UNIVERSITÉ PERPIGNAN VIA DOMITIA

Rapport sur le projet du jeu de huit dames

Projet réalisé par :
- LOUGANI Faouzi, *faouzi.lougani@etudiant.univ-perp.fr*
- GUARDIA Quentin, *quentin.guardia@etudiant.univ-perp.fr*
Étudiants en licence 3 d'informatique à l'UPVD, années 2019-2020.

Sous la direction de M. Naas.

Table des matières

1. Présentation

Dans le cadre de la matière « Interface graphique » enseignée en troisième année de licence informatique à l'UPVD, nous avons eu un projet visant à développer nos compétences en ce qui concerne les interfaces Homme-machine. Nous avons eu le choix entre développer un jeu de 8 dames ou bien un Sudoku. Nous avons opté pour le jeu de 8 dames. Le document présent est le rapport de ce projet et expose les démarches de notre développement. Notre but est donc de développer grâce à Android Studio, une application mobile qui permettra de jouer aux 8 dames. Tout en essayant d'obtenir une interface attrayante.

Mais de quel jeu s'agit-il ? La première étape avant de développer une application est de savoir en quoi elle consiste. Le jeu de 8 dames est un jeu de plateau se jouant seul. Le joueur se trouve initialement face à un échiquier de taille 8x8 vierge de tout pion. Il dispose de huit dames. À terme, il doit placer toutes les dames sans qu'aucune ne se menace selon les règles des échecs. C'est-à-dire qu'aucune ne se retrouve sur la même ligne, colonne ou diagonale. Si l'utilisateur y arrive, la partie est gagnée.

Android Studio sera notre environnement de développement, la version Android Oreo plus précisément. Cet environnement a été conçu par Google pour développer des applications mobiles Android en Java. Il nous permettra d'émuler notre application sur mobile et nous apportera plusieurs fonctionnalités graphiques.

2. Analyse des besoins

Nous avons ensuite réfléchi de quelle manière on pourrait présenter le jeu à l'utilisateur. Sachant qu'un des objectifs est d'avoir une interface bien travaillée. Voici ci-dessous l'application que nous avons initialement imaginé.

Tout d'abord, l'application ne devrait pas s'ouvrir directement sur le plateau. Cela est trop brut. Afin de travailler l'interface, nous avons pensé à mettre en place une page d'accueil. Elle aurait en fond une image en rapport avec le jeu de 8 dames, le logo de l'UPVD ainsi qu'un bouton « Jouer » permettant de démarrer la partie. En essayant de respecter le plus possible l'harmonie des couleurs.

Une fois que l'utilisateur appuie sur le bouton pour démarrer, un échiquier vide ainsi que deux boutons s'affichent. Les deux boutons donnant la possibilité de quitter le jeu ou de recommencer. Une musique de fond rendrait l'application plus vivante. Dès que le joueur clique sur une case, un logo de dame y apparaît avec un son voire une vibration. S'il clique à nouveau dessus, elle disparaît. À partir du moment où il commence à jouer, un chronomètre se lance. Il y a deux issues à la partie :

- Le joueur gagne. Dans quel cas, une boîte de dialogue s'ouvre, en lui spécifiant le temps qu'il a mis avant de gagner. Il a le choix entre rejouer et quitter la partie.
- Le joueur perd. Cela arrive dès que deux dames se situent sur la même ligne, colonne ou diagonale. Une boîte de dialogue s'ouvre, annonçant au joueur sa défaite. Cette fois, elle proposerait : d'annuler le dernier coup, quitter le jeu ou recommencer.

