Breakout on Android / iOS

BOOZ Allan - ETTOUIL BOUDEMIA Yanis, L3 informatique

14 avril 2019

Table des matières

Résumé

Dans ce rapport, nous allons vous présenter notre application "Breakout" aisni que les étapes de son développement

1 Introduction

Breakout est une application mobile sous Android et iOS dont le principe est plutot simple. En effet c'est un jeu qui reprend le principe du Casse-briques dans lequel l'objectif est de reussir à casser toutes les briques présentes à l'aide d'une balle que l'on fait rebondir sur une raquette.

2 Corps

Afin de produire un jeu de ce type il a fallu reflechir à la façon de s'y prendre et reussir à identifier les differents points clés du développement de l'application.

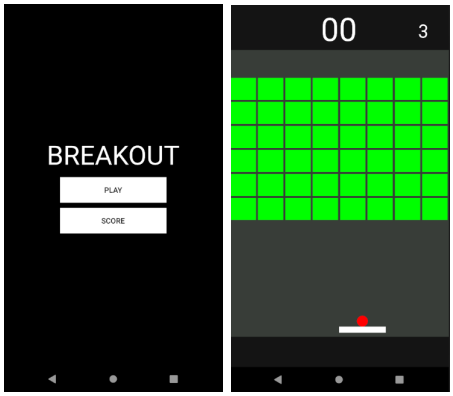
2.1 Android

2.1.1 Architecture de l'application

— Une classe Brique pour définir chaque brique de maniere indépendante.
— Une classe Cercle qui servira à gérer la création de la balle ainsi que ses déplacements.
— Une classe Paddle pour créer et gérer la raquette.

2.1.2 Fonctionnement du jeu

Au lancement de l'application, nous sommes invités sur l'écran du **Menu**, composé de 2 boutons : Jouer et Score, menant tous deux à leurs écrans respectif : écran du jeu et écran des scores.



Une fois arrivé dans le jeu, le joueur peut commencer à faire bouger la balle en touchant l'écran. Devant lui, le joueur fait face à un mur de briques avec une certaines résistance *(résistance de niveau 3)* pour chaque brique.

Il dispose également d'une raquette *(Paddl)* afin de faire bouger au mieux la balle et d'éviter qu'elle atteigne le bas de l'écran. Enfin il suffit au joueur de toucher la partie droite ou gauche de l'écran pour voir la raquette partir dans la direction voulue.

Attention, le jeu dispose d'un système de vie, c'est-à-dire 3 coeurs dont le seul moyen d'en perdre est de laisser la balle toucher le bas de l'écran. Quand les 3 coeurs ont été consommé, la partie se termine sur une défaite et vous emmène directement à l'écran des scores. Ainsi le seul moyen de gagner est de détruire toutes les briques, ce qui revient à vous rediriger vers l'écran des scores à la fin de la partie.

Après une défaite ou une victoire, l'écran des score apparaît et vous propose d'entrer un pseudo afin d'enregistrer votre score. Dans ce tableau des scores, il ne peut exister que 5 scores enregistré, nous informant dans l'ordre décroissant, le meilleur score jusqu'au score le plus bas.

Le déroulement expliqué ci-dessus est géré par la fonction **update()**

```
public void update() {
   //Mise à jour de la raquette et de la balle
   paddle.update(screenWidth);
   cercle.move(fps);
   // Check for ball colliding with a brick
   for (int i = 0; i < nbBricks; i++) {
       if (bricks[i].getVisibility() && collisionBrique(cercle,bricks[i])) {
           if(bricks[i].getRes() > 0) {
               bricks[i].setRes();
               collisionGaucheDroite(cercle,bricks[i]);
           }else{
               bricks[i].setInvisible();
               bricks[i].setRes();
               collisionGaucheDroite(cercle,bricks[i]);
               score+=1;
               if (score==6*8) {
                   paused = true;
                   BuildAWall();
               }
           }
       }
   }
   // Check for ball colliding with paddle
   if (collisionPaddle(cercle, paddle)) {
       cercle.setRandomXVelocity();
       cercle.reverseYVelocity();
       cercle.clearObstacleY((int)paddle.getRect().top - 2);
   }
   //Check for ball colliding with screen
   if (cercle.getX() + cercle.getXSpeed() < cercle.getRayon() ) {
       cercle.reverseXVelocity();
   }
   if (cercle.getY() + cercle.getYSpeed() < brickHeight*2) {
       cercle.reverseYVelocity();
   }
   if (cercle.getX() + cercle.getXSpeed() > (screenWidth - cercle.getRayon())) {
       cercle.reverseXVelocity();
   }
   if (cercle.getY() + cercle.getYSpeed() > (screenHeight- cercle.getRayon())) {
       cercle.reverseYVelocity();
       lives--;
       // Restart game
       if (lives == 0) {
           paused = true;
           BuildAWall();
       }
   }
}
```

