

# Progetto Android Studio - Road To Amsterdam

## Sommario

Progetto Android Studio - Road To Amsterdam.....	1
Introduzione .....	3
Android .....	3
Introduzione agli APK .....	4
Introduzione Android Studio .....	5
Descrizione Preliminare Progetto.....	5
Progetto .....	6
Caratteristiche Videogioco .....	6
Strumenti Utilizzati .....	7
Android Studio.....	7
Android Profiler .....	8
CVS.....	9
GitHub.....	10
Photoshop.....	11
WanderShare Filmora.....	11
codebeautify.org .....	11
Funzionamento Videogioco .....	12
Codice applicazione .....	15
Activity .....	15
Activity Lifecycle .....	16
Activity Stack .....	18
Struttura Directory .....	19
Il motore di gioco.....	20
GameLoop .....	21
SurfaceView ed EngineGame .....	23
DataGraber .....	24
DataGraber per Platform.....	24
DataGraber per Dialog.....	24
Elementi Responsive .....	25
Introduzione .....	25
Posizione Elementi .....	25
Grandezza Elementi.....	26
Scalable Bitmap .....	26

Gli Elementi di Gioco .....	27
Gerarchia Elementi .....	28
Personaggio .....	30
Player .....	30
Animazioni .....	31
Gestione del Movimento .....	33
Gestione delle collisioni .....	34
Gestione del Movimento Altri Elementi .....	35
I suoni .....	37
SoundBG .....	37
Suoni .....	39
Dialoghi .....	40
Gestione Passaggio tra livelli .....	42
EnvironmentContainer .....	42
Salvataggio del livello .....	44
Osservazioni .....	45
Problematiche .....	45
GarbageCollector di JAVA .....	45
Road To Amsterdam e Huawei .....	45
Conclusioni .....	46
Riuscita progetto .....	46
Video Gameplay .....	46
PlayStore .....	47
Riassunto Commit GitHub .....	48

# Introduzione

## Android

Con il termine Android si indica un determinato sistema operativo principalmente destinato ai dispositivi mobile (Smartphone e Tablet), ma anche ai wearable (Smartwatch), multimedia automotive (AUTO) oppure anche TV.

Lo sviluppo di Android è nato nel 2005 da parte di Google e negli anni molteplici versioni e cambiamenti sono stati fatti a pari passo con il cambiamento della tecnologia hardware su cui esso è presente.



Tecnicamente non lo possiamo catalogare come sistema operativo, infatti esso è un firmware dove, semplificando, all'interno sono presenti anche dei "driver" destinati alla gestione della parte telefonica e di comunicazione con la rete.

Android è basato sul Kernel di Linux, ed infatti ha molte cose in comune con esso, lo stesso firmware è scritto in C, C++, la gestione dei permessi e del super user è la stessa.

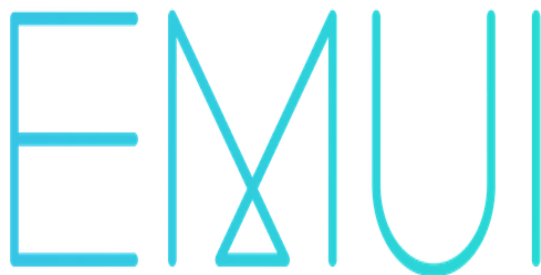
Gli aggiornamenti delle versioni di Android vengono portati avanti da Google Developer che a cadenza annuale rilascia una nuova versione "Stock" di Android, è la cosiddetta versione "pulita".

Il codice di Android Stock rilasciato da Google è Open Source, e ogni casa produttrice di Device può decidere di modificare il firmware, apportando modifiche consistenti o meno, dal miglioramento grafico della UI Android, alla gestione dei File.

Si deve considerare il fatto che nel 2019 Google Developer ha rilasciato l'ultima attuale versione di Android Stock cioè la Android 9 Pie.

Samsung ha modificato il firmware, apportando delle modifiche al sistema della gestione File con il Sistema KNOX per il quale è presente una maggiore sicurezza informatica.

Huawei ha modificato gran Parte della Interfaccia Utente, chiamando il proprio firmware EMUI e rilasciando più versioni di esso, un'altra importante modifica attuata da questa casa produttrice è la gestione della scheda grafica e del metodo che le immagini utilizzano per apparire sullo schermo. Questo problema ha fatto sì che la nostra applicazione abbia dei problemi in tutti i dispositivi Huawei (Honor compresi).



Esistono inoltre versioni Custom del Firmware chiamate Custom ROM, molto diffuse tra gli utenti "smanettoni" che vengono create e rilasciate da Sviluppatori Indipendenti.

Ogni casa produttrice o sviluppatore può liberamente apportare delle modifiche ad Android, e questo è un importante punto da tenere in considerazione per uno Sviluppatore App Android.

Per esempio le modifiche apportate allo stesso Firmware da parte delle case produttrici possono influenzare gravemente il funzionamento delle Applicazioni create dagli sviluppatori, come è successo nel nostro caso.



Uno Sviluppatore si trova costretto dunque a tenere in considerazione non solo la versione di Android Stock, quindi per esempio rendendo il proprio progetto valido oltre la versione 5.0, ma deve tenere in considerazione anche la versione del Firmware modificata dallo stesso produttore.

## Introduzione agli APK

Alla nascita di Android inizialmente le applicazioni venivano create in C++, con IDE poco pratici e con poca documentazione.

Da Alcuni anni però per il sistema Android è disponibile la realizzazione di Applicazioni create in Java essendo stato introdotto il Dalvik JVM e per ultimo l'Android Runtime (ART), entrambi sono Java virtual machine (JVM) che compilano e decompilano il codice java in bytecode eseguibile da Android.

Il bytecode viene memorizzato in file .dex (Dalvik EXecutable) e .odex (Optimized Dalvik EXecutable) files.

 AndroidManifest.xml	.xml	13.7 KB
 classes2.dex	.dex	5.66 MB

*2 Due file all'interno di un APK*

Una applicazione Android non è fatta solo da codice java, ma sono presenti anche file XML, ed elementi multimediali quali Bitmap, PNG, MP4 ecc...

Gli Android Package (APK), sono degli archivi contenenti questi file, organizzati sotto una certa struttura.

Generalmente un utente tradizionale non deve mai sapere cosa c'è dentro un APK, e neanche reperirli, in quanto le applicazioni sono "hostate" sul Play Store o su altri App Market, generalmente neanche lo sviluppatore deve conoscere interamente la struttura degli APK, in quanto sono gli stessi programmi di compilazione e/o IDE che si occupano di creare o modificare i suddetti archivi.

Quello che è indispensabile allo Sviluppatore di applicazioni Android è conoscere il file Manifest.XML, file importantissimo in quanto all'interno sono presenti la dichiarazione delle activity, i permessi che l'applicazione utilizza e altre informazioni fondamentali.

## Introduzione Android Studio

Android Studio è un integrated development environment (IDE); è un ambiente di sviluppo integrato sviluppato e rilasciato da JetBrains s.r.o.

Questo IDE è basato su IntelliJ IDEA, famoso IDE per lo sviluppo di applicativi JAVA, entrambi condividono i Plugin, le estensioni e la configurazione insieme alla gestione dei parametri che sono molto simili tra i due.



Android Studio è stato comunque sviluppato in collaborazione con Google, per questo motivo infatti tutta la documentazione ufficiale che fa riferimento alle App Android sono orientate chiaramente verso questo IDE.

Con Android Studio uno sviluppatore può modificare il proprio codice sorgente, compilarlo, decompilarlo, modificare i file di Layout XML ed averne una visualizzazione plastica Drag & Drop della anteprima.

## Descrizione Preliminare Progetto

Il progetto che abbiamo dovuto affrontare riguarda lo sviluppo di una Applicazione Android attraverso Android Studio. Inizialmente abbiamo affrontato una fase di ideazione e progettazione della Applicazione, successivamente ci siamo esercitati per imparare ad usare al meglio le tecnologie di Android Studio e infine siamo passati allo sviluppo.

