SHOOT ‘EM-DROID

Matteo D’Eramo

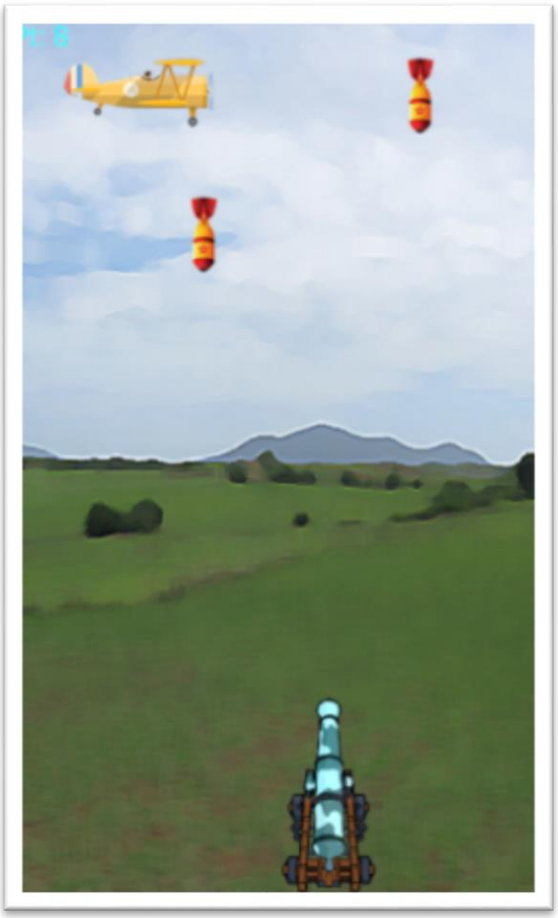
Matricola:290480

INDICE

1.DESCRIZIONE DEL PROBLEMA

L'obiettivo di questa applicazione è quello di sviluppare un videogioco *shoot 'em up* attraverso *android studio* in cui si potrà controllare un cannone che dovrà sparare a degli aeroplani, incrementando il punteggio ad ogni aereo abbattuto e cercando, nel mentre, di evitare le bombe cadenti che se colpiranno il cannone porteranno alla conclusione della partita.

L'aumentare del punteggio comporterà sia un incremento della difficoltà nel colpire gli aeroplani sia una maggior frequenza di bombe cadenti.



Il punteggio sarà tenuto da un indicatore posto in alto a sinistra e verrà salvato in una *leaderboard,* nel caso di punteggio da podio.

La partita potrà essere avviata dal menu principale, il quale contiene anche un pulsante per la *leaderboard* , per il *setting* del cannone e per chiudere l'app.

Il cannone potrà essere spostato a destra o sinistra trascinando il cursore verso una direzione e potrà sparare con un tocco singolo su di esso.

(*piccolo screenshot del videogioco)*

2. SPECIFICA DEI REQUISITI

Le specifiche di progetto possono essere riassunte in 3 macro-categorie:

- *Specifiche funzionali*, ovvero cosa il giocatore potrà fare effettivamente all'interno del gioco
- *Specifiche estetiche e sonoro*, ovvero tutto ciò che riguarda la parte grafica e sonora del videogioco
- *Prestazioni*, ovvero le specifiche riguardanti caricamento e fluidità complessiva.

*(con * sono indicate le specifiche facoltative)*

FUNZIONALITA'

- Accesso alla leaderboard dei punteggi, in cui verranno salvati i punteggi migliori con relativa data.
- Possibilità di modificare il cannone da una lista di stili predefiniti accessibile dal menu principale
- Una volta terminata la partita si può iniziarne un'altra o tornare al menu principale

ESTETICA E SONORO

- Ogni aereo colpito comporta un'esplosione con relativo rumore
- Le bombe cadute al suolo generano un'esplosione
- Due tipologie di aerei (*)
- Più tipologie di bombe, le quali hanno velocità di caduta differente (*)
- Effetti post sparo per il cannone (*)
- Musica di sottofondo per il menu principale e durante la partita
- Background modificabile dal codice