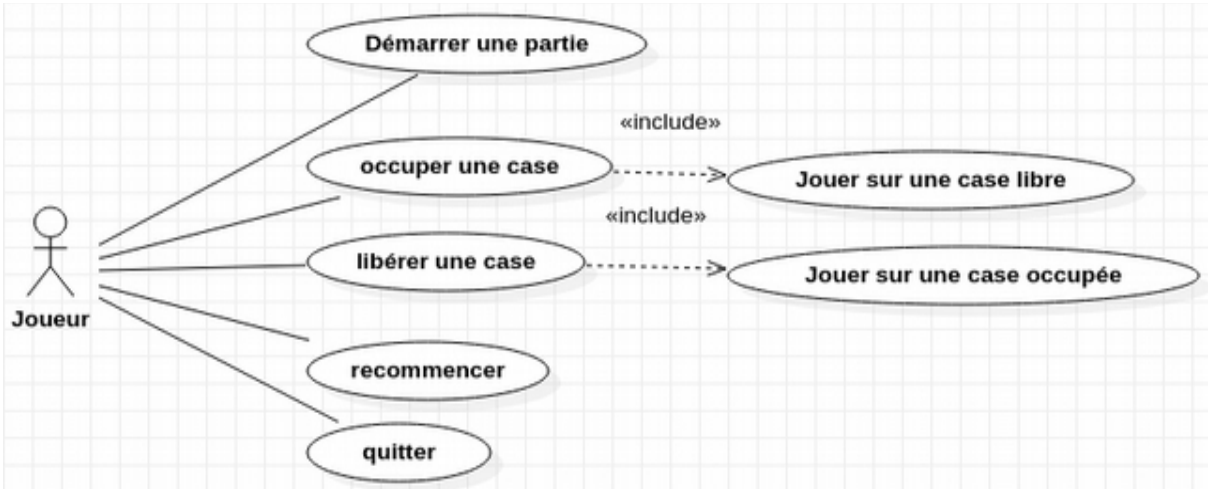
Une partie recommencerait toujours sur un plateau vide. On essayerait de trouver le juste équilibre entre les tailles et les couleurs des divers éléments afin que l'application soit agréable au regard autant que faire se peut. Cette conception étant purement théorique, nous attendrons de voir ce que nous propose Android Studio.

3. Cas d'utilisations

Réfléchissons maintenant un peu plus techniquement. Premièrement, quelles actions le joueur peut faire ? Cela nous permettra d'avoir une idée des fonctions de bases. Voici la liste des actions réalisables que nous avons pensées, et leurs conditions :

- Démarrer une partie
- Occuper une case si elle est libre
- Libérer une case si elle est occupée
- Recommencer
- Quitter la partie

Ce qui donne le diagramme d'utilisation disponible suivant :



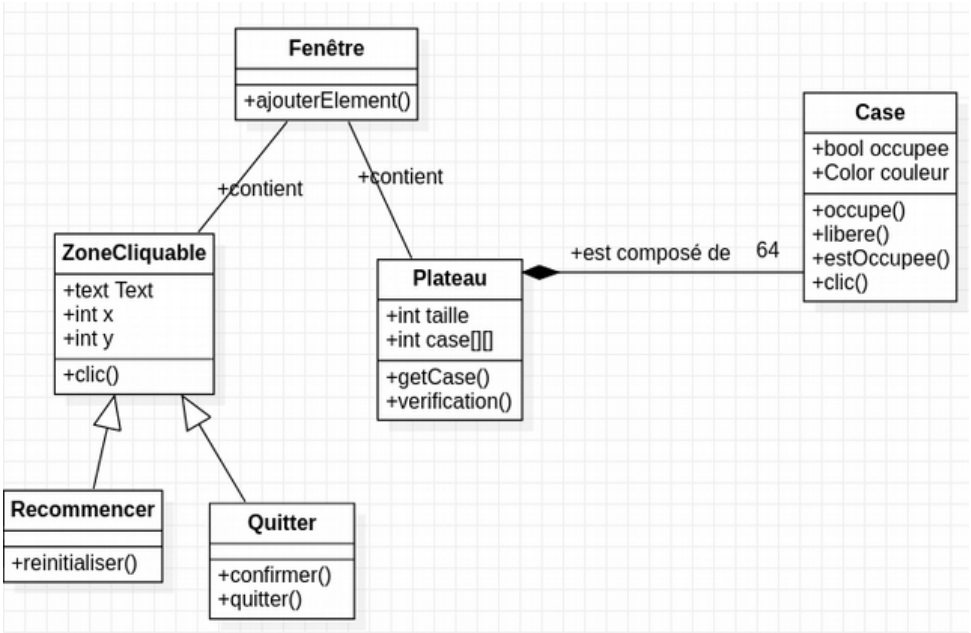
4. Les classes

Voyons maintenant comment ordonnancer la fenêtre et les fonctions selon ce que nous avons déjà. Nous savons déjà que le plateau se compose de 8x8 cases, soit 64 cases. Notre première classe est donc la classe *Plateau*. Cette classe aura pour attribut un tableau de cases et l'entier *taille*, au cas-où l'on souhaite avoir plus de 8x8 cases. Comme la classe a accès à toutes les cases, la première méthode serait *getCase()* pour retourner une case spécifique en fonction de ses coordonnées. De même, grâce à cet accès, une autre méthode sera *verifier()*. Elle vérifier l'état de l'échiquier dans les trois sens : horizontal, vertical et diagonal. Si une rangée contient plus de deux dames, alors la méthode retourne l'information sur la défaite, sinon l'inverse.

