

1 Instructions générales

Suivez attentivement les instructions suivantes. Tout manquement sera sanctionné d'un point négatif.

Créez un nouveau projet en langage Java, nommé selon le schéma “TP7_numéro”, où “numéro” est votre numéro d'étudiant (sans le 'u' qui le précède).

Assurez-vous que la “minSdk” soit bien 19. On peut soit choisir ce paramètre à la création du projet, soit le changer dans le `build.gradle` a posteriori.

Si un morceau de votre code ne compile pas, commentez-le avant de déposer votre rendu sur Eprel : un projet qui ne compile pas recevra 0.

Une fois le projet terminé, exportez votre projet (`File` → `Export` → `Export to Zip File`), et rendez ce fichier sur Eprel. N'archivez pas manuellement le dossier contenant votre code !

2 Présentation du jeu Guacamole

Notre objectif est d'implémenter une version légère du jeu “Whack-a-mole” (littéralement, “Frappe-la-taupe”). Dans ce jeu, une taupe apparaît à un endroit aléatoire de l'écran, et l'objectif du ou de la joueuse est de la frapper (c'est-à-dire, de toucher son emplacement à l'écran) le plus rapidement possible. Dès que la taupe est touchée, elle réapparaît à un nouvel endroit de l'écran, et le jeu continue.

In fine, le jeu devra ressembler à l'image présentée en Figure 5.

Ce devoir est divisé en quatre sections :

- Les Sections 3 et 4 proposent de mettre en place une version minimale du jeu. Ce sont ces sections qui concentreront la majorité des points, aussi ne s'inquiétera-t-on pas si l'on n'a pas le temps de passer à la suite.
- La Section 5 a pour objectif d'attribuer un score à chaque partie, et de se souvenir du meilleur score local, d'une session à une autre.
- Enfin, la Section 6 propose de maintenir en ligne le pseudonyme et le score maximal réalisé par n'importe quel.le joueur ou joueuse.

3 Layout

Question 1 Créez une classe `Taupiniere` héritant de `ConstraintLayout`. Une `Taupiniere` aura les attributs suivants :

- une `ImageView` nommée `taupe`
- quatre `int` nommés `taupeWidth`, `taupeHeight`, `taupeX` et `taupeY`

Question 2 Ajoutez les images `taupe.png` et `taupiniere.jpg` à votre projet Android Studio en tant que ressource, en suivant la procédure illustrée en Figure 1.

Question 3 Dotez `Taupiniere` d'un constructeur acceptant un `Context`. On commencera par demander à la `Taupiniere` de se dessiner (ce n'est pas le comportement par défaut des `ConstraintLayout`), comme suit :

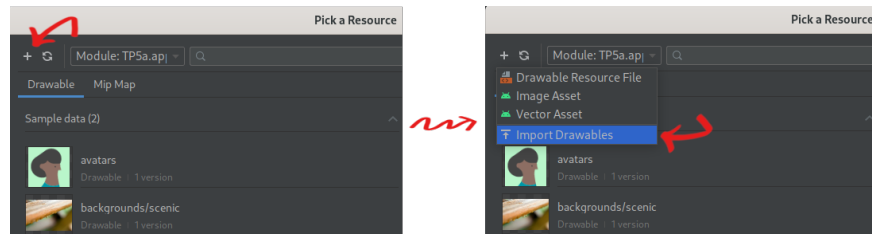


FIGURE 1 – Ajouter une image au projet.

```
public Taupiniere(Context context) {
    super(context);
    setWillNotDraw(false);
    ...
}
```

Dans le reste du constructeur,

- vous affecterez à `taupe` une `ImageView` nouvellement créée. Puis, à l'aide d'un appel à `setImageResource(R.drawable.???)`, vous ferez en sorte qu'elle affiche l'image `taupe.png`
- vous ajouterez cette `View` à la `Taupiniere`, sans spécifier de contraintes
- à l'aide de la méthode `setBackgroundResource()`, vous afficherez l'image `taupiniere.jpg` en arrière-plan de la `Taupiniere`.

Question 4 Tel quel, la taille de la `taupe` dépend de la résolution de l'écran. On veut que la taille de la taupe soit à peu près de la largeur d'un doigt, et ce quelle que soit la résolution de l'appareil. Pour ce faire, on va préciser la taille de la `taupe` en *density-independent pixels*.