PRESTAZIONI

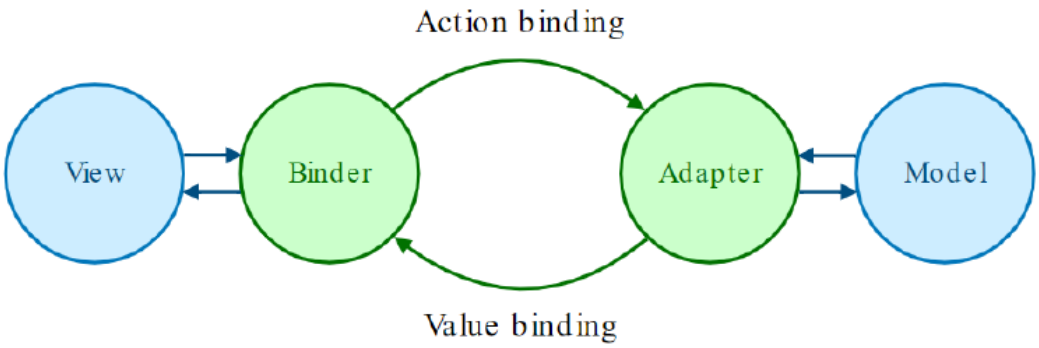
- Fluidità nei movimenti del cannone e nella risposta dell'applicazione al tocco
- Fluidità nelle collisioni
- Caricamenti senza rallentamenti
- Leggerezza in termini di memoria occupata dall'applicazione (*)
- Compatibilità con smartphone android qualsiasi (*)

3. PROGETTO DI SISTEMA

Ci addentriamo nella parte tecnica del documento, cercando di descrivere sinteticamente l'architettura del progetto, illustrandone prima l'architettura software (top) per poi scendere nel dettaglio dei blocchi funzionali (down) che la compongono.

3.1 ARCHITETTURA SOFTWARE

Per la realizzazione di questo progetto è stato seguito un design pattern OOP (*object oriented programming*), con modello architetturale MV (*Model, View*); seppur quest'ultimo sia un metodo non convenzionale, ha permesso di diminuire il carico di lavoro in momenti di *refactoring* e durante la gestione delle animazioni, questo perché si ha a che fare con soli due packages.



(fonte: https://www.cocoawithlove.com/blog/mvc-without-the-c.html)

- **Model**: è costituito da tutte le classi atte alla rappresentazione dello stato dell'applicazione e della logica, mantenendo informazioni di tutto ciò che l'utente visualizza a video; tutte le classi che estendono *Activity* si trovano in questa sezione.
- **View**: si occupa di rappresentare i dati all'utente raccogliendone l'input che, attraverso un'apposita interfaccia, permetterà di cambiare lo stato del *Model* che a sua volta risponderà al *View* per determinare cosa deve o non deve essere visualizzato.

La logica del controller, che ha il compito di ricevere le informazioni in input date dall'utente e di gestire quelle in output, è spalmata all'interno dei macro-moduli suddetti.