Compte tenu de ce que nous venons d'évoquer, il nous faut une classe *Case*. Cette dernière aura pour attribut le booléen *occupee* qui retourne vrai si la case est occupée, faux sinon. Le second attribut est la couleur *couleur* qui déterminera si la case est blanche ou noire.

Il manque deux derniers éléments : la possibilité au joueur de recommencer une partie ou bien de quitter la partie. Pour cela il nous faut créer des boutons. Ou autrement une zone cliquable. La classe *ZoneCliquable* sera alors définie par ses attributs *Texte* pour le texte contenu dans le bouton et les entiers *x* et *y* pour la taille. Les deux classes filles seront *Recommencer* qui appelle la méthode *reinitialiser()* et la classe *Quitter* qui appelle les méthodes *confirmer()* et *quitter()*, qui sont explicitement nommées.

La jonction entre les zones cliquables et le plateau est la classe *Fenêtre*, qui contient tout. In fine, nous obtenons le diagramme de classes suivant :

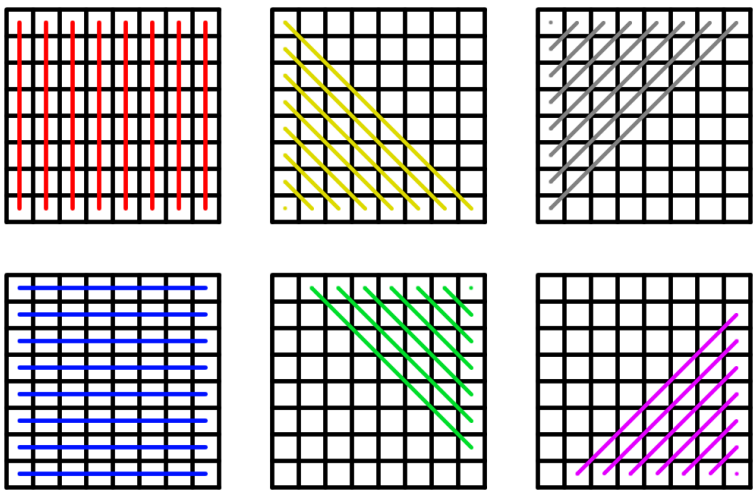


5. Réalisation

Comme nous maîtrisons relativement bien le Java, nous nous sommes permis de d'abord implémenter l'application purement en Java sur Eclipse afin de pouvoir l'exécuter sur l'ordinateur. Ainsi, avant de nous attaquer à Android Studio que nous n'avons jamais réellement utilisé, nous pourrons tester les fonctions importantes et simuler une partie pour nous familiariser avec le rouage des algorithmes nécessaires au 8 dames. Nous nous sommes basés sur les diagrammes précédents sans chercher à travailler l'interface. La partie intéressante de la réalisation concerne l'algorithme de vérification. Il y avait plusieurs manières possibles de faire cette vérification.

Notre choix a été de tester toutes les lignes horizontales, verticales et diagonales montantes et descendantes grâce à des boucles imbriquées. À chaque fois qu'une dame est rencontrée, un compteur spécifique à la rangée s'incrémente. Si un compteur dépasse deux, la fonction retourne vrai. Sinon les compteurs sont remis à zéro à chaque nouvelle rangée, et à la fin *faux* est retourné par défaut. Toute la grille est donc vérifiée à chaque coup.

Une manière de tester tous les alignements possibles se fait grâce à six boucles. Une pour les alignements verticaux, une pour les alignements diagonaux, deux pour les alignements diagonaux du haut à gauche jusqu'en bas à droite et deux pour les alignements diagonaux du haut à droite jusqu'en bas à gauche. Les six boucles sont représentées par des couleurs sur le graphique juste après. Ainsi, on peut initialiser six compteurs à zéro au début de la fonction *verification()*. Comme il s'agit d'une grille de taille huit par huit, la boucle principale tournerait huit fois, de i allant de 0 à 8 exclus et servirait pour toutes les vérifications.



Les vérifications horizontales et verticales se font dans une boucle imbriquée allant jusqu'à 8. Les deux vérifications diagonales allant jusqu'à la diagonale du milieu (en jaune et en gris) sont comprises dans une boucle imbriquée allant de 0 à i inclus. Et enfin les deux vérifications diagonales s'arrêtant avant la diagonale principale (vert et violet) se font dans une boucle allant de 0 à i exclus.