Recopiez le code suivant à la fin du constructeur de `Taupiniere` :

```
DisplayMetrics dm = getResources().getDisplayMetrics();
int l = (int) TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_DIP, 100, dm);
taupe.setLayoutParams(new LayoutParams(l, l));
```

Question 5 Dans `onCreate()`, créez une `Taupiniere` et affichez-la sur tout l'écran grâce à `setContentView()`.

À ce stade, votre UI devrait ressembler à la Figure 2.

4 Fonctionnalités élémentaires

On va maintenant gérer le déroulement du jeu.

On rappelle comment créer un nouvel entier tiré aléatoirement (uniformément) entre 0 et $n - 1$:

```
Random r = new Random();
r.nextInt(n); // entier entre 0 et n-1
```



FIGURE 2 – Premier aperçu de l’UI.

Question 6 Réécrivez la méthode `onDraw()` de `Taupiniere` de sorte que la position de la **taupe** soit tirée aléatoirement à chaque fois que la `Taupiniere` se dessine. On fera bien attention à ce qu’aucune partie de la **taupe** ne puisse dépasser des bords.

On s’aidera pour ce faire des méthodes suivantes :

- `View::getWidth()`
- `View::getHeight()`
- `View::setX()`
- `View::setY()`

Question 7 Faites en sorte que lorsque la joueuse ou le joueur touche la **taupe** à l’écran, la `Taupiniere` se redessine - déplaçant ainsi la **taupe** vers une nouvelle position aléatoire.

On a maintenant une version minimale du jeu Guacamole. La Figure 3 illustre le comportement désiré de Guacamole à ce stade.

5 Gestion du score

Il va maintenant s’agir de calculer un score pour chaque partie. Une partie dure 10 tours, c’est-à-dire qu’une fois que la **taupe** a été frappée pour la dixième fois, la partie s’achève, et une nouvelle partie démarre immédiatement.

Avant de s’occuper de calculer le score, on va modifier le layout pour ajouter une fenêtre d’affichage du score.



FIGURE 3 – Aperçu du comportement du jeu, en appuyant deux fois sur la taupe.

Question 8 Dans `MainActivity`, faites en sorte que le `ConstraintLayout` global possède deux enfants : une `Taupiniere` `taupiniere`, comme décrit dans les parties précédentes, ainsi qu’une `TextView` `scoreView`, qui affichera pour l’instant “Score : 0”.

Écrivez des contraintes plaçant `scoreView` au-dessus de `taupiniere`. On formera une chaîne verticale avec ces deux `View`, et on fera en sorte de les centrer horizontalement.

Afin de s’assurer que `taupiniere` occupe toute la place disponible, on pourra s’appuyer sur l’instruction suivante :

```
taupiniere.setLayoutParams(
    new ConstraintLayout.LayoutParams(
        ConstraintLayout.LayoutParams.MATCH_PARENT,
        ConstraintLayout.LayoutParams.MATCH_CONSTRAINT
    )
);
```

Le résultat attendu à ce stade est présenté à gauche de la Figure 4.

Intéressons-nous maintenant au calcul du score. Chaque coup rapporte un nombre de point qui est égal à 1000 moins le nombre de millisecondes qui se sont écoulées entre l’apparition de la `taupe` et le moment où elle a été frappée. On décide de ne pas donner de points négatifs : si le temps de réaction est supérieur à une seconde, le score de ce coup sera donc de 0.

Question 9 En utilisant la méthode `SystemClock.uptimeMillis()` (qui donne l’heure actuelle en millisecondes) et la méthode `MotionEvent::getTime()` (qui renvoie la date d’un événement de toucher dans le même système horaire), faites en sorte qu’après chaque coup, le score total cumulé s’affiche dans `scoreView`.