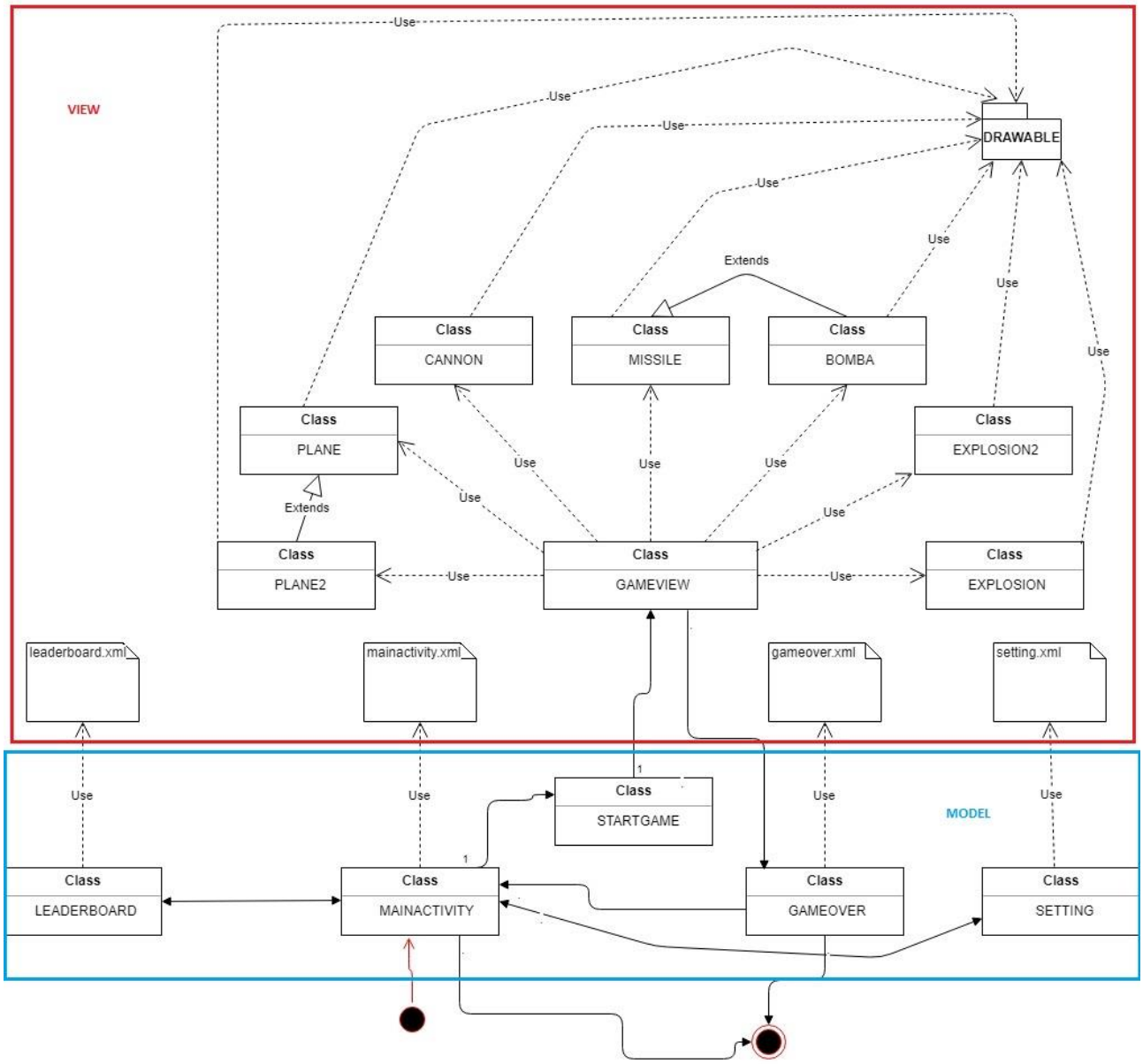


Figura 1: architettura di sistema

Nell'immagine precedente si può notare la struttura macroscopica del progetto; a partire da **MainActivity**, la classe che imposta la schermata iniziale con il metodo *setContentView()*, si può iniziare a giocare attraverso il metodo *startGame* che crea un nuovo *Intent* (ovvero viene "aperta una nuova pagina") per la classe **Startgame**, in cui il metodo *onCreate* genera l'ambiente di gioco effettivo utilizzando un oggetto *Gameview* e uno di tipo *mediaplayer* (per l'aspetto sonoro del livello).

In **Gameview** la partita si conclude con la chiusura dell'attività corrente e l'apertura di un'altra attraverso un nuovo *Intent* riferito alla classe **Gameover**, che imposta la schermata xml omonima e dà la possibilità di uscire o effettuare un'altra partita.

Nel grafico le frecce non tratteggiate rappresentano i possibili passaggi da una classe all'altra attraverso la seguente procedura:

- Creazione di una nuova attività relativa alla classe puntata dalla freccia
- Chiusura dell'attività da cui parte la freccia

La presenza di frecce bilaterali indica che le attività sono una raggiungibile dall'altra e viceversa. Le frecce tratteggiate rappresentano invece un utilizzo di una classe o elemento di una cartella.

NOTA

Alcuni metodi riguardanti la parte logica come la gestione delle collisioni sono stati invece gestiti nel View; Consapevole dell'incoerenza con il MV che avrebbe assegnato determinati metodi al Model, giustifico questa inesattezza con la decisione presa di voler focalizzare l'attenzione sull'approccio OOP, il quale da una parte ha facilitato notevolmente la scrittura del codice e la risoluzione di errori o bug dato che fornisce una visione d'insieme abbastanza chiara del progetto, anche se dall'altra comporta una minor estendibilità dell'applicazione per eventuali altri livelli.