Nella fase di ideazione il problema principale è stato trovare un'idea realizzabile e non banale, che potesse permetterci di creare un progetto interessante e che ci potesse dare anche soddisfazioni personali al di là dell'ambito didattico.

Inizialmente abbiamo avuto tantissime idee, ma molte non rientravano completamente nella consegna assegnata al progetto, e nonostante fossero stati progetti interessanti abbiamo optato per realizzare una cosa più classica e che ci permettesse di approfondire al meglio più conoscenze di Android.

Abbiamo optato per la realizzazione di un videogioco 2D Platform/Adventure narrativo basato sull'esplorazione di livelli e sulla interattività attraverso dialoghi con altri personaggi. In base alle scelte personali che si fanno nei dialoghi il percorso narrativo cambia dando la possibilità a percorsi diversi nella storia narrativa dove sono presenti più finali.

Abbiamo optato per un'applicazione del genere perché ci dava il modo di lavorare su ambiti diversi di Android come per esempio la creazione del movimento del Player fino ad arrivare alla visualizzazione dei dialoghi.

Per approfondire le nostre conoscenze in ambito Android Studio sono servite molte video guide cercate su YouTube e corsi appositi in lingua inglese.

Tra i tanti vale la pena citare il sito [medium.com](https://medium.com) il quale molte volte si è rivelato il più aggiornato e con presenti guide attuali e autorevoli.

Un'altra piattaforma di studio fondamentale è stata la stessa base di documentazione di [Android Developer](https://developer.android.com).

Si è cercato invece di evitare corsi in lingua italiana oppure amatoriali, in quanto molte volte contenevano delle inesattezze e, anche se "funzionava", erano metodi di programmazione sbagliati che non permettevano uno giusto sviluppo.

La fase di sviluppo del progetto, l'ultima, è stata la più lunga, e si è deciso di attuare durante questa fase un procedimento di miglioramento continuo, per il quale gli strumenti e i nuovi concetti necessari per Java necessitassero comunque di uno studio continuo.

In questa maniera siamo riusciti a tenere comunque traccia nel tempo del lavoro svolto senza alienarci in parti del progetto col passare del tempo.

# Progetto

## Caratteristiche Videogioco

Il nostro videogioco si chiama Road To Amsterdam, tradotto "Il viaggio verso Amsterdam".

Road To Amsterdam è un road game, un videogioco su strada letteralmente, cioè è ambientato in più location e l'intera storia è un viaggio in macchina, non vi si dà importanza alla destinazione in sé ma al viaggio che il gruppo deve intraprendere.

I personaggi presenti all'interno del videogioco sono alcuni membri della classe 5IA e altri personaggi inventati creati verosimilmente alle persone che si potrebbero trovare nelle rispettive location.

Il protagonista è Michele Cazzaro, e i suoi compagni di viaggio sono Matteo Carpentieri, Riccardo Fazzi, Lorenzo Salmaso, Carlo Menaggia, Ennio Italiano, Andrei Bobirica e Filippo Morbiato.

Si è deciso Michele come protagonista, e non uno di noi due sviluppatori, perchè è stato lui che ha proposto di organizzare il vero viaggio ad Amsterdam, attualmente programmato per Luglio 2019, essendo lui attualmente impegnato nella organizzazione abbiamo deciso di omaggiarlo dandogli il ruolo di protagonista nella omonima avventura videoludica.

Il videogioco non ha l'obiettivo di porre un livello di sfida, o creare competitività tramite punteggi o sistemi di vita e danno al personaggio; ha l'unico obiettivo di raccontare una storia, trasmettere a chi gioca delle emozioni efficacemente e farlo immedesimare nel ciclo narrativo dando al giocatore delle scelte da effettuare che determinarono il corso delle vicende.

Chiarito il target del videogioco, lo si può paragonare a certi giochi usciti negli anni 90, cioè i punta e clicca, che tramite un sistema di dialogo avevano la grande particolarità di avere una storia avvincente e interessante.

Un altro stile di opera simile sono i libro-gioco, popolari negli anni 80, che tramite delle scelte del lettore concludevano la storia con finali diversi.

Il gioco è diviso in più livelli che formano una storyline, cioè un ciclo narrativo dove sono presenti diramazioni e ricongiungimenti.

Il videogioco è diviso in più parti, c'è la parte video utilizzata per dei brevi intermezzi introduttivi essi servono anche per le parti finali, cioè dei brevi intermezzi conclusivi. La parte platform cioè il gioco in sé, che consiste nel giocatore che muove il protagonista e lo fa interagire con gli elementi della mappa, e l'ultima parte cioè i dialoghi, parte fondamentale in quanto sono tramite essi che si comunica e si esprimono i concetti e le idee nel gioco.

Tutte queste 3 parti sono costantemente correlate tra di loro e interdipendenti.

Si è deciso inoltre di utilizzare dei video creati in precedenza per gli intermezzi introduttivi per tre motivi principalmente:

Il primo è che il tempo necessario al progetto per creare anche un sistema di animazioni testo/immagini con effetti di scorrimento per creare una parte introduttiva ai livelli non era sufficiente, quindi si è deciso di optare per questa soluzione in quanto era più immediata.

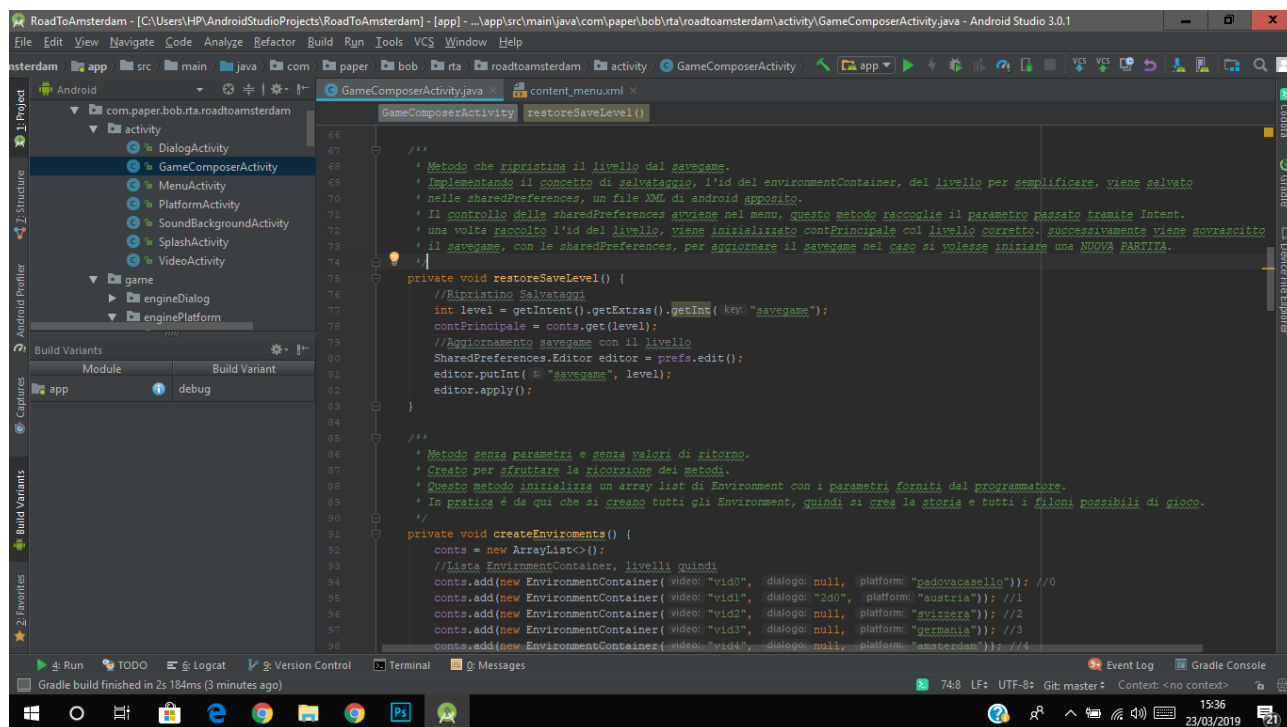
Il secondo motivo è perché la parte di introduzione non è strettamente necessaria al giocatore, si può infatti decidere di saltarla; ha un legame più debole con le altre 2 parti, cioè con la parte del dialogo e del platform, effettivamente giocabili dall'utente.

