Rapport développement mobile (Cassebrique)

ŚALVAN Fabien, L3 informatique

27 avril 2018

Résumé

Ce rapport contient l'explication et l'architecture d'un jeu de type Casse brique LaTeX.

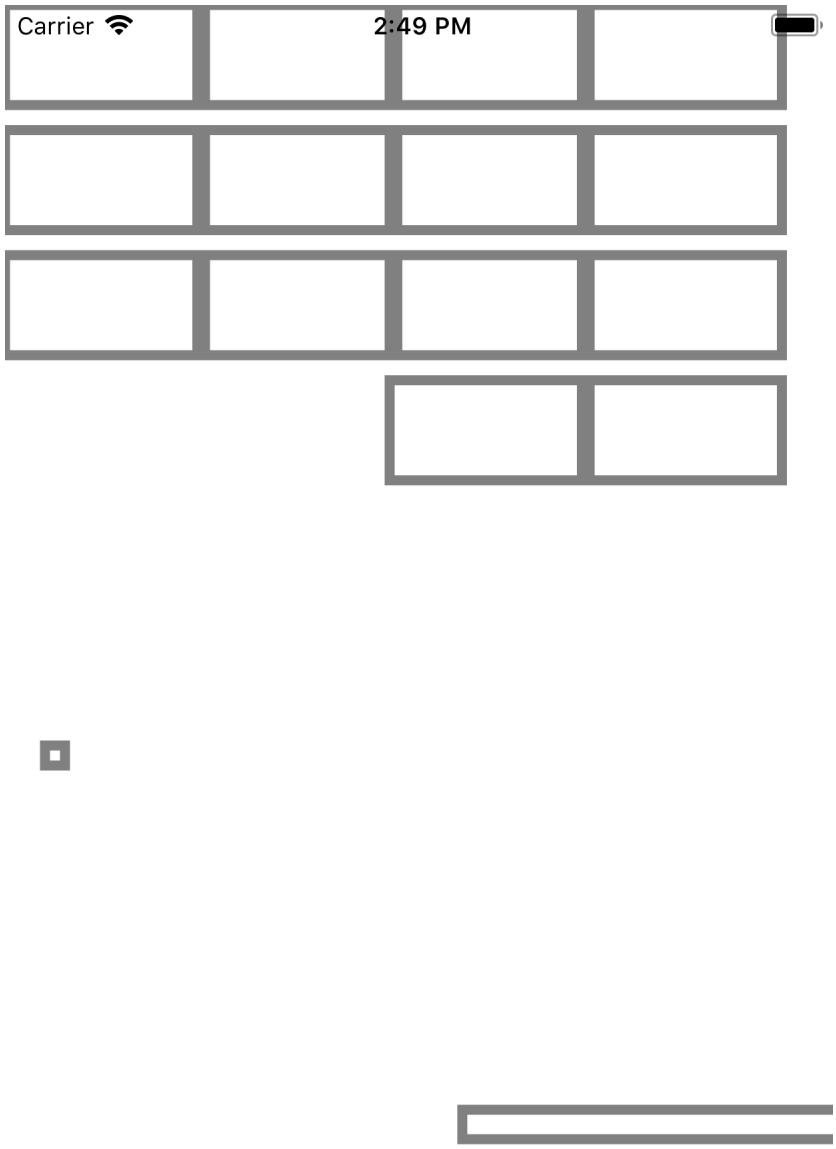
1 Introduction

Dans le cadre de la matière de développement mobile , on a du développer un jeu a sortir sur android et sur IOS . Ce rapport contient la présentation de l'application sur les différentes plates-formes , ainsi que l'architecture plus détaillée du code .

2 Description générale de l'application

Vue de l'écran de jeu : Android

IOS



Cette application est un casse brique . Le jeu contient ces éléments principaux : -La Raquette : elle sert a empêcher la balle de tomber hors de l'écran . Elle est contrôlée par le joueur , et suis la position du doigt sur l'écran .

