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 recevera 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

La saison des champignons bat son plein. À défaut d'enfiler nos bottes pour aller ramasser cèpes et golmottes, nous allons implémenter un petit jeu de rapidité sur ce thème.

Au cours du jeu, des champignons apparaissent aléatoirement à l'écran, et l'objectif du ou de la joueuse est de les cueillir (en touchant leur emplacement à l'écran) le plus rapidement possible. Attention cependant : la mycologie est un sport dangereux, et certains champignons sont à éviter !

Trois variétés célèbres de champignons retiendront notre intérêt : les Cèpes, les Amanites Phalloïdes et les Amanites Tue-Mouches. Ces champignons sont présentés en Figure 1.
In fine, le jeu devra ressembler à l'image présentée en Figure 5.

(a) Le délicieux **Cèpe bronzé**, que l'on reconnaît à son pied obèse et aux tubes présents sous son chapeau foncé.

(b) Un exemplaire d'**Amanite Phalloïde**, la variété mortelle qui fait le plus de victimes en France. À éviter à tout prix !

(c) L'**Amanite Tue-Mouches**, un champignon hallucinogène qui provoque également des coliques aiguës.

FIGURE 1 – Trois fameuses variétés de champignons.

Ce devoir est divisé en quatre sections :
— les Sections 3 et 4 proposent de mettre en place une version minimale du jeu, où l'on ne joue qu'avec des cèpes. La Section 4 concentrera la majorité des points du sujet.

— dans la Section 5, on ajoutera les Amanites Phalloïdes au jeu.
— enfin, la Section 6 propose d'intégrer les Amanites Tue-Mouches et leur effet hallucinogène.

3 Mise en place du sous-bois

Question 1 Ajoutez les images `sous_bois.jpg`, `cepe.png`, `phalloide.png` et `tue_mouches.png` à votre projet Android Studio en tant que ressource, en ouvrant l'onglet "ressource manager" dans la barre gauche, et en suivant la procédure illustrée en Figure 2.

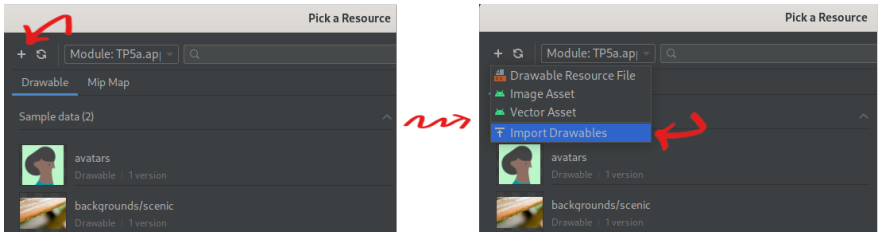

FIGURE 2 – Ajouter une image au projet.

Question 2 On désire limiter le jeu au format paysage. Pour cela, dans le `Manifest`, ajoutez la clef/valeur

```
android:screenOrientation="landscape"
```

comme attribut de l'activité principale.

Question 3 Créez une classe `SousBois` héritant de `ConstraintLayout`. Un `SousBois` aura les attributs suivants :
— deux `int`, nommés `score` (initialisé à 0) et `dim`
— un `Random` nommé `random` (on importera `java.util.Random`)
— un `Handler` nommé `handler` (on importera `android.os.Handler`)
On verra plus loin qu'un `Handler` permet de gérer des tâches à effectuer au bout d'un temps fixé, ce qui sera utile pour faire apparaître et disparaître des champignons à l'écran.

Question 4 Dotez `SousBois` d'un constructueur acceptant un `Context`, en recopiant le code ci-dessous.

```
public SousBois(Context context) {
    super(context);
    setWillNotDraw(false);

    DisplayMetrics dm = getResources().getDisplayMetrics();
    dim = (int) TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_DIP,50,dm);

    handler = new Handler();
    random = new Random();
}
```

Voici quelques explications concernant cette ébauche de constructeur :
— avec `setWillNotDraw(false)`, on demande au `SousBois` de se dessiner (ce n'est pas le comportement par défaut des `ConstraintLayout`).
— on calcule ensuite `dim`, qui sera la taille (en pixels) des images des champignons. Ces deux lignes permettent de faire en sorte que les images soient indépendantes de la résolution de l'écran ("DIP" signifie *density-independent pixels*).
— enfin, on instancie le `Handler` et le `Random`.

Question 5 Toujours dans le constructeur, faites en sorte que le fond d'écran du `Sous-Bois` affiche `sous_bois.jpg`, à l'aide d'un appel à `setBackgroundResource(R.drawable.???)`.