2.1.3 Score et persistance

Nous avons choisi d'integrer le score et sa persistance dans vette version du jeu. Arbitrairement nous avons décidé de ne garder que les 5 meilleurs score en mémoire et pour ce faire, nous avons utilisé l'outil **SharedPreference**, qui permet de stocker dans le cache de l'application des valeurs auxquelles sont assosciées des clés pour y acceder.

Par exemple cette commande permet de récuperer la valeur dont la clé est ***LatestScore*** *et de récuperer 0 si cette valeur n'existe pas*

```
prefs.getInt("LatestScore", 0);
```

Les valeurs des scores sont mémorisées directement dans les SharedPreferences et leurs clés respectives sont les noms des joueurs. Ces mêmes noms sont alors stockés dans une liste ce qui ne nous permettra par la suite de pouvoir trier les noms des joueurs en fonction de leur scores.

```
public void addPlayer(String n){
    int LatestScore = prefs.getInt("LatestScore", 0);
    boolean replace = false;
    for(int i =0; i<PlayerList.size(); i++){
        if(PlayerList.get(i).equals(n)){
            replace = true;
            if(prefs.getInt(PlayerList.get(i), 0) < LatestScore){
                editor.putInt(n, LatestScore);
                date_editor.putString(PlayerName.getText().toString(), currentDate);
            }
        }
    }
    if(replace==false){
        PlayerList.add(n);
        editor.putInt(n, LatestScore);
        date_editor.putString(PlayerName.getText().toString(), currentDate);
    }
    editor.apply(); date_editor.apply();
}
```

La fonction **addPlayer()** est celle qui s'occupe d'ajouter un nouveau score dans nos SharedPreferences et dans la liste. Nous y avons également ajouté une condition dans le cas ou un joueur enregistre plusieurs scores à la suite. Nous gardons donc le score le plus élevé pour chaque joueur.

Ici le classement est realisé dans l'ordre décroissant (Le score le plus haut est donc le premier de la liste tandis que le plus bas est le dernier. Nous avons donc une fonction **SortList()** qui va s'occuper de trier les joueurs selon leurs scores et obtenir la liste des joueurs dans l'ordre désiré.

```
public void SortList(){
    boolean sorted = false;
    int taille = PlayerList.size();
    while(!sorted){
        sorted=true;
        for(int i=0; i<taille-1; i++){
            int val1=prefs.getInt(PlayerList.get(i),0);
            int val2=prefs.getInt(PlayerList.get(i+1),0);
            if(val1< val2){
                String temp = PlayerList.get(i);
                PlayerList.set(i, PlayerList.get(i+1));
                PlayerList.set(i+1, temp);
                sorted = false;
            }
        }
        taille--;
    }
}
```

Enfin nous devons mémoriser les noms des joueurs présents dans cette liste afin de pouvoir à leurs scores de maniere persistante. Il suffit donc d'ajouter aux SharedPreferences les noms des jouerus présents dans la liste. Pour éviter une trop grandes quantité de noms sauvegardés, nous faisons en sorte de vider notre base puis d'y ajouter les nouveaux noms des joueurs. Cela permet de ne pas avoir plus de 5 noms mémorisés à la fois.