Pour chaque direction d'alignement, on incrémente le compteur associée si une case occupée est rencontrée. À la fin des boucles imbriquées, mais toujours dans la boucle principale, on vérifie qu'aucune valeur des six compteurs ne dépasse 1. Si c'est le cas, c'est que deux dames sont alignées et on retourne *true*. Sinon on retourne *false* tout à la fin. Voici la fonction en java :

```
public static boolean verification() {

		//Variables
		int i,j;
		int[] res= {0,0,0,0,0,0}; //Les compteurs

		for(i=0;i<8;i++) {

			//On initialise les compteurs à 0
			for(j=0;j<6;j++)
				res[j]=0;

			//Tests horizontal et vertical
			for(j=0;j<8;j++) {
				if(plateau[i][j].isOccupee())
					res[0]++;
				if(plateau[j][i].isOccupee())
					res[1]++;
			}

			//Premiers tests diagonaux
			for(j=0;j<=i;j++) {
				if(plateau[i-j][j].isOccupee())
					res[2]++;
				if(plateau[7-i+j][j].isOccupee())
					res[3]++;


			}

			//Seconds tests diagonaux
			for(j=0;j<i;j++) {
				if(plateau[8-i+j][8-j-1].isOccupee())
					res[4]++;
				if(plateau[j][8-i+j].isOccupee())
					res[5]++;
			}

			//Vérification des compteurs
			for(j=0;j<6;j++)
				if(res[j]>1) return true;

		}
		return false;
	}
```

Nous avons fait les tests unitaires à partir de cette fonction sur Eclipse, avec une interface graphique également.

Bien que nous ayons pu tout implémenter en Java, le code n'avait jusqu'à présent que de valide le traitement du jeu. Maintenant, il faut tout refaire sur Android Studio pour avoir une interface plus esthétique, comment définit précédemment, et disponible sur téléphone portable.

6. Usage de l'environnement Android Studio

Android Studio nous offre plusieurs outils afin d'intégrer notre projet. Nous les présentons dans cette partie.

6.1 AndroidManifest.xml

Ce fichier de paramétrage est obligatoire dans tous les projets. Il se trouve dans *app/src/main*. Ce dernier sert à paramétrer le projet en indiquant son nom, les activités présentes, les packages, l'icône du projet, la version d'Android et ainsi de suite. Par exemple, on peut voir dans le nôtre qu'il y a deux activités : *MainActivity* et *Main2Activity*. Nous expliciterons plus tard de quoi il s'agit.

6.2 Les vues

La vue comme on l'a déjà étudié en cours, est le bloc de construction de base de l'interface utilisateur dans Android. Donc elle peut être considérée comme un rectangle à l'écran qui montre un certain type de contenu. Il peut s'agir d'une image, d'un texte, d'un bouton ou de tout ce qu'une application Android peut afficher.

Du côté programmatique, nous avons créé nos vues dans les fichiers sources en Java, comme le *MainActivity.java* qui correspond à une activité. Voici une capture d'écran de notre code, qui illustre l'ajout d'un bouton à notre vue :

```
public void event_quitter(View view){


    Intent homeIntent = new Intent(Intent.ACTION_MAIN);
    homeIntent.addCategory( Intent.CATEGORY_HOME );
    homeIntent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    startActivity(homeIntent);

}
```

Il faut inclure la bibliothèque responsable des vues avec cette ligne en début de fichier :

```
import android.view.View;
```

Du côté de l'approche déclarative, nous définissons les Views directement dans les fichiers XML de conception associés au fichier sources que nous développons. Par exemple, on les défini dans *activity_main2.xml* pour *Main2Activity.java*, la deuxième activité. Voici un exemple d'information contenue dans le fichier XML :

```
<Button
    android:layout_width="200dp"
    android:layout_height="55dp"
    android:onClick="event_quitter"
    android:text="Quitter jeu" />
```

Les fichiers XML que nous créons se trouvent dans le dossier *./app/src/main/res/layout* alors que les fichiers Java se trouvent dans *./app/src/main/java/com/example/myapplication*. À l'exception du fichier *AndroidManifest.xml*.

On peut paramétrer dans ces fichiers des attributs. Voici ce qu'indique la capture d'écran ci-dessus. Les paramètres *android:layout_width* et *android:layout_height* nous permettent de définir la largeur et la hauteur du rectangle invisible créé par la vue. Et *android:text* ajoute le texte à l'intérieur de notre type vue, ici un bouton. Le plus important c'est *android:onClick* qui donne l'événement à réaliser une fois que l'on a cliqué sur le bouton. Cela connecte la vue *View* à la fonction *event_quitter(View view)* afin d'exécuter son contenu.