3.2 DESCRIZIONE DEI MODULI

MODEL

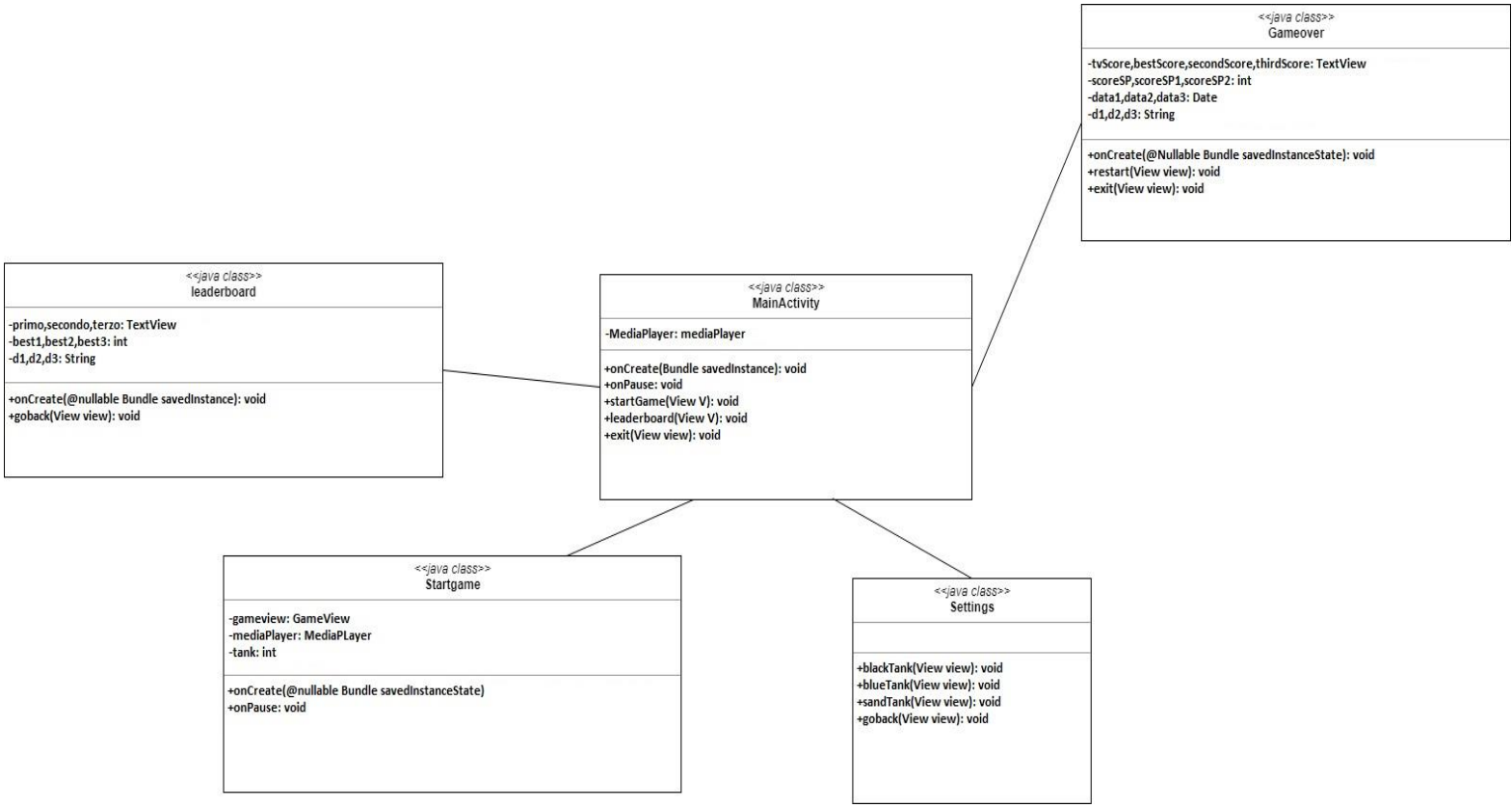


Figura 2: Model

NOTA: (i layout verranno esposti in questa sezione per facilitare la spiegazione dei metodi delle classi del Model)