Question 6 Pour finir, débrouillez-vous pour que layout de l'`Activity` principale corresponde exactement à un `SousBois`. On utilisera pour cela `setContentView()`.
À ce stade, votre UI devrait ressembler à la Figure 3.

FIGURE 3 – Premier aperçu de l'UI.

4 Cèpes

4.1 Notre premier cèpe

Question 7 Créez une classe `Cepe`, qui est destinée à représenter cet excellent comestible qu'est le cèpe, cf. Figure 1a. Un `Cepe` possède deux attributs :

— une `ImageView` nommée `view`
— un `SousBois` nommé `sousBois`

Question 8 Implémentez un constructeur de `Cepe` avec les bons arguments pour qu'à sa création,
— `sousBois` reçoive un `SousBois` passé en argument du constructeur
— `view` affiche l'image `cepe.png` (on utilisera `ImageView::setImageResource()`)
— `view` soit ajoutée à `sousBois` (on rappelle que `SousBois` étend `ConstraintLayout`)
— `view` soit un carré dont les dimensions ont été calculées à la Question 4. Pour cela, on pourra recopier la ligne suivante :

```
view.setLayoutParams(new ConstraintLayout.LayoutParams(dim,dim));
```

où `dim` est récupéré en argument du constructeur.

Question 9 Créez un `Cepe` dans le constructeur du `SousBois` : vous devriez maintenant obtenir le résultat présenté en Figure 4.

FIGURE 4 – Un cèpe a poussé en haut à gauche de l'écran !

Question 10 Dans le constructeur de `Cepe`, définissez un callback pour `view.onClickListener()` (n'importe quelle `View` peut implémenter ce callback, pas seulement les `Button` !) de sorte qu'un clic sur l'image ait les deux effets suivants :
— l'image disparaît de l'écran. On utilisera pour cela la méthode `sousBois.removeView()` (c'est le raison pour laquelle on avait besoin de connaître le `SousBois` dans `Cepe`)
— le `score` augmente d'un point.

Pour terminer cette première version du jeu, il nous reste essentiellement deux fonctionnalités à implémenter :

— faire apparaître un cèpe de temps à autre, en tirant aléatoirement chaque position (Section 4.2)
— faire disparaître chaque cèpe au bout d'un certain temps, s'il n'a pas été ramassé (Section 4.3)

4.2 Apparition des cèpes

On cherche donc à faire en sorte qu'un cèpe apparaisse de temps en temps : on décide qu'un nouveau cèpe apparaîtra entre 0 et 2 secondes après le précédent. Pour cela, on va utiliser le `Handler` qui a été initialisé dans la Section 3.

Un `Handler` permet de gérer en ensemble de tâches à exécuter à une certaine date, ou après un certain délai. Chacune de ces tâches doit implémenter l'interface `Runnable`, qui ne possède qu'une seule méthode : `void run()`.
Par exemple, si on veut afficher "Golmotte" dans le log au bout de 1.5 secondes, on pourra le faire de la sorte, en utilisant une implémentation anonyme de `Runnable` :

```
Runnable affichage = new Runnable(){
  @Override
  public void run(){
    Log.v("Test","Golmotte");
  }
};

handler.postDelayed(affichage,1500);
```

On aura donc compris que `Handler::postDelayed` permet d'exécuter la méthode `run()` de son premier argument (qui doit donc être un `Runnable`), après un délai égal à son second argument (dont l'unité est la milliseconde).

Question 11 En utilisant `handler`, modifiez le constructeur de `SousBois` (qui, jusque-là, créait immédiatement le premier `Cepe`) pour que le premier `Cepe` soit seulement créé au bout de 2 secondes.

Pour pimenter le jeu, on va faire en sorte que la position d'un nouveau `Cepe` soit aléatoire. On rappelle comment créer un nouvel entier tiré aléatoirement (uniformément) entre 0 et $n-1$:

```
random.nextInt(n); // entier entre 0 et n-1
```

Question 12 Complétez le code du constructeur de `Cepe` pour que la position de `view` soit aléatoire.

On s'aidera pour ce faire des méthodes suivantes :

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

Les deux premières vous permettront d'obtenir les dimensions du `Sous-Bois` (on se souviendra qu'il ne faut pas les utiliser dans le constructeur, où le `Sous-Bois` n'a pas encore été placé à

l'écran, et donc a des dimensions nulles), tandis que les deux dernières permettront de positionner le `Cepe`.

Question 13 Reprenez la question précédente pour qu'aucun bout de `Cepe` ne dépasse de l'écran. On se souviendra que `dim` correspond à la largeur et à la longueur en pixels de l'image du cèpe.