-La balle : c'est elle qui est envoyée sur les briques afin de les détruire pour compléter les niveaux . Si elle tombe en dessous de la raquette , le joueur perd une vie . Si le joueur n'as plus de vie , la partie est terminée .

Le jeu est constitué de différents niveaux , de difficultés croissante . Le joueur doit finir le

niveau actuelle (en détruisant toutes les briques) afin d'accéder au niveau suivant .

2.1 Déroulement d'une partie

A cause du temps imposé pour faire ce projet , et des difficultés détaillées dans la partie 4 , la version IOS de l'application n'as pas pu être totalement finie . Cette partie va donc détailler le déroulement de l'application pour les 2 systèmes .

2.1.1 Android

L'application se lance sur le menu principal , de la le joueur peut démarrer une nouvelle partie ou consulter les scores . En démarrant une nouvelle partie , le joueur se retrouve avec 3 vie au niveau 1 . Pour passer de niveau , il doit détruire toutes les briques du niveau . Chaque brique rapporte un score de 10 points .Si le joueur quitte l'application , sa partie est automatiquement sauvegardée et il peut la charger depuis le menu de chargement. Si le joueur perd , il est envoyé a l'écran de game over , ou il a la possibilité de sauvegarder son score en entrant son nom et en cliquant sur sauvegarder . Il peut aussi choisir de recommencer une partie ou quitter l'application . Si il recommence , il retourne au niveau 1 avec 3 vies .

Si le joueur finis les 3 niveaux proposés , il est envoyé sur l'écran de victoire , ou son score est aussi automatiquement enregistré . Toutes les personnes ayant fini les 3 niveaux ont le même score , car ils auront détruit le même nombre de brique .

2.1.2 IOS

L'application se lance directement sur l'écran de jeu . Il n'y a pas d'autre fenêtre ou de menu . Comme pour android , le joueur commence au niveau 1 avec 3 vies . Si il perd toutes ses vies , l'interface de jeu disparaît et la mention game over apparaît sur l'écran . Après un délai de 5 secondes , le jeu se relance au niveau 1 avec 3 vies . Si le joueur finis les 3 niveaux , la mention Game Completed appairait a la place , et tout comme le game overt , le jeu recommence au niveau 1 avec 3 vies au bout de 5 secondes .

3 Architecture du code

3.1 Android

Pour la partie Android , l'application est divisée en plusieurs Classes . Chacune des grande classe ci dessous sont relié a une view portant le même nom

3.1.1 Game

C'est la view qui gère l'écran de jeu . Pour des raisons de clarté , cette view utilise plusieurs classes permettant le bon fonctionnement du casse brique . Cette classe est reliée a toutes les classes cités dans cette liste , de manière a ce que chacune de ces classes peuvent communiquer/accéder a des informations contenue dans une autre de ces classes. De plus , c'est dans cette classe que les fonctions principales du jeu sont regroupées : elle contient la boucle principale qui va gérer tout les événements du jeu .

- Ball : c'est la classe représentant la balle . Une balle est représentée par un rect (x , y , longueur , largeur) , ainsi que par sa vitesse . A chaque frame (itération de la boucle principale) , les coordonnée de la balle sont additionnées avec sa vitesse .
- Raquette : c'est la classe représentant la raquette . Elle est aussi représentée par un rect .
- Brick : c'est la classe représentant une brique . Comme la raquette , elle est représentée par un rect . Contrairement aux objet ci dessus , il y a un nombre indéterminée de brique dans une scène . Les briques sont donc stockées dans une liste de la classe Game . Ensuite , chaque brique de cette liste est dessinée et vérifie si aucune balle ne leur est entrée dedans .