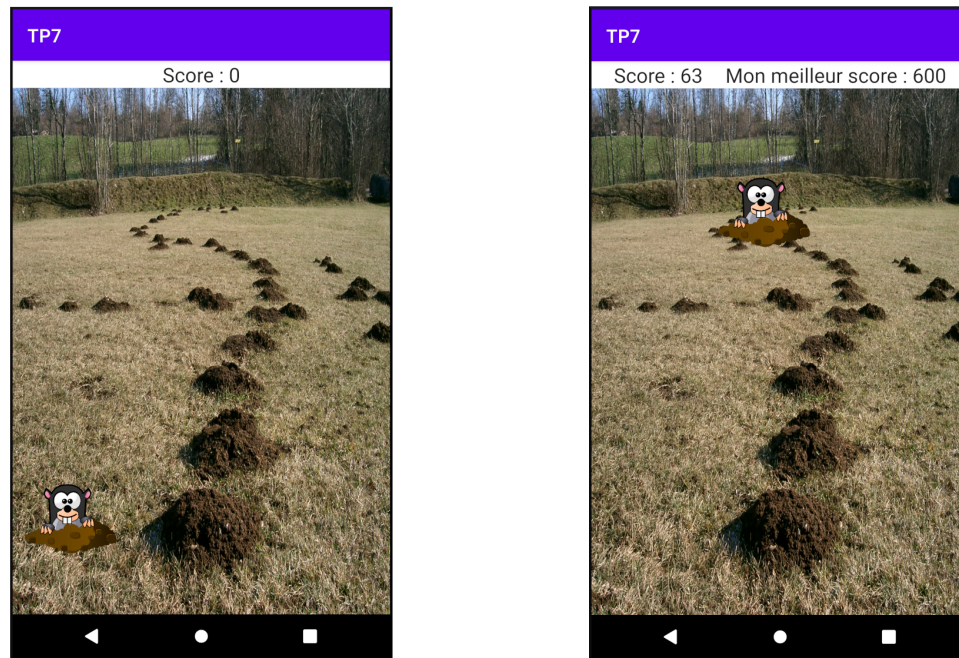


FIGURE 4 – Ajout du score (à gauche), puis du score maximal (à droite).

Pour avoir accès à `scoreView` (qui est une variable de `MainActivity`) depuis `taupiniere`, on pourra recopier la méthode suivante dans `Taupiniere`, qui renvoie la `MainActivity` dont elle dépend :

```
private MainActivity getMainActivity() {
    Context context = getContext();
    while (context instanceof ContextWrapper) {
        if (context instanceof MainActivity) {
            return (MainActivity) context;
        }
        context = ((ContextWrapper) context).getBaseContext();
    }
    return null;
}
```

Pour finir, on veut calculer le score d'une partie, c'est-à-dire calculer le score de chaque tranche de dix coups d'affilée.

Question 10 Faites en sorte que le score se cumule sur dix coups successifs, puis soit remis à zéro.

À la fin de chaque tranche de dix coups, on vérifiera si le score est supérieur au score maximal jamais atteint, et si c'est le cas, on modifiera le score maximal afin de s'en souvenir lors d'une prochaine session. On utilisera pour cela un `SharedPreferences`.

Pour terminer, on ajoutera une autre `TextView` à la droite de `scoreView` qui contiendra le score maximal, comme illustré à droite de la Figure 4.

6 Meilleur score et réseau (💀)

Pour terminer, on veut pouvoir comparer nos scores à ceux des autres joueurs/joueuses en ligne, afin que le score maximal général, ainsi que le pseudonyme de la personne l'ayant atteint, soit répertorié en ligne.

L'url auquel trouver cette information est le suivant :

http://lacl.fr/julien.grange/Enseignements/Programmation_mobile/21_22/TP7/guacamole.php

Le `JSONObject` retourné possède donc deux champs : un champ textuel associé à la clef "pseudo", et un champ entier associé à la clef "score".

Question 11 Commencez par récupérer, à l'ouverture de l'application, les données (pseudonyme et score) du meilleur joueur, pour les afficher dans une nouvelle `TextView`, comme illustré en Figure 5.



FIGURE 5 – Ajout du score du ou de la meilleure joueuse en réseau (ici, MoleSlayer).

Question 12 À l'issue d'une partie où le score obtenu a battu le score maximal, envoyez au serveur distant votre nouveau score, accompagné de votre pseudonyme de prédilection.

On passera ces données sous forme d'un `JSONObject` object ayant le format indiqué plus haut, via une requête POST. Pour cela, on pourra utiliser la version surchargée suivante du constructeur `JsonObjectRequest` :

```
public JsonObjectRequest(int method, String url, JSONObject object,
    Response.Listener<JSONObject> listener,
    Response.ErrorListener errorListener)
```

où `method` prend la valeur constante `POST`, `object` est l'objet JSON représentant notre performance, et le reste des arguments est le même que dans la Question 11.

Indication : le serveur vérifiera que votre score est bien plus élevé que le précédent avant de mettre à jour le meilleur score. Si les scores deviennent trop difficiles à battre et que vous n'arrivez pas à vérifier que votre code fonctionne correctement, n'hésitez pas à nous demander de remettre à zéro le score maximal stocké sur le serveur.