônes ont été utilisées à de multiples reprises au sein des vues, que ce soit pour signifier le statut d'une étape ou pour moderniser l'interface via des "boutons images". Tout cela permet par ailleurs de renforcer l'identité propre de l'application Rallye.