In conclusione l'ultimo motivo è che "era una pratica comune", attualmente nel 2019 non succede più questo, i videogiochi sono interamente prodotti graficamente dal motore di gioco, e non sono presenti video, ma fino al 2008 era una pratica comune mettere dei video all'interno del videogioco in quanto riduce il peso complessivo del programma e permette maggiore velocità e prestazioni grafiche.

## Strumenti Utilizzati

### Android Studio

Android Studio è stato il primo tra gli strumenti per lo sviluppo utilizzati, le principali operazioni svolte attraverso di esso sono state: la scrittura del codice JAVA, la creazione dei layout delle Activity, l'esecuzione delle applicazioni, la compilazione del progetto, e la creazione del APK della applicazione.



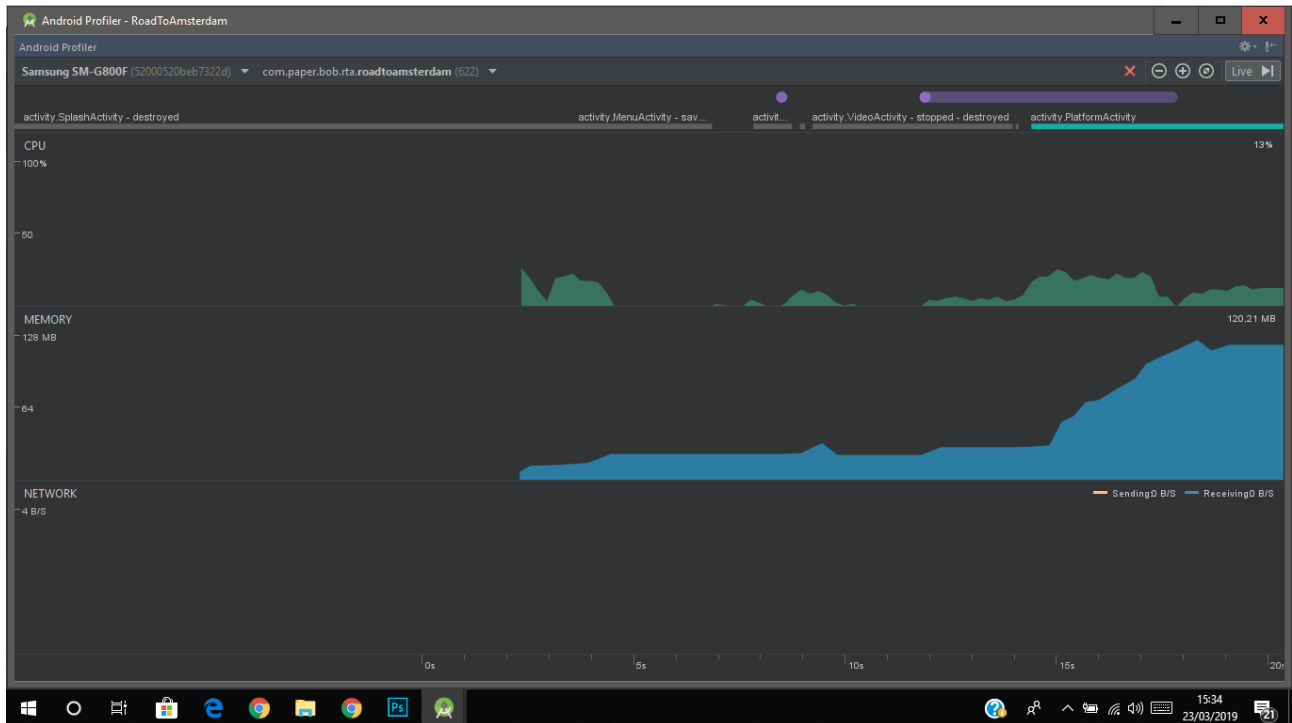
### 3 Esempio di scrittura codice JAVA con Android Studio

Android Studio ha integrato anche un emulatore Android, che noi abbiamo deciso di non utilizzare in quanto è meglio provare le applicazioni su dispositivi veri, sui quali si possono verificare più problemi reali, quindi successivamente si può prevenirli.

Inoltre un altro motivo dell'esecuzione delle applicazioni su dispositivi fisici è il fatto che il gioco deve per forza essere giocato tramite touch screen.

## Android Profiler

Android Studio ha integrato anche un altro strumento cioè Android Profiler, strumento utile per analizzare le prestazioni dell'applicazione in fase di esecuzione ed esaminare le risorse utilizzate sul dispositivo.



### 4 Esame prestazioni App con Android Profiler

Attraverso Android Profiler siamo riusciti a capire le risorse utilizzate dalla nostra applicazione e individuare le falle nel loro utilizzo, riuscendo così a prevenire eventuali crash o blocchi nella applicazione.



## CVS

Android studio è ricco di strumenti aggiuntivi di esame del codice, di correzione degli errori, e molti Plugin scaricabili permettono di facilitare la vita dello sviluppatore ancora di più.

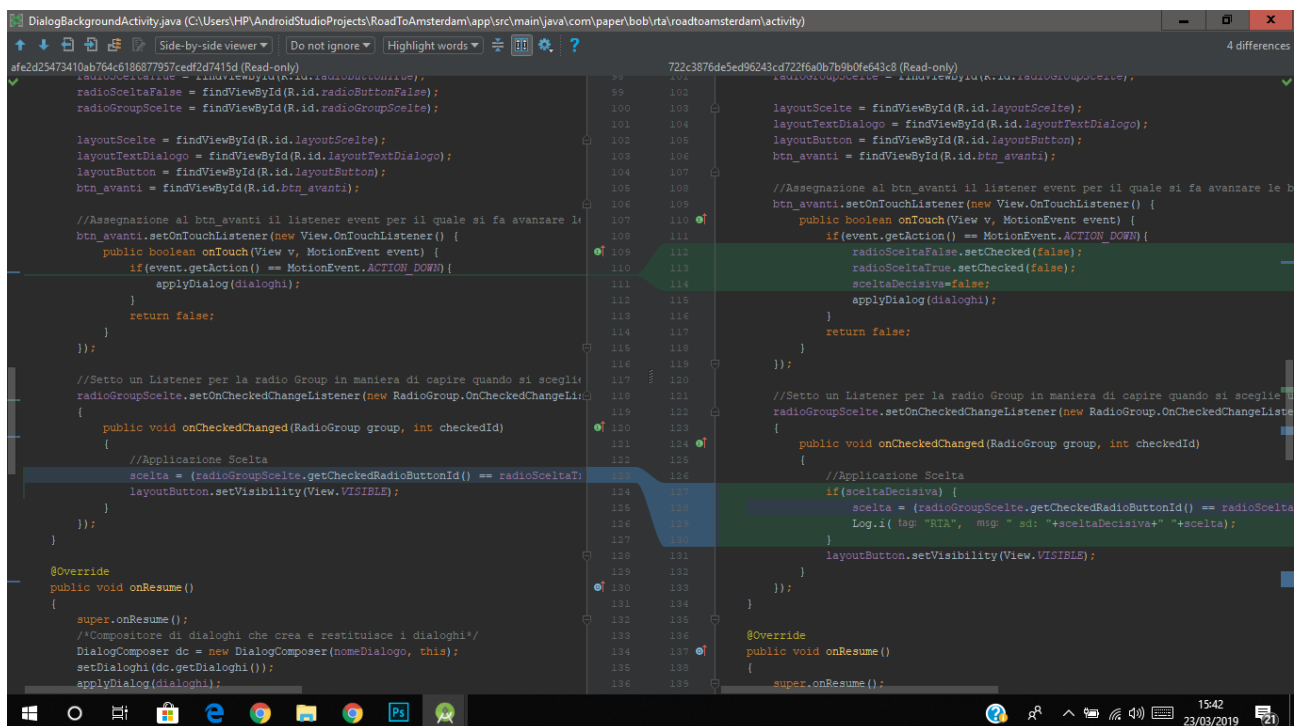
Uno tra questi strumenti è una implementazione di Android Studio con il mondo dei CVS.