6.3 Autres bibliothèques utilisées

Nous avons fait appel à plusieurs bibliothèques pour leurs fonctionnalités.

Dans le groupe de bibliothèques *android.graphics*, nous avons utilisé *Point* et *Color*. Voici comment importer par exemple *android.graphics.Point* :

```
import android.graphics.Point;
```

Il faut le placer dans l'entête. Cela permet dans notre projet de créer le composant graphique *point* que nous avons utilisé dans la fonction *creer_grille()* pour paramétrer les coordonnées de chacune des cases. Ensuite,

```
import android.os.SystemClock;
import android.widget.Chronometer;
```

Nous ont servi à initialiser et lancer le chronomètre qui indique en combien de temps le joueur à fini la partie. Aussi,

```
import android.media.MediaPlayer;
```

Nous a été utile pour implémenter la musique de fond une fois l'application lancée. Plusieurs autres bibliothèques ont été importées de la sorte pour mieux intégrer le jeu de huit dames et personnaliser l'interface. Toutes les importations sont disponibles sous la ligne réservée au package dans chaque fichier.

6.4 Les fichiers XML: le cas de activity_main2.xml

On s'attarde en particulier sur ce fichier car c'est celui qui concerne le jeu de huit dames. On peut voir dans ce fichier que tout est placé dans une balise *RelativeLayout*. Cette balise est ce qu'on appelle un *ViewGroup*, car elle contient et affiche des *Views* enfants, qui sont des éléments en position relative à l'intérieur du *ViewGroup*.

Voici l'entête :

```
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".Main2Activity"
```

Le lien indique que l'on utilise les éléments XML par défaut. Les *layouts* indiquent que l'on prend la dimension maximale et le contexte que le fichier est associé à *Main2Activity*.

Les balises *LinearLayout* organisent les deux views des boutons permettant de rejouer et quitter, pour qu'ils soient alignés horizontalement et verticalement. *GridLayout* place les enfants dans une grille rectangulaire. Il y a aussi la balise réservée pour le chronomètre et celle pour le bouton qui réinitialise le compteur. Dans chacune de ces balises, on peut observer des attributs importants comme :

```
android:layout_height="wrap_content"
android:layout_width="match_parent"
```

Qui définissent les propriétés de chaque view.

6.5 Emplacement des images et médias

On a placé les images dont nous avions besoin dans *./res/drawable* et la musique dans *./res/raw*.

7. Intégration

Nous voici enfin face à Android Studio. Tout démarre sur *Main Activity* qui est vide pour l'instant. Nous créons alors de nouveaux fichiers pour déclarer les classes nécessaires.

La première chose à faire a été de créer la classe *Case*. Sans nous soucier pour l'instant du graphique. Une case est définie par sa ligne et par sa colonne se sont ses attributs.

La classe *Jeu* a également état défini. Cette dernière contient le plateau de dames *[][]positions_reines*. Chaque case du tableau contient 0 ou 1 selon si la case est occupée. Donc initialement, toutes les cases sont définies à 0. La taille de la grille dépend de l'entrée, 8x8 par défaut. Il y a également une variable réservée pour le nombre de dames à jouer.

Toujours dans la même classe, nous avons implémenté la fonction *verification()*, que nous avions déjà développée en Java et que nous avons adapté à notre nouveau code. Elle sera toujours appelée par la fonction *affecter_reine()* qui prend en paramètre des abscisse et ordonnée et qui remplace la valeur de la case par son opposé, 0 ou 1. Tout en vérifiant, si cela n'entraîne pas la défaite du joueur. Si le joueur ne perd pas, la partie continue, sinon on affecte 1 à *a*, ce qui ouvre une boîte de dialogue, comme expliqué plus tard.

La fonction *a_gagner()* permet de déterminer si le joueur a gagné en comptant le nombre de dames posées durant la partie en cours. En effet, si le joueur n'a pas perdu et que 8 dames sont posées, alors il a gagné. Nous pouvons aussi noter la présence de la fonction *position_actuelle()* qui retourne la case en fonction des coordonnées entrées en paramètre et les fonctions *get_a()* et *set_a()* qui modifient la variable *a*, qui vaut 0 ou 1. C'est grâce à cette variable qu'une boîte de dialogue s'ouvrira en cas de défaite. En effet, la variable sera récupérée à chaque coup dans le *MainActivity*, où l'on gérera les boîtes graphiques. Nous verrons cela plus tard dans le détail.

