Depot :  
-documentation  
-code  
-apk  
-preuves (pdf)

->playstore

-% de travail

Documentation

|  |  |
| --- | --- |
| Je sais décrire le contexte de mon application |  |
| Je sais faire un diagramme de cas d’utilisation |  |
| Je sais concevoir un diagramme UML de qualité |  |
| Je sais décrire un diagramme UML |  |

Code

|  |  |
| --- | --- |
| Je sais utiliser les Intents | Ils sont utilisés dans la BaseActivity->changeActivity, classe mère des activités pour éviter les redondances.  La plupart du temps, ce sont des boutons de navigation qui permettent de passer d’une vue à l’autre. On les configure via configureNavigationButton de BaseActivity qui fait appel à changeActivity.  On sauvegarde dans les Intents une instance de la partie en cours (ou null), voir section **persistance légère**. |
| J’utilise le SDK le plus bas possible | Le **minSdkVersion**  de notre fichier **gradle** est de 15, pour un support de 99% des téléphones android sur le marché. |
| Je sais distinguer mes ressources en utilisant les qualifiers | Toutes nos ressources sont situées dans le bon qualifier : -Nos images et icons dans le **drawable**.  -Nos fichier de vue xml dans **layout**.  -Notre logo dans **mipmap**.  -Nos musiques dans **raw**.  -Les autres fichiers tel que strings.xlm dans **values**. |
| Je sais modifier le manifestde mon application | Le manifest a été changé pour permettre l’utilisation de la caméra :  <uses-permission android:name="android.permission.CAMERA" /> <uses-feature android:name="android.hardware.camera.front" android:required="false"/>  Voir section **Application**. |
| Je sais faire des vues xml en utilisant les composants adéquats | Tous nos choix de layout ont été réfléchis pour être les mieux adaptés au contenant qu’ils affichent.  Le ConstraintLayout est conservé pour **activity\_camera** car pratique pour définir des vues superposés (le contenu de la caméra avec un bouton photo par dessus)    Les **LinearLayout** pour la vue d’acceuil, de pause et de score.    La **ListView** pour l’affichage des scores.    Le **RelativeLayout** pour certains éléments.    Le **GridLayout** pour l’affichage des informations de la partie. |
| Je sais coder proprement mes activités, pour qu’elle ne relaye que les évènements | Nos activités releyent bien les évènements au modèle. Le meilleur exemple est la relation entre **GameActivity et Partie**. Par exemple ligne 280 de GameActivity**.**  public void onTextChanged(CharSequence s, int start, int before, int count) {  // Informer la partie qu'un caractère vient d'être écrit  partie.caractereWritten();  Dès l’écriture d’un caractère par l’utilisateur, l’information est envoyée au modèle (partie).  De manière générale, GameActivity met à jour Partie dés que besoin. Lorsqu’un mot est écrit, ou bien est descendue en bas, ou encore quand le jeu est se met en pause. |
| J’ai un véritable métier | Notre métier respecte les patron de conception étudiées à la période d’avant, notamment :  Le **patron observeur**, utilisé 2 fois :  -Pour la GameLoop, tous les abonnés au GameTimer sont notifiés chaque seconde.  -Pour faire communiquer la Partie et la GameActivity, par exemple pour prévenir GameActivity qu’un nouveau mot à afficher a été ajouté dans Partie.  Ces 2 relations observeurs/observés sont initiées lors de la création de GameActivity, et sont supprimées lorsque l’on change d’activité.  **GameActivity->onResume() ligne 120 :**  partie.attach(this); gameTimer.attach(partie); gameTimer.attach(this);  **GameActivity->Quit() ligne 150 :**  partie.dettach(this); gameTimer.dettach(this); gameTimer.dettach(partie);  Le patron **fabrique** : dans **WordFactory**. Qui permet de créer une instance de WordDatabase (liste de mots) qui va récupérer ses mots dans le bon fichier texte en fonction du WordType. Permet une bonne évolutivité de l’application en rajoutant facilement de nouvelles listes de mots. |
| Je sais séparer la vue du modèle | La vue et le modèle ne communiquent pas.  Seul le context est passé à certaines classes qui doivent avoir accés au ressources du projet. Par exemple notre Wordfactory permet de créer dynamiquement une instance de WordDatabase en fonction d’un WordType. De plus, cela permet dans certains cas un meilleur respect du Single Responsability Principle. |
| Je maitrise de cycle de vie de l’application | Gestion des onPause, OnCreate, onResume etc.  Meilleur exemple dans GameActivity où on enregistre les mots quand on passe dans le onPause ( on stop aussi la musique et ferme le clavier), et rechargement des mots dans le onResume si l’écran est vertical. |
| Je sais utiliser le findViewById | Exemple pris dans GameActivity |
| Je sais gérer les permissions dynamiques |  |
| Je sais gérer la persistance légère |  |
| Je sais gérer la persistance profonde | Les scores sont enregistrés dans le téléphone de l’utilisateur. Ils sont donc retrouvables même après la fermeture de l'application.  ScoreReader et scoreWriter lisent et écrivent dans un fichier sauvegarder dans le téléphone. |
| Je sais afficher une collection de données | Exemple avec ScoreActivity et Activity\_score dans lequels on met des score\_list\_view\_layout |
| Je sais coder mon propre adaptateur | Exemple de scoreAdapteur |
| Je maitrise l’usage des fragments | Les fragments ne sont pas fonctionnels. |
| Je maitrise l’utilisation de GIT | Une branche pour chacun des utilisateurs, code écrit sur la branche, tester et merge quand cela fonctionne (rebase master sur la branche, gérer les conflits sui besoin et merge la branche sur le master quand tout fonctionne, cela permet d’avoir à tout moment un master fonctionnel sans bug) |

Application

|  |  |
| --- | --- |
| Je sais utiliser la caméra | La caméra est utilisé quand vous perdez une partie, vous prenez une photo et elle sera enregistré et retrouvable dans les scores. |