I CVS, Concurrent Versions System, come dice il nome, sistema di versioni concorrenti è un sistema per la gestione di più versioni del programma, per il quale si tiene traccia delle versioni precedenti e delle modifiche che si sono effettuate.

Per implementare i CVS si è dovuto installare Git e associarlo ad Android Studio, Git non è altro che un software che tramite un sistema controlla le versioni di un certo programma. In pratica tiene traccia delle versioni del codice anteriori e le tiene sempre disponibili in caso se ne fosse necessitati.

Git crea dei commit, cioè delle versioni di programma, salvate in locale, che occupano poco spazio e sono ottimizzate.

Una particolare e fondamentale utilità è la chiara visualizzazione delle parti di codice cambiate nel tempo.

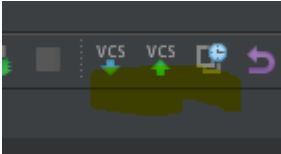


5 Confronto Commit tramite VCS

## GitHub

Precedentemente abbiamo spiegato il CVS integrato con Android Studio e Git, ma un altro strumento che è stato indispensabile durante la realizzazione del progetto è stato Github, esso è un sito che si occupa di hostare i commit creati da Git, in maniera da poter successivamente avere le versioni del progetto anche in Cloud e accedervi in maniera remota.

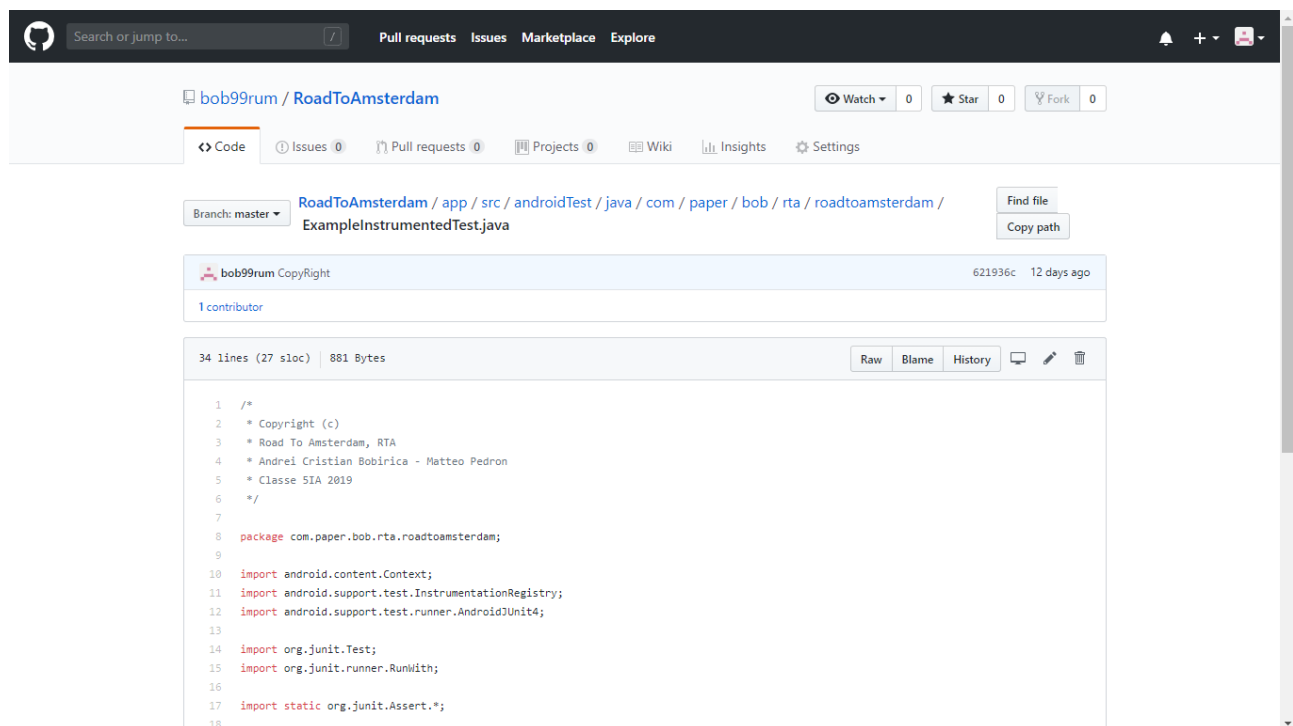
Collegandosi al link <https://github.com/bob99rum/RoadToAmsterdam> si può accedere alla repository di tutto il progetto visualizzando tutti i commit e le modifiche attuate al progetto nel tempo.



Le utilità di utilizzare GitHub con Android Studio sono molteplici, il primo motivo è perché anch'esso è implementato e tramite un singolo pulsante si può caricare tutto in Cloud o aggiornare il progetto dal Cloud.

Un'altra utilità è che lavorando in gruppo si è più collaborativi ed entrambi i partner sono in grado di rimanere aggiornati con il lavoro dell'altro, riuscendo ad ottimizzare i tempi dovuti a problemi di controllo versione, e/o unione delle parti di un progetto.

Permette inoltre la visualizzazione immediata del codice all'interno di una interfaccia Web.



### 6 Visualizzazione Codice nella pagina WEB GitHub

Avendo utilizzato GitHub siamo riusciti a tenere traccia nel tempo non solo delle modifiche fatte, chiaramente visibili e confrontabili dalla pagina web, ma anche delle note aggiuntive scritte da noi nei commit.

Per un progetto esoso in termini di materiali, tempi e lavoro è stato utile tenere traccia del lavoro svolto.

A fine relazione è possibile visualizzare il riassunto dei commit per ogni data di lavoro, esaminando le modifiche apportate e le nuove aggiunte.

## Photoshop

Photoshop è stato lo strumento utilizzato per la creazione degli elementi grafici, dai personaggi ai background, dalle animazioni agli ostacoli, ogni elemento grafico PNG è stato creato o modificato tramite Photoshop.

