Rapport développement pour mobile

Fabien HOARAU & Erwan GUICHARD, L3 informatique

14 avril 2019

Résumé

Dans ce rapport, nous allons vous présenter notre application mobile développée sous Swift et Android Studio. Nous nous sommes répartis les tâches : Fabien s'occupe de la partie iOS et Erwan, Android.

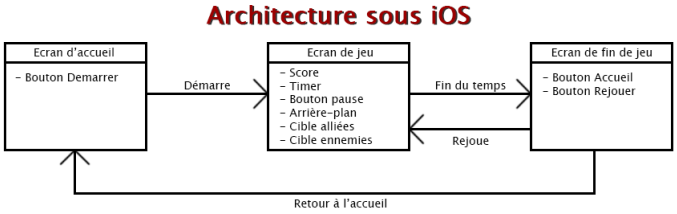
1 Introduction

Notre jeu se nomme "Shoot It !", le but du jeu est de "tirer" sur des voleurs tout en évitant les civils. Si on touche un voleur, on gagne un point et si on touche un civil, on perd alors cinq points tout cela dans un temps imparti.

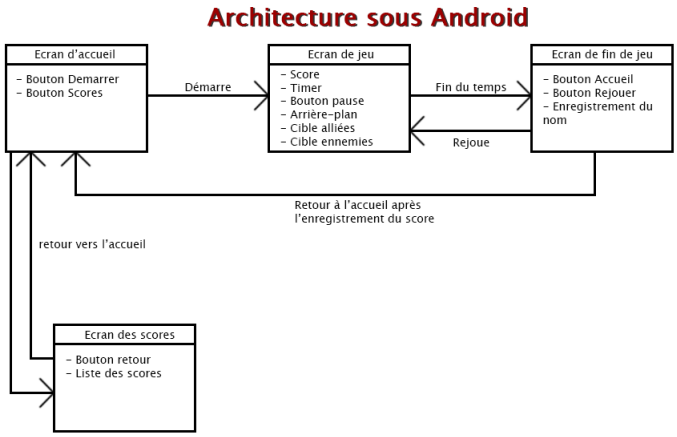
2 Description générale de l'application

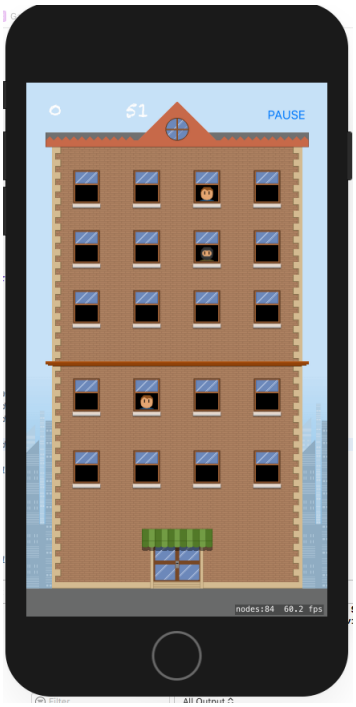
Voici une capture d'écran d'une partie en cours sous iOS :

2.1 Architecture de l'application sous iOS



2.2 Architecture de l'application sous Android



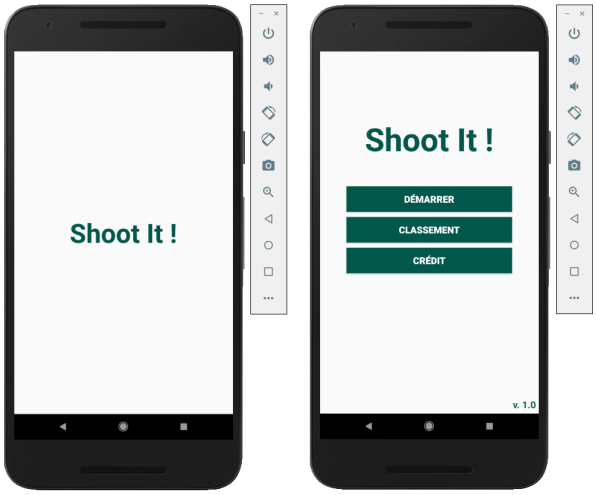


Le joueur dispose d'une minute pour toucher le plus de voleur possible. À la fin de la partie, le joueur enregistre son nom et le meilleur score sera affiché en tête d'une liste (cette fonctionnalité a été réalisée uniquement sous Android).