MainActivity

Questa classe permette di avviare l'applicazione; il metodo principale è *onCreate* (presente in tutte le classi del *Model*) che permette di impostare il layout e di generare la musica di sottofondo attraverso un oggetto *mediaplayer* il quale utilizza un file mp3 dalla cartella *raw*. Gli altri metodi presenti sono quelli che permettono il passaggio ad una nuova activity:il metodo *startgame* permette di avviare il gioco ed è associato ad un metodo *onclick* relativo ad un'imageView presente nel codice xml della schermata principale; *leaderboard* e *setting* consentono di accedere alle schermate xml omonime (anche questi sono associati a due imageView della schermata principale tramite *onClick*) e infine il metodo *exit* interrompe l'attività corrente e chiude l'applicazione.

Leaderboard

Questa classe gestisce l'Activity della *leaderboard*; i valori da immettere sono quelli relativi a 6 variabili statiche presenti nella classe Gameover.

Figura 3: leaderboard.xml

Il meccanismo della *leaderboard* funziona in questo modo:

- Nella classe **Gameover** sono istanziate tre variabili statiche int e tre variabili statiche *String*. Vengono aggiornate nel caso in cui il valore dello *score* ottenuto durante la partita deve essere inserito in una certa posizione del podio
- Essendo le variabili statiche, nella classe **Leaderboard** è sufficiente richiamare le variabili e modificare se necessario il valore corrente di 3 variabili *int* istanziate nella classe e 3 variabili *String* contenenti le date in cui avviene una modifica del podio corrente (i 6 valori verranno poi impostati in 3 *Textview* di *leaderboard.xml*). Oltre al metodo *onCreate* è presente un altro metodo per tornare al menu principale.

Gameover

Sharepreferences è un'interfaccia che consente di memorizzare delle istanze e condividerle tra le classi; queste "preferenze" possono essere modificate tramite un oggetto *editor*.

Figura 4: gameover.xml

Questa proprietà è stata sfruttata per tenere traccia dei punteggi migliori ottenuti giocando e le relative date; come già accennato, 3 variabili *int* e 3 variabili *string* vengono impostate come preferenze inizialmente con valori 0 e " " e nel caso in cui un punteggio sia migliore di una delle preferenze, esso viene salvato nella posizione giusta insieme alla data, il quale è una conversione in tipo *string* di una variabile *date* equivalente ad un istanza della classe **Calendar** (Calendar.getInstance().getTime()). Nel layout, a fianco dell'Imageview centrale è riportata una versione semplificata del podio.

Settings

Questa classe consente di modificare lo stile del cannone; contiene un *radiogroup* di *radiobutton* che se selezionati impostano una variabile statica ad un valore compreso tra 0 e 2.

Figura 5: settings.xml

Questo variabile viene utilizzata all'interno della classe **Cannon** e a seconda del suo valore viene impostata un *bitmap* differente.


color: Black


color: Mimetic sand


color: Mimetic snow

Startgame

La classe è responsabile della creazione del livello attraverso *setContentView(gameview)* chiamato all'interno di *onCreate;* è presente anche un metodo *Pause* che interrompe la musica di sottofondo quando c'è un cambio di *Activity*.