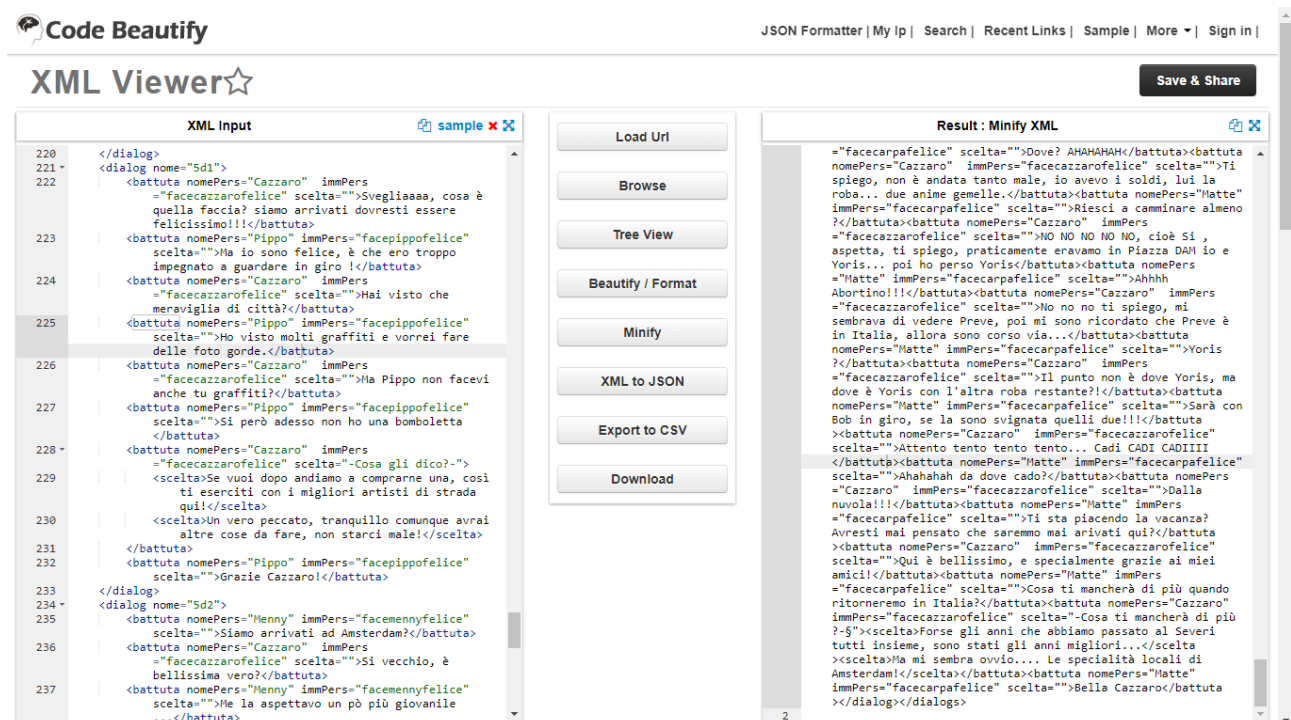
## WanderShare Filmora

Questo software è stato utilizzato per la realizzazione dei video, ha all'interno di sé molti pattern e template già creati e utilizzarlo non ha necessitato nessuno studio di video editing, permettendo così di realizzare comunque dei video di introduzione abbastanza sofisticati, ma permettendoci di focalizzare l'attenzione e il tempo su Android Studio.

## codebeautify.org

Grazie a questo editor online siamo riusciti a interagire in maniera pratica e veloce con il codice XML, all'interno del progetto abbiamo avuto l'esigenza di creare un sistema di lettura di file XML, che non ha nulla a che vedere con i layout XML, e tramite questo strumento siamo riusciti in pochi passaggi a modificare nel tempo questi file dandone l'indentazione quando serviva, e rendendoli minify quando gli si dava in elaborazione all'applicazione.

Nello sviluppo della applicazione ¼ del tempo è stato portato via dalla compilazione di questi file XML, i quali senza questo strumento, avrebbero richiesto molto più tempo.



7 Editing codice XML tramite Code Beutify

## Funzionamento Videogioco

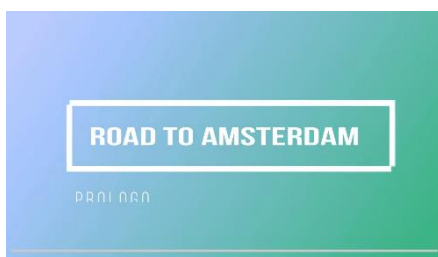
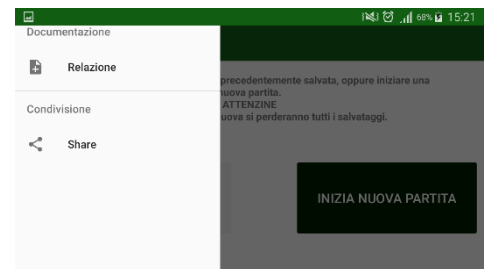


Appena avviata l'applicazione appare una Splash screen, cioè una schermata a caricamento istantaneo che serve per coprire i tempi morti di caricamento di successive parti della applicazione, appena l'applicazione sarà caricata dalla Splash Screen si passerà al menù.

Questa pratica viene utilizzata da quasi tutti i Developer per applicazioni Android ed è caldamente suggerito da Google stesso l'utilizzo in quanto si riducono i tempi morti e si garantisce al utente una esperienza molto più fluida di utilizzo.

Successivamente viene avviato il menù, dal quale sulla barra a sinistra si ha i riferimenti alla suddetta relazione, e in basso il pulsante share, per condividere l'applicazione, il link PlayStore della applicazione, con il sistema di condivisione Android.

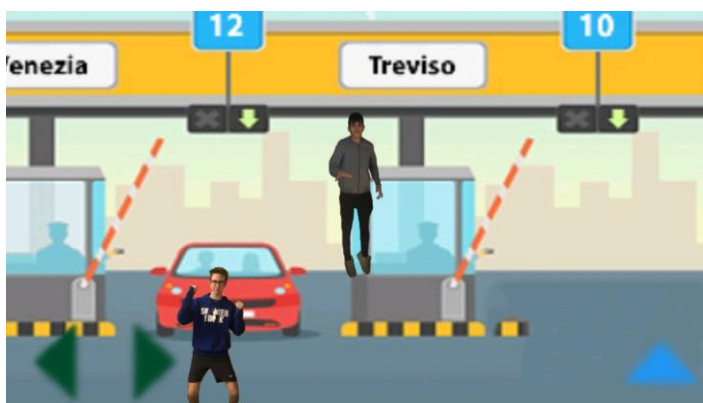
Dal menù si può decidere inoltre se iniziare una nuova partita oppure se caricare quella salvata, se presente.



La parte successiva è l'avvio del gioco, nel senso che viene recuperato il livello precedentemente salvato, oppure si inizia una nuova partita, e viene avviato lo Story Line.

In questa fase parte sempre un video di introduzione di pochi secondi che l'utente può decidere se saltare oppure no.

In ultimo viene avviato il platform, nel quale l'utente attraverso i tasti posti a destra e a sinistra può muoversi nella mappa e interagire con gli ostacoli.



Il giocatore può effettuare tutti i movimenti possibili, tranne nel caso fosse impossibilitato da ostacoli sulla mappa.

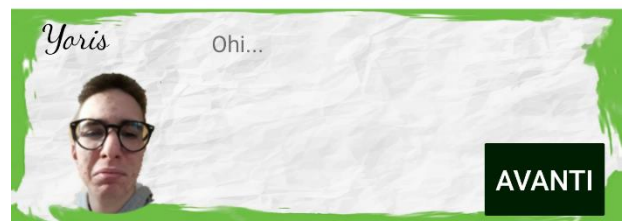
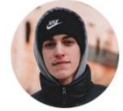
Trascinando il dito sullo schermo ci si può orientare in un certo limite visivo.

I movimenti possibili sono in 2 dimensioni e generalmente è richiesto che il giocatore vada da sinistra a destra, esplorando la mappa.

Quando si trova un personaggio, esso può essere interattivo o no, se è interattivo appena lo si toccherà, partirà il dialogo. Per capire se è interattivo o no lo si può notare dalla presenza della “lampada idea” al di sopra di essi.



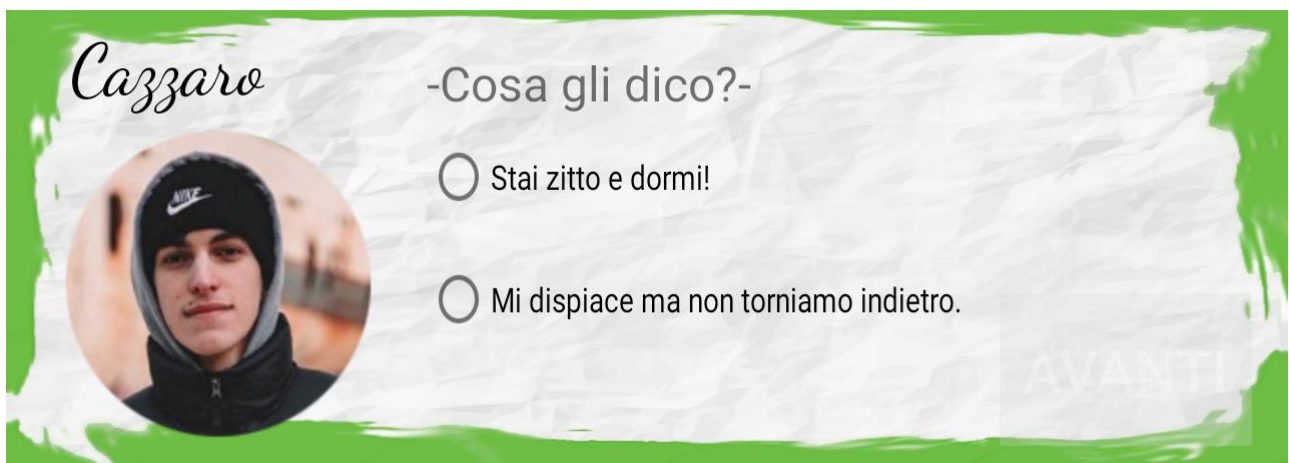
Cazzaro



1 Esempio di un Dialogo

Una volta partita la fase di dialogo inizierà un discorso botta e risposta tra Cazzaro, il protagonista, e l'interlocutore con il quale arrivati in certi momenti si potranno effettuare delle scelte.