Suite à toute cette théorie, il est temps de s'intéresser à l'interface de l'application. Retour dans la classe *Case*. Nous avons mis des couleurs, qui ont pour code #ffffff (blanc) et #00ffff (turquoise) plutôt que du noir et du blanc pour le damier. Cela est plus léger pour l'œil. Nous avons ajouté la fonction *dessiner_reine()* qui dessine une dame si l'entier entré en paramètre vaut 1, et l'enlève sinon. Cela grâce à la méthode *setImageResource()*, qui prend en paramètre l'objet *R.drawable.reine* qui correspond à l'image de la reine ; ou le chiffre 0 qui correspond à une case vierge.

Maintenant, penchons-nous sur les deux activités. C'est ici que seront gérés les événements, *events* en anglais, par l'intermédiaire des *ActionListener*. Ces derniers permettront à l'application de détecter le contact du joueur sur les diverses zones du graphique. Typiquement utilisés pour appuyer sur une case ou un bouton.

Nous avons dû créer deux activités principales, et donc deux *MainActivity*. La première correspond à la page d'accueil. Celle-ci est composée d'une image de fond personnalisée et d'un bouton *Jouer* afin que l'arrivée sur le plateau ne soit pas trop brutale. L'image de fond correspond à un jeu d'échec avec le logo de l'UPVD en haut à droite. Le bouton amenant au jeu est un *ActionListener*, qui exécute la fonction *openActivity()*, qui elle créer un nouvel *Intent*, *Main2Activity*, puis le lance grâce à *startActivity()*. Concrètement, cela ouvre la deuxième activité principale, qui contient les éléments définis ci-dessous.

Comme la partie est lancée par la méthode *onCreate()*, nous en avons profité afin d'ajouter un *MediaPlayer* qui met une musique de fond. De même, nous avons ajouté le widget *simpleChronometer*, qui se lance grâce à *start()* sur le *onCreate()*.

La fonction *creer_grille()* affiche le plateau graphiquement, en le divisant en cases, en fonction de la taille de l'écran. C'est ici que les objets *Case* sont créés. Nous utilisons des *layouts*, dont nous reparlerons plus tard, ainsi que les *ActionListener* pour que chaque case soit une zone cliquable. Puis vient la fonction *dessiner_grille()* qui affiche en réalité les dames sur le plateau. Pour cela, nous faisons appel à la fonction *dessiner_reine()*. Puis la fonction récupère le *a* dont nous avons précédemment parlé. Si *a* vaut 1, une boîte de dialogue s'ouvre, annonçant la défaite. Il y a alors deux boutons possible, rejouer appelant *raffrichir_jeu()* et *simpleChronometer.start()* ou quitter appelant *System.exit(0)*.

Il y a aussi la fonction *interface_affecter_reine()*. Dans un premier temps, cette fonction appelle *affecter_reine()* afin d'ajouter ou enlever la dame de la case concernée. Dans un second temps, elle appelle la fonction *a_gagner()* pour voir si le joueur a gagné. Si c'est le cas, la fonction *interface_agagner()* est appelée. Elle permet d'afficher une boîte de dialogue indiquant au joueur qu'il a gagné. Le chronomètre s'arrête dans une même temps, grâce à la méthode *stop()*. Le joueur a alors le même choix que pour la boîte de dialogue précédemment décrite.

Les fenêtres de dialogue sont créées avec *AlertDialog.Builder()*. On peut leur affecter un titre, un message ainsi que des boutons, comme *rejouer* ou *quitter* auxquels on associe un *ActionListener*. Les paramètres de mise en page, de *layout* en anglais sont définis dans *activity_main.xml*. Pour rappel, tout est défini en XML dans ce fichier : le nombre de cellules dans la grille de jeu (les cases), la taille et le texte des boutons, etc.

Enfin, le dernier détail améliorable à notre portée était le logo de l'application, qui était jusqu'à présent celui fourni par défaut. Android Studio propose l'option *Image Asset*, qui permet de choisir et configurer une image afin qu'elle apparaisse en tant qu'icône de l'application. Nous avons donc pu personnalisé le logo avec une couronne, symbolisant la dame. Nous avons décidé de garder le titre « My Application » pour prouver que nous avons tout fait de A à Z, à partir de l'accueil par défaut d'Android Studio.

8. Structuration du projet

Le fichier Java *MainActivity* lance *Main2Activity*, qui lui se base sur les classes *Jeu* et *Case*. Les deux activités sont affichées sur l'interface. Les propriétés d'affichage sont données par les attributs situés dans *activity_main.xml* et *activity_main2.xml* respectivement. Le diagramme liant classes, activités et fichiers XML est disponible en annexe.