VIEW

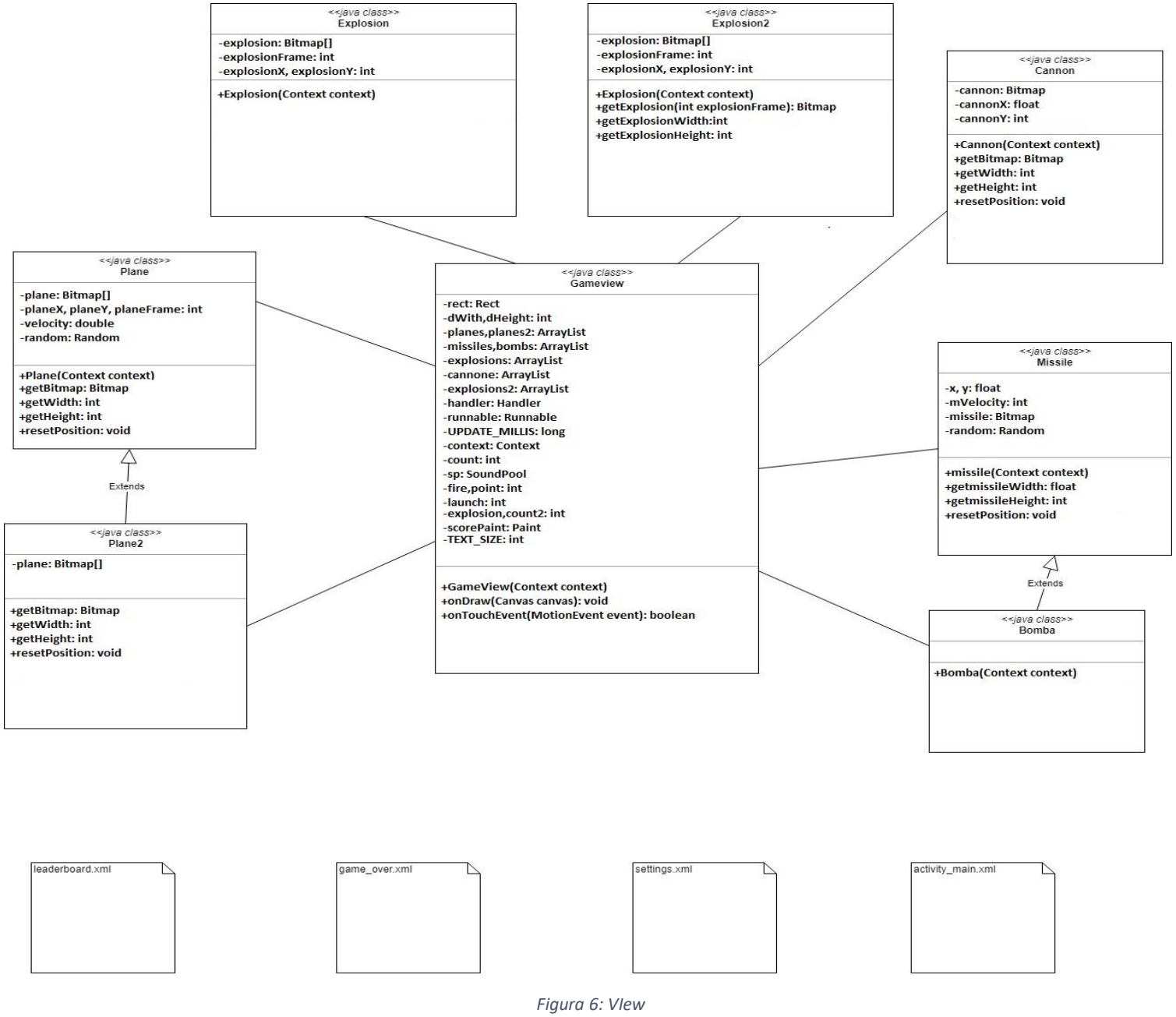


Figura 6: VIew

Gameview

Questa classe costituisce il motore grafico dell'applicazione; è qui che vengono gestite tutte le dinamiche di gameplay come collisioni, touch events ecc. e ci sono 3 metodi principali:

- *Gameview (costruttore):* il costruttore permette di definire gli attributi e gli elementi fondanti del videogame; è qui che vengono inizializzati tutti gli oggetti (bombe, aerei, cannone ecc..) che saranno presenti all'avvio del gioco e che andranno ad essere disegnati tramite il metodo *onDraw*
- *OnDraw:* con questo metodo vengono effettivamente "disegnati" gli oggetti e il background di gioco presi dalla cartella *drawable* attraverso *BitmapFactory.decodeResource()*; per oggetti come gli aerei o esplosioni che necessitano uno scorrimento di immagini vengono utilizzati dei cicli for, i quali sono utilizzati anche per il movimento effettivo degli oggetti nel *canvas* (viene attuato un aggiornamento delle coordinate degli oggetti istante per istante). Il sistema di collisioni è molto semplice: infatti un oggetto-proiettile e un oggetto-aereo collidono solo se si sovrappongono le rispettive coordinate, opportunamente inizializzate nelle rispettive classi. Viene aggiunto un oggetto-esplosione ad ogni collisione nella posizione in cui l'aereo è stato colpito e ad ogni oggetto-bomba caduta al suolo. Si crea un altro oggetto bomba dopo il raggiungimento di determinati punteggi e utilizzando *if-statements* e se una bomba colpisce il cannone, viene aperta l'activity **Gameover** e chiusa l'*Activity* corrente.
- *onTouchEvent:* è il metodo con cui viene gestito il movimento effettivo del cannone; si distingue un caso *ACTION DOWN* (singolo tocco sullo schermo) che permette di sparare (creare un oggetto-missile) toccando l'immagine del cannone e un caso *ACTION MOVE* (trascinamento) con cui si può far scorrere il cannone orizzontalmente il quale seguirà la posizione del cursore/dito.