Yoris



2 Esempio di scelte all'interno dei dialoghi

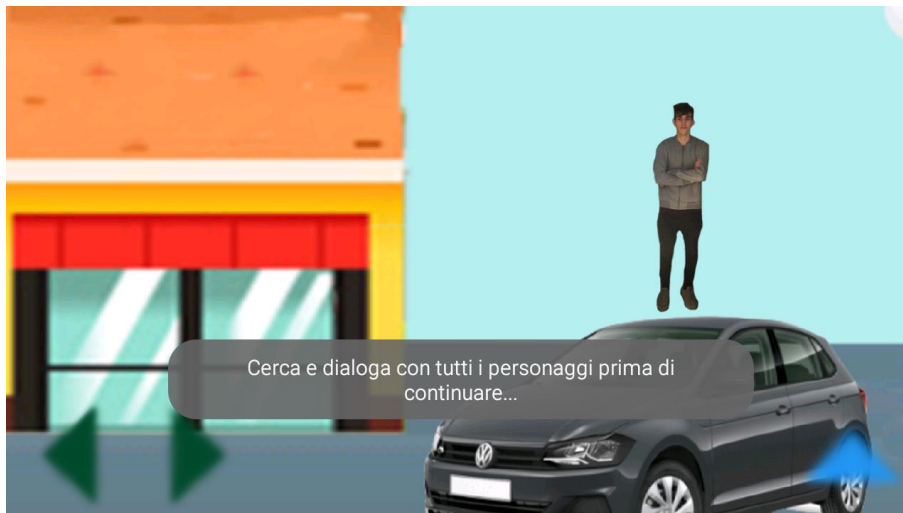
Le scelte possono essere tante, possono essere anche molteplici per un singolo personaggio, in un livello se ne possono incontrare più di personaggi, quindi il numero di scelte sale esponenzialmente.

Generalmente possiamo definire quasi tutte queste scelte, delle scelte finte, nel senso che non servono realmente a influenzare il dialogo, e neanche la storia, in quanto sarebbe stato quasi impossibile creare un sistema che per ogni livello creasse delle sotto diramazioni nell'arco narrativo per ogni singola scelta.

Quello per cui si è optato è più semplice.

Si dà al utente l'illusione di dover fare delle scelte, ma lui in verità essendo preso dal gioco non si rende conto che solamente un dialogo di un singolo personaggio per livello, contiene una scelta fondamentale.

Quella scelta fondamentale in genere è una variabile booleana, true o false, A o B, Si o No, Qui o Lì, questo permette a ogni singolo livello di poter avere due diramazioni, rendendo la creazione della Story Line molto più facile e gestibile.



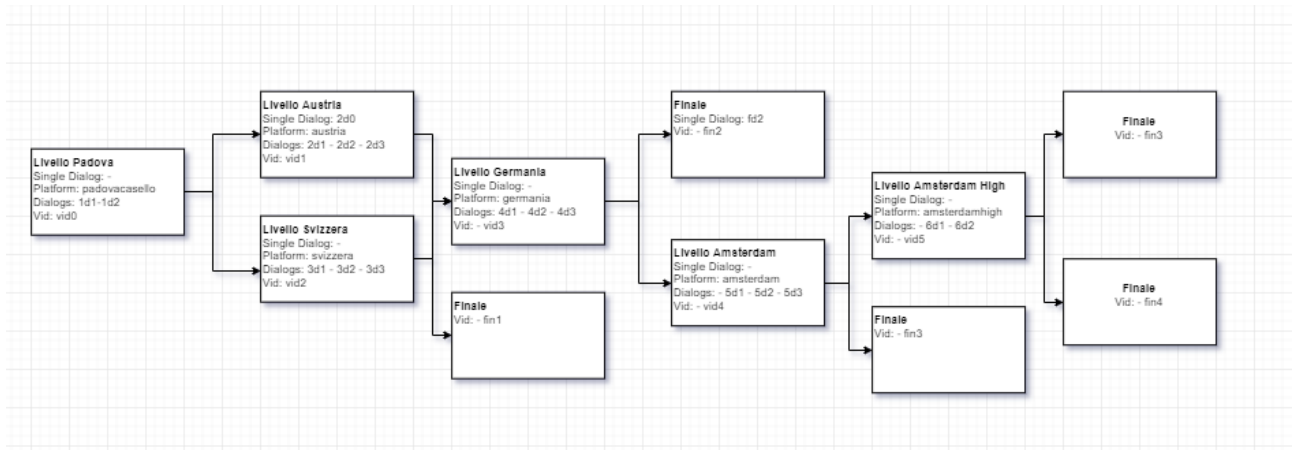
Finito il dialogo si deve esplorare la mappa fino ad arrivare alla parte destra del livello, dove ad ogni livello sarà presente la Polo, arrivare alla macchina significa concludere il livello.

Attenzione però, prima di poter concludere il livello si deve aver dialogato con tutti i personaggi.

Una volta finito il livello il motore di gioco, in base alle proprie scelte, farà progredire l'avventura su un livello diverso fino ad arrivare ad un finale, fatto anch'esso da un video conclusivo "The End".

Questo appena spiegato è il funzionamento del gioco, sono disponibili in totale 6 livelli con 5 possibili Finali.

L'immagine sottostante descrive le sotto-diramazioni della trama che un giocatore può affrontare.



3 Schema con tutti i livelli disponibili e i finali

È possibile a volte che l'ordine struttura Video-Platform-Dialoghi non venga rispettato, e vengano mostrati dei dialoghi introduttivi subito dopo la parte video, oppure un dialogo subito prima la parte video, come nel caso di un finale.

Per elaborare questo modo di creare i livelli si è dovuto creare un motore di gioco il quale è davvero molto flessibile e permette se si vuole, di creare numerose diramazioni, e interscambiare Video, Platform e Dialoghi tra di loro rendendoli opzionali o non.



# Codice applicazione

## Activity

La programmazione di una App Android necessita della comprensione dei concetti di Activity e di Activity Lifecycle.

Una Activity, attività, è una parte della applicazione, generalmente riconducibile a una singola finestra di visualizzazione, che ha uno specifico scopo e si occupa di uno specifico compito.

Le Activity in Android sono definite tramite codice Java e possono essere collegate a una View, cioè a un documento di layout XML che racchiude elementi visualizzabili da mostrare a schermo.

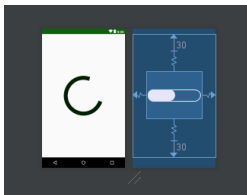
Una Activity è un blocco di operazioni che l'applicazione deve effettuare per realizzare una "attività" e con le quali per visualizzare le informazioni della Activity si utilizzano le View, cioè i layout XML, ai quali si crea un collegamento di dataBinding per rappresentare le informazioni elaborate.

Esempio

Definiamo una classe Activity:

```
public class GameComposerActivity extends SoundBackgroundActivity {
```

Creiamo una View:



Gli settiamo la View:

```
setContentView(R.layout.activity_game_composer);
```

Dichiariamo nel Manifest.XML la nuova Activity, per esempio della splashActivity:

```
<activity  
    android:name=".activity.SplashActivity"  
    android:configChanges="orientation|screenSize|keyboardHidden"  
    android:theme="@style/SplashTheme">  
    <intent-filter>  
        <action android:name="android.intent.action.MAIN" />  
        <category android:name="android.intent.category.LAUNCHER" />  
    </intent-filter>  
</activity>
```

Da notare il fatto che dichiarando questa Activity si è settato anche un Intent filter per il quale si decide quale Activity si avvia per prima all'avvio dell'applicazione e specificando che è l'Activity primaria.