Le diagramme de séquence est également disponible en annexe.

9. Bilan

À chaque fois que nous avons ajouté un nouvel élément au code, nous compilions juste après. Après avoir tout intégré et quelques ajustement, notre interface fonctionnait enfin.

Nous sommes satisfaits du rendu. Bien qu'Android Studio soit un outil complexe au premier abord, il est de pair, complet. Ce qui nous a permis d'intégrer plusieurs fonctionnalités à cette application toute simple. Un de nos plus gros problèmes a été de créer deux activités principales : celle d'accueil et celle contenant le plateau. Ce problème a été résolu. Cependant, nous avons eu un autre problème, notre chronomètre ne peut pas se réinitialiser seul lorsqu'une partie recommence. D'où le bouton, afin qu'il y ait un *event* le réinitialisant. On a préféré ne pas perdre trop de temps dessus et améliorer le reste. En annexes, sont disponibles des captures d'écran présentant le projet final.

10. Sources

Dans le rapport :

- Diagrammes de classe et de cas d'utilisations du programmes Java : captures d'écran personnelles sur le logiciel StarUML.

- Images du code : captures d'écran personnelles et réalisées sur Android Studio.

- Schémas du plateau : personnels.

Dans l'application :

- Musique : https://soundcloud.com/ikson/a-while-free-download

- Logo de la page d'accueil : https://image.freepik.com/vecteurs-libre/icone-couronne-qualite-premium_53876-63008.jpg

- Fond de la page d'accueil : https://fsb.zobj.net/

- Image de la reine : http://lubeck.fr/les-8-reines/reine.png

- Logo de l'UPVD : https://upload.wikimedia.org/wikipedia/fr/thumb/e/e6/UPVD_logo.svg/1024px-UPVD_logo.svg.png

11. Annexes

Ci-dessous, captures d'écran de l'application réalisé grâce à l'émulation d'Android Studio. De gauche à droite : écran d'accueil, partie en cours, partie terminée.

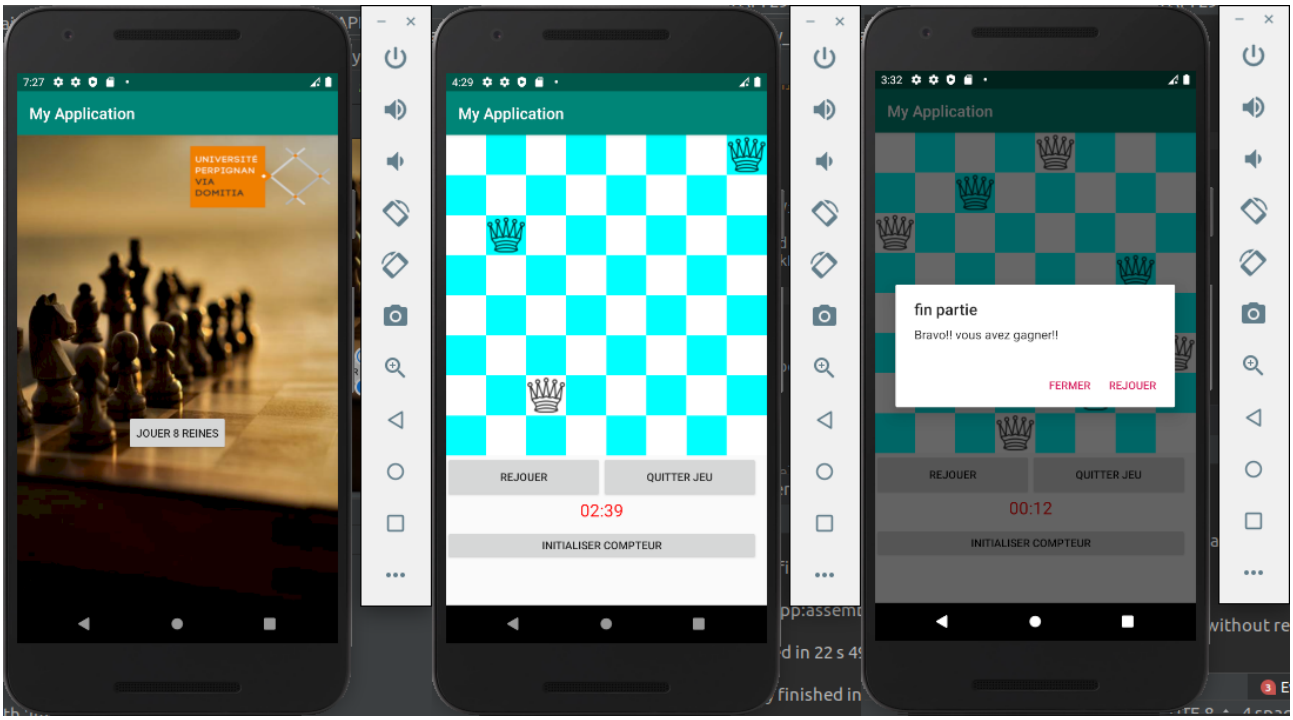


Diagramme regroupant activités, classes, interface et fichiers XML, capture d'écran de StarUML :