3 Code Android et IOS

3.1 Android

Le code est organisé de la façon suivante : Un écran de chargement qui affiche le nom du jeu puis on arrive sur l'écran d'accueil du jeu :

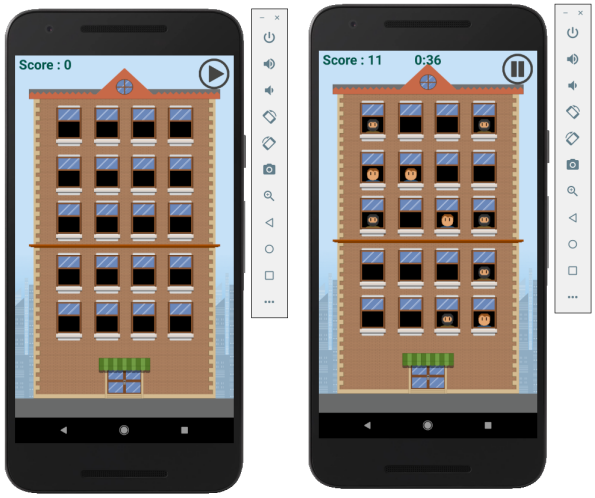


L'acceuil contient trois bouton :

1. Démarrer : Pour lancer le jeu
2. Classement : Pour afficher les scores réalisés
3. Crédit : Pour afficher les crédits du jeu

1. Démarrer :

La partie Démarrer nous affiche l'immeuble vide puis quand on clqiue sur le bouton Play en haut à droite le jeu se lance.



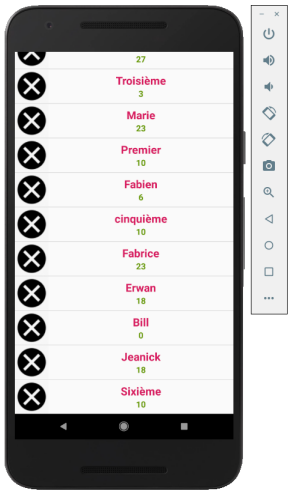
Voici une partie importante du code de l'activité *GameActivity*, l'affichage des fenêtres. Elle consiste à généré des boutons fenêtres avec aléatoirement un voleur, un civil ou une fenêtre vide et qui incrémentera ou décrémentera le score selon la situation via la fonction *addScore*.

```
//fonction pour la création des fenêtres interactives
public void newFenetres(){
    for (int i = 0; i <imageButtons.length ; i++) {

        String imageButtonID = "fen"+(i+1);
        int resID = getResources().getIdentifier(imageButtonID,"id",getPackageName());
        imageButtons[i] = ((ImageButton) findViewById(resID));
        final int[] img = new int[]{R.drawable.fenetre_vide,R.drawable.fenetre_voleur,R.drawable
        final Random rand = new Random();

        final int randomImgFen = rand.nextInt(img.length);

        imageButtons[i].setImageDrawable(getResources().getDrawable(img[randomImgFen]));
        imageButtons[i].setSoundEffectsEnabled(false);
        imageButtons[i].getBackground();

        buttonState[i]= randomImgFen;
        buttonID.add(i, "" + imageButtons[i].getId());

        final int finalI = i;
        imageButtons[i].setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if(timerRunning) {
                    addScore(v.getId() , finalI);
                }
            }
        });
    }
}
```

Une fois le temps écoulé, le jeu se quitte automatiquement et amène le joueur vers l'écran de fin et lui demande d'entrer son nom. S'il n'entr epas son nom un message lui dira qu'il faut entrer un nom et une fois cela fait il verra l'écran de fin qui lui proposera de rejouer ou de revenir à l'accueil.

1. Classement :

La partie classement référence tous les scores réalisés en jeu.