Nella nostra applicazione c'è stata la necessità di utilizzare molteplici Activity, con molteplici View, ed il passaggio tra di esse è stato effettuato tramite gli Intent.

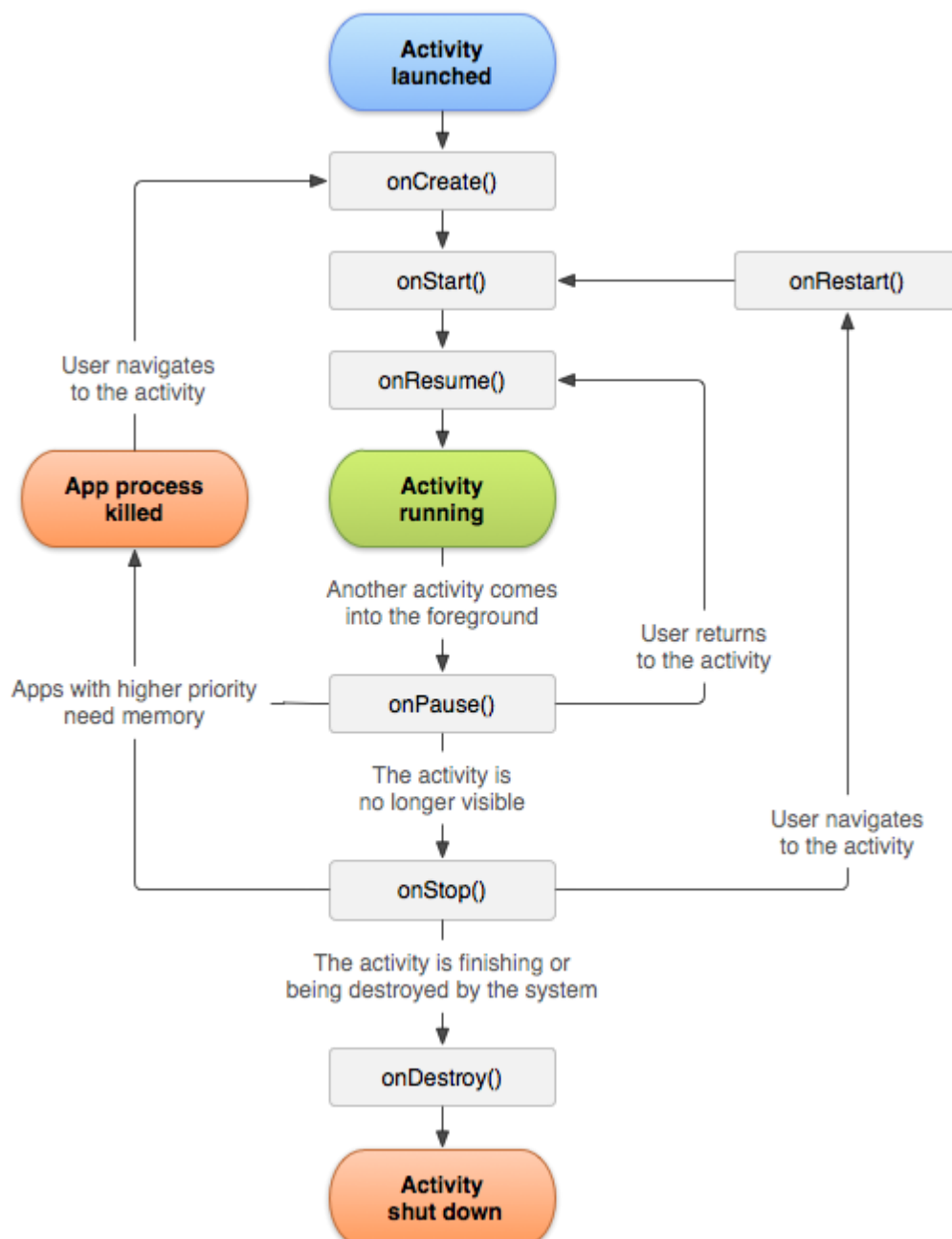
## Activity Lifecycle

Abbiamo detto che una Activity è un insieme di operazioni che vengono eseguite nell'applicazione, ma quelle operazioni non vengono eseguite tutte insieme in maniera procedurale. In base allo stato del dispositivo e allo stato della applicazione vengono richiamati metodi diversi dell'Activity Lifecycle, nei quali si eseguono le suddette operazioni.

Per stato del dispositivo si può intendere se lo schermo è spento oppure no, se viene fatto scattare qualche sensore, se viene premuto qualche pulsante.

Mentre per stato della applicazione si intende in che parte dell'esecuzione essa si sta trovando, in background oppure in foreground.

Le operazioni delle Activity vengono eseguite a cavallo di questi stati, infatti vengono richiamati i metodi dell'Activity lifecycle automaticamente.





### 11 Illustrazione Activity Lifecycle

Come si può vedere nello schema superiore i metodi vengono richiamati in maniera ciclica in base agli eventi scatenanti.

Per esempio appena si avvia l'Activity viene richiamato onCreate(), successivamente onStart() quando l'Activity è visualizzabile ed onResume() quando è interagibile().

Dopo esser stati richiamati i metodi iniziali, l'applicazione generalmente entra in uno stato di paralisi, infatti aspetta un feedback dall'utente, questo stato è il periodo in cui noi utilizziamo l'applicazione.

Successivamente grazie a dei Listener che fanno scatenare degli eventi provocati dall'utente, l'applicazione può cambiare di stato, generalmente finendo l'Activity oppure mandando l'Activity in background.

Per questo motivo adesso vengono richiamati i metodi opposti cioè onPause() quando viene messa l'Activity in background, on Stop() quando viene stoppata l'Activity ed onDestroy() quando viene eliminata l'Activity.

Un dubbio che potrebbe esserci è: perché siamo necessitati a dividere le operazioni in tutte queste Activity? Uno potrebbe pensare che comunque i metodi dell'Activity lifecycle possono essere divisi in due parti, la parte iniziale e la parte finale, e quindi mettere tutte le operazioni da eseguire in questi due blocchi.

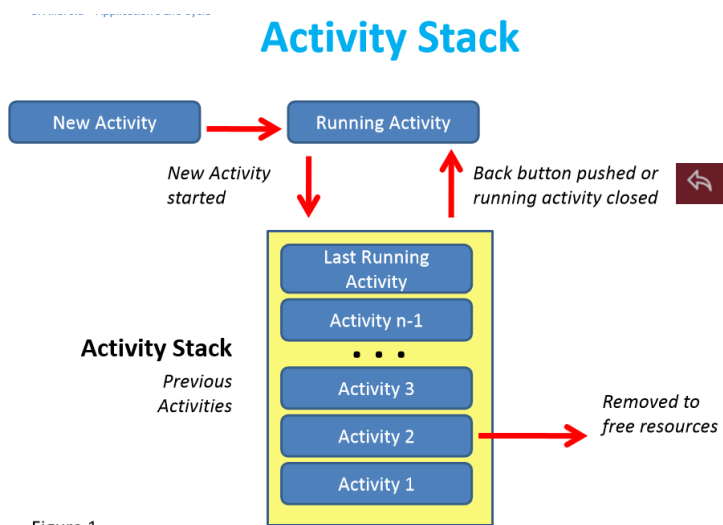
Quindi per esempio mettere tutte le operazioni iniziali di inizializzazione nel metodo onCreate() e le operazioni finali nel onDestroy().

Questo è una cosa sbagliata perché va a gravare sulle prestazioni della applicazione, facendo così si eseguirebbe inutilmente altre operazioni di cui non sarebbe necessaria la riesecuzione.