On va maintenant faire en sorte que plusieurs `Cepe` poussent les uns après les autres.

Question 14 Modifiez le code du `run()` que vous avez écrit à la Question 11 pour que chaque fois qu'un `Cepe` est créé, il programme la création d'un autre `Cepe` après un délai de 2 secondes.

Question 15 Enfin, plutôt que d'avoir un délai fixe de 2 secondes, tirez aléatoirement un délai entre 0 et 2000 millisecondes à chaque création de `Cepe`.

Question 16 Affichez un `Toast` portant le message "Nouveau palier : n points" (où n est le nombre de points) à chaque fois que le `score` dépasse une nouvelle dizaine.

4.3 Disparition des cèpes

Pour éviter de surcharger l'écran, on veut maintenant s'assurer que s'il n'est pas cueilli, un `Cepe` quitte l'écran au bout de 2 secondes.

Question 17 Toujours à l'aide du `handler`, ajoutez un temps limite d'apparition de 2 secondes pour chaque `Cepe`. On veillera évidemment à ce qu'un `Cepe` qui disparaît tout seul ne rapporte pas de points.

5 Amanites Phalloïdes (☠ : champignon mortel)

Nous avons à ce stade une version minimale du jeu ; on chercher maintenant à l'étoffer. Pour cela, on va ajouter une deuxième variété de champignons : les Amanites Phalloïdes, cf. Figure 1b. Les mycophiles en herbe se souviendront que ces champignons sont mortels.

Question 18 Pour éviter de dupliquer du code, on va utiliser le pouvoir de l'orienté objet. Créez une classe abstraite `Champignon`, et basculez tout le code de `Cepe` qui est partagé par tous les champignons dans cette nouvelle classe. Créez une nouvelle classe `Phalloide` et faites hériter les classes `Cepe` et `Phalloide` de `Champignon`. Une `Phalloide` se comportera comme un `Cepe`, mis à part sur les points suivants :

- l'image de fond sera `phalloide.png`
- la consommation d'une `Phalloide` est mortelle : en récolter une ramènera le `score` à 0, et affichera le `Toast` "Très mauvaise idée !" à l'écran.

Question 19 Faites en sorte qu'au moment de faire pousser un champignon, il y ait une chance sur deux que ce soit un `Cepe`, et une chance sur deux que ce soit une `Phalloide`.

6 Amanites Tue-Mouches (☠ : section de difficulté mortelle)

Pour terminer notre jeu, on va ajouter un champignon toxique au mix : l'Amanite Tue-Mouches, cf. Figure 1c. En plus de provoquer des vomissements sévères, ce champignon non-comestible possède des propriétés hallucinogènes.

Question 20 Créez la classe `TueMouches`, qui hérite de `Champignon`, avec les particularités suivantes :

— son image de fond est `tue_mouches.png`

— ramasser une `TueMouches` fait baisser le score de 3 points (le `score` ne pourra pas devenir négatif).

Modifiez votre code pour que chaque variété de `Champignon` apparaisse un tiers du temps.

Pendant 10 seconde après chaque ingestion de `TueMouches`, on veut que l'utilisatrice subisse des hallucinations visuelles. Plus précisément, chaque nouveau `Champignon` qui apparaîtra pendant ce laps de temps devra se déformer pendant toute sa durée de vie.

Question 21 Créez deux `ObjectAnimator` via la méthode `ObjectAnimator.ofFloat()` : à la différence de l'`ObjectAnimator` vu en cours, on utilisera les propriétés "scaleX" et "scaleY", qui permettent d'étirer chaque dimension d'une `View` en suivant une succession de facteurs `float`. On veut donc que chaque `Champignon` qui pousse pendant l'hallucination voit successivement sa taille doubler, puis être diminuée de moitié, puis doubler à nouveau, etc. (de manière continue) pendant toute sa durée de vie. On utilisera `ObjectAnimator::setDuration()` pour fixer la durée de l'animation.

Question 22 Assurez-vous maintenant que la consommation d'une nouvelle `TueMouches` prolonge l'hallucination. À titre d'exemple, si on consomme une `TueMouches` 3 secondes après le départ, puis une autre 9 secondes après le départ, alors l'hallucination devra s'étendre entre 3 secondes et 19 secondes.

FIGURE 5 – Moins de 10 secondes après l'ingestion d'une `TueMouches`, nous sommes en pleine hallucination : au moment de la capture, la `TueMouches` présente à l'écran vient de terminer de réduire en taille et s'apprête à grossir à nouveau, tandis que la `Phalloide` est en train de grossir en parallèle. Le `Cepe` est quant à lui dans une phase intermédiaire, où il a presque sa taille normale.