— BrickGenerator : c'est une classe dont la principale fonction sert a generer un mur de brique , permettant de facilement créer un niveau . La fonction prend en argument x et y , et génère X colonnes de brique contenant Y brique chacune , et adapte la taille des briques pour que elle prennent toute la largeur de l'écran .
— Gamestate : c'est la classe représentant l'état actuel de la partie . Elle contient les variables permettant de voir la quantités de balles restantes , le niveau actuel , le score du joueur . Elle génère aussi automatiquement le niveau suivant une fois toutes les briques détruites .
— Slot : Cette classe est juste une structure permettant de sauvegarder toutes les informations nécessaires a la sauvegarde/chargement d'une partie . Elle est sérialisable , de manière a être enregistrée dans un fichier . Elle ,ne contient qu'un constructeur , dont le code permet de bien visualiser les données sauvegardées .

```
public Slot(String Name , List<Brick> Bricks , int Level , int Score , int Ball_left){
    name = Name;
    bricks = Bricks;
    level = Level;
    score = Score;
    ball_left = Ball_left;
   date = new Date().toString();
}
```

— ScoreSlot : Cette classe a la même utilité que la classe slot , mais est utilisée pour sauvegarder les meilleur score uniquement . Elle ne prend que le nom et le score du joueur
— Save : Cette classe sert a gérer la sauvegarde et le chargement .Elle ne contient que deux paramètre important

```
public  List<Slot> saves;
```

qui est une liste de la classe slot , qui permet d'avoir accès a toutes les sauvegardes existante.

```
public  List<ScoreSlot> scores;
```

qui est une liste de la classe ScoreSlot , qui permet d'avoir accès a touts les scores existants.
Les deux classes vont chercher dans la mémoire du téléphone si un fichier de sauvegarde existe déjà . Si c'est le cas , elle l'ouvre et charge en mémoire la liste contenu dedans . Si le fichier n'existe pas , la classe va initialiser une liste vide (afin de ne jamais pointer vers une liste null pour la sauvegarde)
Pour la sauvegarde , a chaque fois que la classe game est détruite , elle va automatiquement créer un slot , qu'il va rajouter a la liste de slot de cette classe . Pour les scores , la fonction de sauvegarde sera appelée uniquement si le joueur souhaite sauvegarder son score . La sauvegarde va fonctionner comme pour sauvegarder les parties , c'est a dire en serializant une liste contenant touts les scores . Mais en plus de cela , la fonction va classer les scores par ordre du meilleur score vers le pire . Pour cela , la classe ScoreSLot doit avoir les éléments suivants :

```
implements Comparable<ScoreSlot>

 public int compareTo(@NonNull ScoreSlot toSort)
```

La fonction compareTo doit renvoyer -1 si l'élément comparé est inférieur , 0 si les 2 éléments sont égaux , ou 1 si l'élément comparé est supérieur .
Une fois la classe a classer déclarée correctement , il suffit d'employer cette fonction sur la liste de classe :

```
Collections.sort(scores);
```

scores étant la liste de scores .
Cette classe va ensuite se charger de sérializer cette liste afin de la sauvegarder de manière

persistante.

3.1.2 CanvasView

C'est la view qui permet le Dessin des objets (brique , balle , ect) sur l'écran de jeu . Elle est incluse dans la View principale , afin de pouvoir afficher simultanément des informations texte comme le score , les vies restantes , ect .

3.1.3 Load

Cette classe permet de Charger une partie . Elle est constituée d'une liste , donc chaque cellule est constituée des parties suivantes :

— Le nombre de vie restante au joueur
— Le score actuel du joueur
— La date a laquelle la partie a été enregistrée
— Un bouton pour charger la partie associée
— Un bouton pour supprimer la partie associée

3.1.4 Score

Cette classe permet de consulter les scores . Elle est constituée d'une liste , donc chaque cellule est constituée des parties suivantes :

— Le nom du joueur
— Le score actuel du joueur

3.1.5 Result

Cette classe apparaît quand le joueur finis une partie (que ce soit une victoire ou un game over) . Elle indique le score que le joueur a fait durant cette partie , ainsi que différentes options , comme relancer une partie ou quitter le jeu . Elle permet aussi de sauvegarder le score du joueur dans la liste des meilleurs scores .