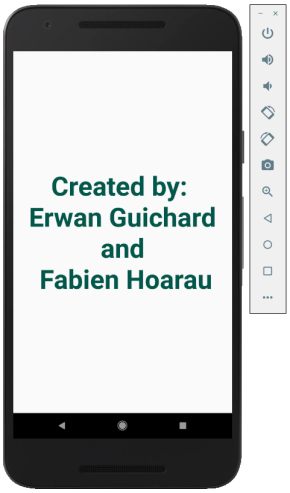


Voici le code de l'activité *RankingActivity* permettant la récupération des résultats de tous les joueurs dans les données de préférence du jeu qui permet la sauvegarde de toutes les données même après la fermeture de l'application :

```
SharedPreferences setting = getApplicationContext().getSharedPreferences("PREFS", Context.MODE_P
    final Map<String, ?> getList = setting.getAll();

    //stockage des données dans une liste pour traiter par la suite
    for (Map.Entry<String,?> entry : getList.entrySet()){
        entry.getKey();
        entry.getValue();
        listOnView.put(entry.getKey(),String.valueOf(entry.getValue()));
    }

    //création de l'adapteur pour mapper les données statiques aux vues définies dans le fichier
    adapter = new SimpleAdapter(this,listItems,R.layout.activity_list_item,
            new String[]{"Name","Score"},
            new int[]{R.id.text1,R.id.text2});

    //stockage du nom et du score des joueurs dans la liste dela vue
    Iterator it = listOnView.entrySet().iterator();
    while(it.hasNext()){
        HashMap<String,String> resultsMap = new HashMap<>();
        Map.Entry pair = (Map.Entry)it.next();
        resultsMap.put("Name", pair.getKey().toString());
        resultsMap.put("Score",pair.getValue().toString());
        listItems.add(resultsMap);
```

1. Crédit :

La partie Crédit affiche juste nos noms parce qu'on trouvait ça marrant :)



3.2 iOS

La classe *WhackSlot* :

```
class WhackSlot: SKNode{
    var charNode: SKSpriteNode!

    var isVisible = false
    var isHit = false

    func configure(at position: CGPoint){
        self.position = position    //La position du "Node"

        let sprite = SKSpriteNode(imageNamed: "fenetre")
        addChild(sprite)

        let cropNode = SKCropNode()
        cropNode.position = CGPoint(x:0, y:-12)
        cropNode.zPosition = 1
        cropNode.maskNode = SKSpriteNode(imageNamed: "mask")

        charNode = SKSpriteNode(imageNamed: "good")
        charNode.position = CGPoint(x: 0 , y: -88)
        //charNode.name = "character"
        cropNode.addChild(charNode)

        addChild(cropNode)
    }
```

Cette sous-classe de SKNode est en grande partie le corps du jeu, c'est elle qui va encapsuler les fonctionnalitées liées aux fenêtres et aux cibles. Cela évite de surcharger la scène avec du code.

4 Quelques points délicats/intéressants

4.1 Android

La suppression des scores et le tri : La phase de suppression et de tri était assez compliquée à gérer à cause du format de stockage des données des joueurs. Nous n'avons donc pas du finalisé

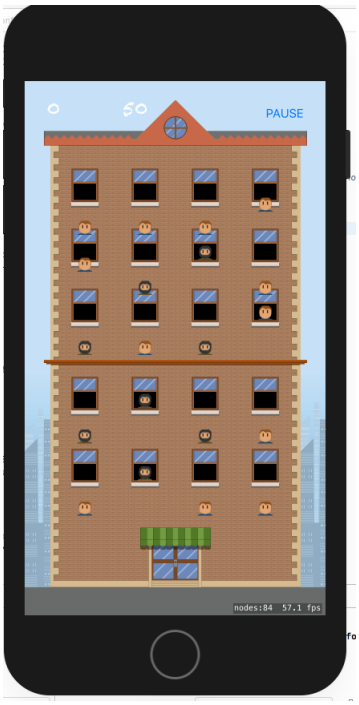
cette partie qui est quand même assez importante.

La détection des boutons : cette phase nous a vraiment posées pas mal de soucis car il fallait donner à chacun un id différent pour pouvoir les différencier et par la suite tester quels types de boutons ils sont.

Le spam clic : Un gros bug faisait que spammer une fenêtre entre chaque actualisation de fenêtre permettait au joueur d'incrémenter son score plusieurs fois sur la même fenêtre, mais le fait de mettre un id à chaque bouton à permis de résoudre aussi ce problème.

4.2 iOS

Le **CropNode** : Le CropNode est une classe issue de SKNode, c'est un masque qui va servir à cacher nos cibles pour donner l'illusion d'apparition lorsque celles-ci vont se mettre à bouger sur l'axe y. Sans le masque, les cibles sont en réalité placées de cette façon.



5 Conclusion

Pour conclure ce rapport, nous tenons à remercier la documentation d'Apple[2] et d'Android[1], le site Hacking With Swift[3] pour les références concernant l'architecture du jeu, ainsi que la chaîne Youtube[4] *The Swift Guy* pour les bases de Swift concernant les View controllers.

Références

[1] Android developers. https://developer.android.com.

[2] iOS developer. https://developer.apple.com.

[3] Hacking with swift. https://www.hackingwithswift.com.

[4] The swift guy. https://www.youtube.com/channel/UC-d1NWv5IWtIkfH47ux4dWA/.