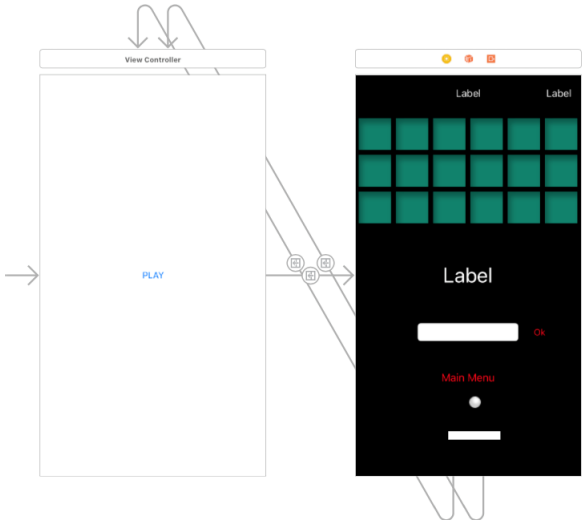
```
public void SaveList(){
    player_editor.clear();
    player_editor.apply();
    for(int i=0; i<PlayerList.size(); i++){
        String key ="best"+Integer.toString(i+1);
        player_editor.putString(key, PlayerList.get(i));
        player_editor.apply();
    }
}
```

2.2 iOS

2.2.1 Architecture de l'application

Pour la version iOS nous avons modifié notre approche du probleme et avons opté pour l'utilisation d'images plutot que pour la création d'objets tels que les Briques/Cercles/Paddle que nous avons fait en Android auparavant. Notre jeu se présente donc de la maniere suivante :

— ViewController = Menu du jeu
— GameViewController = Le jeu en lui même



2.2.2 Le jeu

Dans *Breakout* tout repose sur le déplacement de la balle ainsi que ses interractions avec les briques et la raquette.

```
timer = Timer.scheduledTimer(timeInterval:0.005, target:self,
        selector:#selector(self.boucleJeu), userInfo:nil, repeats:true)
```

C'est pourquoi nous avons besoin d'un timer qui va repeter **boucleJeu()** toute les 0.005 secondes (soit 200 executions par seconde pour obtenir une certaine fluidité), jusqu'à la fin de la partie.

Cette fonction suit le déroulement suivant :

— Modifier les coordonnées de la balle
— Vérifier une possible collision avec les briques ou la raquette
  — - Dans le cas d'une collision avec une brique, cette même brique est détruite et le sens de déplacement de la balle est modifié. De plus le score est incrémenté de 1.
  — - Dans le cas d'une collision avec la raquette, la balle voit son sens de déplacement vertical inversé
— Si toutes les briques sont détruites, alors la partie s'arette : **Vous avez gagné**
— Si la balle touche le bord bas de l'écran, le joueur perd une vie
— Si toutes les vies sont épuisées, alors la partie s'arette : **Vous avez perdu**

Enfin il reste à mettre en place le déplacement de la raquette (*Paddle*).

```
override func touchesMoved(_touches: Set<UITouch>, with event: UIEvent?){
    let touch = touches.first!
    let location:CGPoint = touch.location(in: touch.view)
    paddle.center = CGPoint(x:location.X, y:paddle.center.y)
}
```

Cette fonction va donc permettre à la raquette de suivre le déplacement du doigt sur l'écran mais uniquement sur son axe des abscisses pour qu'elle ne puisse se déplacer qu'horizontalement.

3 Conclusion

Notre projet de création de jeu mobile pour Android et iOS a donc abouti au développement de *Breakout*, notre version du casse-brique, que nous continueront de faire évoluer au fil du temps.

Références

[1] https://stackoverflow.com/questions/23024831/android-shared-preferences-example

[2] https://openclassrooms.com/fr/courses/2023346-creez-des-applications-pour-android/2028219-apprenez-a-dessiner

[3] https://fr.jeffprod.com/blog/2015/les-bases-d-un-jeu-android-en-2d/

[4] https://yal.cc/rectangle-circle-intersection-test/

[5] https://openclassrooms.com/fr/courses/973182-creez-des-applications-pour-iphone-ipad-et-ipod-tou 972249-un-jeu-de-casse-briques