3.1.6 Menu

Cette Classe sert a gérer le menu d'accueil . Il n'y a pas de code complexe dans cette classe . Elle ne contient que des fonctions pour accéder aux autre scènes du jeu .

3.2 iOS

Pour la partie IOS , l'architecture a été inspirée de celle déjà utilisée pour Android , mais a cause des différences entre les 2 langages , ainsi que par manque de temps , l'architecture n'est pas entièrement identique

3.2.1 ViewController

Cette classe est celle qui gère la vue de l'écran de jeu . Contrairement a android ou toutes les fonctions ont été mises dans l'équivalent de cette classe , sur IOS , cette classe sert uniquement a instancier la view personnalisée permettant de dessiner sur l'écran , ainsi que de mettre a jour les textes d'information (points de vie , etc)

3.2.2 Raquette

Cette Classe représente la vue personnalisée , servant a l'affichage des éléments dessinées (raquette , balle , briques) Contrairement a l'architectures android , toutes les classes servant au bon fonctionnement ce celle ci on été déclarée directement dans le même fichier comme sous classes , voir simplement en fonctions , présentées ci dessous :

Fonctions :

— Reset() : Contrairement a android ou l'écran de jeu est réinitialisé en le détruisant puis en le recréant , pour IOS cette fonction sert a réinitialiser une partie . Elle vide la liste de briques , remet la vie a 3 , le score a 0 , et ensuite recharge le niveau 1 . Elle permet aussi de réinitialiser le timer qui permet de faire le délai entre la fin de la partie et le début d'une nouvelle .

— Generate(x , y) : Cette fonction génère x colonnes de y briques . Elle adapte directement la taille des briques a l'écran .

— Cette fonction permet de rafraîchir l'image a intervalle régulier .

```
@objc func update(){
    self.setNeedsDisplay()
}
```

— Elle est appelée grâce a la fonction suivante qui appelle update toutes les 0.02 secondes , soit a 50 images par seconde . :

```
timer = Timer.scheduledTimer(timeInterval: 0.02, target: self, selector: #selector(sel
```

La conséquence de ces deux fonctions font que c'est la fonction draw() qui est appelée au final .

— Dans la fonction draw , j'ai rassemblé tout le moteur physique et graphique du jeu . Dans cette fonction , la position de la balle est updaté a chaque frame en fonction de sa vitesse , les détections avec les briques et la raquettes sont effectuées , la vie est updaté si la balle est perdue , le niveau change si toutes les briques sont détruites , et l'écran de game over apparaît si le joueur perd toutes ses vies .
Les fonctions devant être utilisée a chaque frame sont rassemblé directement dans la fonction draw , qui est appelée a intervalle régulier (toutes les 0.02 secondes , soit 50 fois par seconde)

En plus de ces fonctions , les classes suivantes sont déclarées dans le même fichier :

— Balle : représente la balle . Contient la fonction move() , qui ajoute la vitesse de la balle a sa position . Appelée 50 fois par seconde , cela crée le mouvement de la balle .

— Paddle : représente la raquette

— Brick : représente une brique

4 Quelques points délicats/intéressants

4.1 Points Délicats

4.1.1 Programmer sur IOS

J'ai rencontré beaucoup de difficultés pour programmer sur IOS , étant donnée que les outils de programmations ne sont disponible que sur mac . N'ayant pas eu la possibilités de partir tout le temps programmer dans les salles de TP (par exemple le soir , le week-end end) , j'ai du programmer sur une machine virtuelle installée sur mon ordinateur , devant donc programmer sur une machine virtuelle peu performante . De plus , certaines particularité du langage swift ont été particulièrement déroutante , comme l'obligation de mettre une valeur dans toutes les variables d'une classe héritée d'une autre AVANT d'appeler le constructeur . La différence d'emplacement de certains caractères spéciaux indispensable pour programmer , voir leur absence n'as pas aidé non plus . (par exemple ctrl + shit + l pour afficher un [ )