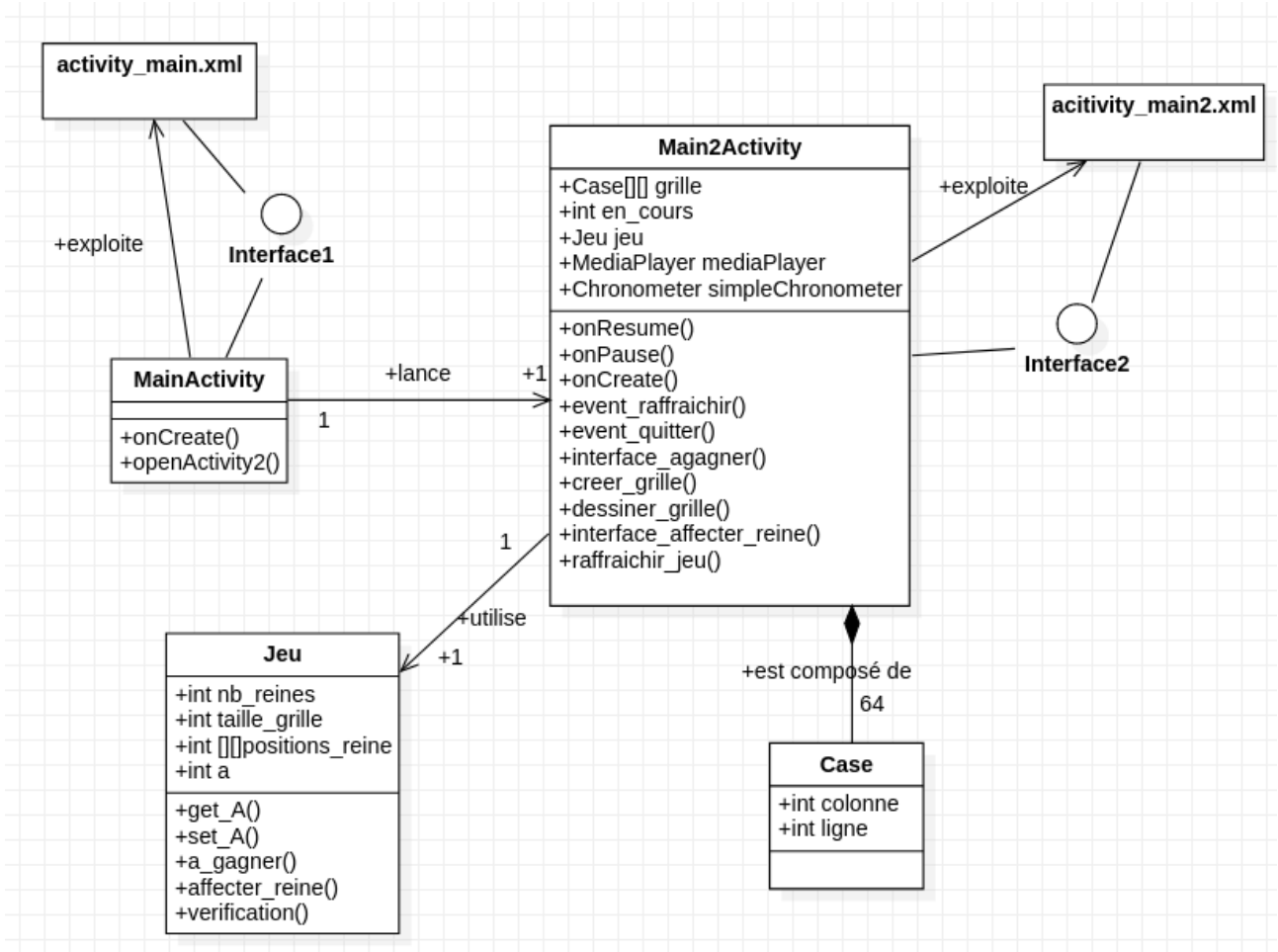


Diagramme de séquence, capture d'écran du logiciel Umbrello :