Plane & Plane2

Le due classi presentano due metodi *getWidth* e *getHeight* per larghezza/altezza delle immagini degli aerei (inserite in array di *bitmap*) e un metodo *resetPosition* che viene opportunamente invocato in **Gameview** quando l'aereo arriva all'estremità dello schermo e anche all'interno del costruttore per inizializzare la posizione dell'oggetto; in **Plane** l'oggetto parte dalla destra dello schermo mentre in **Plane2** da sinistra. L'attributo *velocity* è un valore che permette di incrementare la coordinata x dell'aereo in *onDraw* e ovviamente un valore alto comporta anche una velocità alta dell'aereo.

Missile & Bomba

Queste classi hanno caratteristiche simili a **Plane/Plane2** (i metodi *getWidth, getHeight, resetPosition*) ma hanno un movimento verticale nello schermo; inoltre, **Missile** rappresenta gli oggetti-missile del cannone e per questo ha coordinate iniziali pari alla posizione della bocca del cannone, mentre **Bomba** rappresenta le bombe cadenti dall'alto e ha condizioni iniziali randomiche.

Cannon

Questa classe presenta metodi analoghi ai metodi già trattati nelle classi descritte in precedenza, con l'unica eccezione riguardante la *Bitmap* utilizzata: a seconda del valore di Startgame.tank si utilizzerà un diverso stile del cannone, che può essere *black, mimetic sand* e *mimetic snow*.

3.3 PROBLEMI RISCONTRATI

Problemi di esportazione

Un'anomalia riscontrata riguarda il funzionamento della *leaderboard* su smartphone fisico; mentre nell'*AVD* non causa alcun problema, nella versione installata su *Huawei p10 lite* si sono verificati alcuni problemi nel salvare il punteggio correttamente, dato che effettuando la prima partita il punteggio salvato (che sarà l'unico presente e quindi anche quello massimo) viene salvato nella prima e nella terza posizione del podio. Il problema risiedeva in una cattiva formulazione del codice ed è stata prontamente risolta.

Un altro problema riscontrato su *Huawei* riguarda la fluidità delle collisioni che è minore rispetto alla versione testata sull'emulatore *AVD*, a causa molto probabilmente della minor potenza del dispositivo rispetto al portatile su cui è stato creato il progetto (Acer Extensa 2520 con processore Intel(R) Core(TM) i5-6200U da 2.30GHz).

Problemi di codice

Uno degli aspetti che ha comportato un maggiore studio è stata la *leaderboard*; facendo alcune ricerche ho potuto constatare che esistono delle API di Google play games (queste) che permettono di implementare la *leaderboard* online ma, non ritenendo l'utilizzo di API inerente all'esame, ho preferito optare per un approccio diverso anche se troverei interessante poter approfondire questo aspetto in futuro.

4. ESTENDIBILITA’ E PERSONALIZZAZIONE

Ci sono alcune funzionalità che avrebbe ulteriormente migliorato l’esperienza di gioco ma che, per mancanza di tempo, non è stato possibile inserire:

- Era stato pensato un effetto post-sparo con del fumo uscente dalla canna del cannone, ma la gestione dell’animazione del fumo sarebbe stata troppo complessa
- Un livello ambientato nei mari con le navi al posto degli aerei era un’idea azzeccata, ma non eseguita per mancanza in primis di elementi grafici adatti
- La modifica delle condizioni climatiche, random o selezionabili dal menu principale, poteva essere un ulteriore arricchimento grafico per il videogame.

5. BIBLIOGRAFIA

- https://stackoverflow.com/
- https://it.wikipedia.org/wiki/Shoot_%27em_up
- https://developer.android.com/studio
- https://www.geeksforgeeks.org/
- https://www.youtube.com/watch?v=yqljFYoZrQI&list=PL53F6DF40FBC249BA