4.1.2 Gérer la taille de l'écran

Que ce soit pour android ou pour IOS , j'ai basé tout les éléments dessiné du jeu (balle , brique , raquette) en pourcentage de la taille de l'écran . De cette manière , le jeu ressort plutôt correctement peu importe la taille ou le format de l'écran . Cependant , que ce soit sur IOS ou android , j'ai eu des difficultés a mettre ce système en place

— Sur Android : j'ai récupéré la taille de l'écran grâce a la fonction getWidth() et getHeight() . Au début , cela faisait que rien n'apparaissait sur l'écran . Après quelques recherches , je me suis rendu compte que ces fonctions retournent 0 si elles sont appelées avant que le canvas dessine pour la première fois . Or c'est le cas vu que ces valeurs doivent être initialisées pour dessiner les autres éléments. Ce problème a été résolu en faisant dessiner le canvas une première fois , affichant rien , puis appeler la fonction pour initialiser le reste du casse brique .
— Sur Apple : Pour récupérer la taille de l'écran , on utilise la fonction UIScreen.main.bounds.width et UIScreen.main.bounds.width . Le problème rencontré a été que ces valeurs semblent ne pas correspondre avec la vraie taille de l'écran . Par exemple , en faisant 4 briques ayant chacune 1/4 des valeurs ci dessus , les briques dépassent l'écran , des fois sont dessinées entièrement en dehors . Pour régler ce problème , j'ai fais les briques plus petite , et rajouté des espaces entres les briques pour que le décalage ne se remarque pas . La sauvegarde : Du fait de l'architecture de mon jeu , j'ai du trouver un autre système de sauvegarde que celui proposé dans le cour . après plusieurs recherches et essai , j'ai fini par sauvegarder grâce a la sérialisation binaire . J'ai rassemblé toutes les données a sauvegarder dans une classe adaptée , la classe Slot , qui est elle même sauvegardé via une autre classe , qui ne contient qu'un seul attribut : une liste de Slot , afin de pouvoir récupérer directement toutes les sauvegardes d'un coup .

4.2 Points interessants

4.2.1 Moteur physique

étant donné qu'android studio n'as pas de moteur physique , j'ai du créer mon propre moteur moi même . Cela a été un défi , mais ça reste un point intéressant . La détection des collisions n'as pas été la chose la plus compliquée , étant donné que une fonction permettait déjà de le faire a condition d'utiliser les rects , mais le plus compliqué a été de faire réagir la balle aux collisions. Au début j'ai opté pour un système plutôt compliqué pour trouvé l'angle dans lequel se déplace la balle , et de le faire pivoter de 90 degré . Cela n'as pas été concluant . Au final , j'ai opté pour un système différent mais beaucoup plus simple pour chaque type de collision . Pour les bord de l'écran : si c'est les cotés , la vitesse sur X s'inverse juste . Si c'est le haut , la vitesse sur Y s'inverse . Pour les briques : j'ai gardé le système initial . Il fonctionne suffisamment bien pour les briques , il posait surtout problème pour les bords de l'écran .

pour la raquette : J'ai crée un système permettant au joueur d'avoir un certain contrôle sur la balle . Si la balle touche l'extrémité la plus a gauche de la raquette , elle ira a 45 degré sur la gauche . Si elle touche le centre , elle ira a 45 degré sur la droite . si elle touche pile le centre , elle ira en haut en ligne droite . Ensuite , pour chaque autre point de la raquette , elle ira dans un angle proportionnel a la partie de la raquette touché . (par exemple , si elle touche légèrement sur la gauche de la raquette , elle ira légèrement a gauche)

5 Conclusion

Pour Conclure , malgré les difficultés , cela a été une matière intéressante .Cela m'as permis de comprendre les difficultés de sortir des applications sur plusieurs plate formes , car cela nécessite de recréer plusieurs fois le même projet . Cela m'as aussi permis d'appréhender les difficultés occasionnées par la grande variété de taille et format d'écran disponible de nos jours . Mais le plus intéressant a été de comprendre comment programmer un moteur physique basique , le plus dur ayant été de gérer la détection et le comportement des objets en cas de collisions.