È come guidare la macchina, si gira la chiave, si guida, ci si ferma e si spegne. Quando si è ad un semaforo e bisogna ripartire non si eseguono tutte queste operazioni da capo, ma si comincia da un determinato punto. La stessa cosa avviene in Android con l'Activity Lifecycle.

Esempio di operazioni dichiarate all'interno di un metodo dell'Activity Lifecycle:

```
/**
 * Metodo onResume richiamato dall'activityLyfeCycle
 */
@Override
public void onResume()
{
    super.onResume();
    /*Compositore di dialoghi che crea e restituisce i dialoghi*/
    DialogComposer dc = new DialogComposer(nomeDialogo, this);
    setDialoghi(dc.getDialoghi());
    //Richiamo la funzione aplyDialoghi che applica i dialoghi alla view,
    mostrandoli e formattandoli sotto forma di Pila
    applyDialog(dialoghi);
}
```

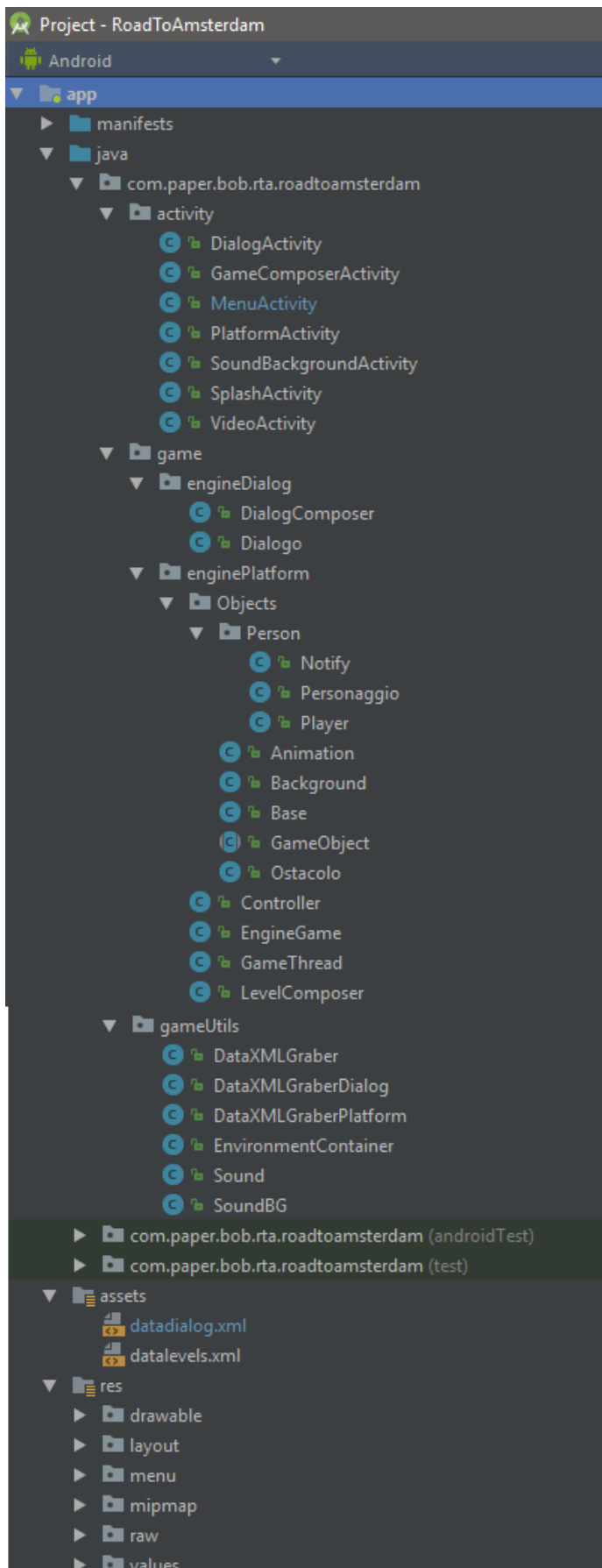


## Activity Stack

Infine è importante spiegare come le Activity rimangono in background e cosa succede quando se ne apre una nuova.

Le istanze delle Activity vengono memorizzate dentro una Pila, quindi seguendo la politica LIFO aprendo o chiudendo una Activity ce ne sarà un'altra sottostante che verrà ripresa.

## Struttura Directory



In Road To Amsterdam si è deciso di strutturare le directory come visibile nell'immagine a sinistra.

All'interno del package principale il codice è suddiviso in maniera tale da distinguerne l'utilizzo.

Nella cartella Activity sono racchiuse tutte le Activity create; nella cartella game è racchiuso tutto il motore di gioco; nella cartella gameUtils sono racchiusi degli strumenti necessari al motore di gioco, delle utility, realizzare comunque da noi, ma che non fanno parte del motore di gioco, per intenderci sono utility che generalmente forniscono informazioni o per esempio le classi per la gestione dei suoni che non forniscono informazioni ma che sono riutilizzabili anche in altri progetti.

Il motore di gioco game contiene le classi necessarie per l'elaborazione delle due parti giocabili, cioè la parte platform e la parte dialog.

Nella cartella engineDialog sono contenute le classi necessarie per la rappresentazione della fase dialoghi.

Nella cartella enginePlatform sono contenute più classi e più cartelle, quindi le classi che servono per la creazione del gameloop, la classe del motore di gioco e le astrazioni degli elementi di gioco quali per esempio Ostacoli o Personaggi.

Nella cartella assets sono contenuti le informazioni necessarie per la generazione dei livelli e dei dialoghi.

Dentro la cartella res sono presenti più cartelle contenenti risorse:

drawable contiene tutti gli elementi disegnabili, quali immagini, immagini xml ecc.

layout contiene tutte le View contenute in file xml.

raw contiene tutte le risorse multimediali come per esempio video o suoni.

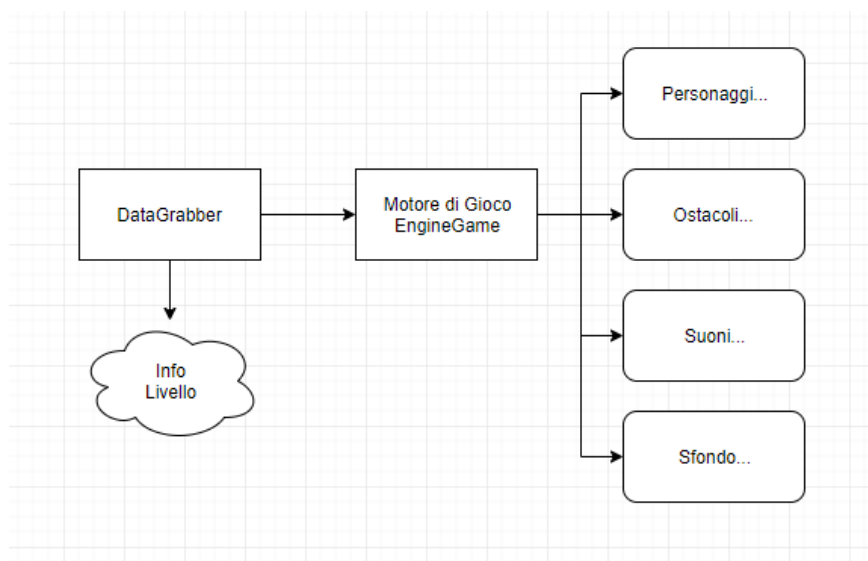
Values contiene dei file xml che servono per la dichiarazioni di valori, cioè di costanti da utilizzare all'interno della applicazione.

## Il motore di gioco

Tenendo in considerazione che il videogioco inizialmente nella fase di progettazione avrebbe dovuto contenere più di 15 livelli, in definitiva ne abbiamo creati 6, ci siamo resi conto subito che creare ogni livello "a mano" sarebbe stato difficile, molto frustrante e ci avrebbe portato via molto tempo.

Allora abbiamo iniziato a ideare il concetto di motore di gioco, cioè un sistema per il quale si definisce un motore in grado di adattarsi e generare automaticamente i livelli in base a dei parametri di input.

Per essere più specifici nel caso in cui avessimo deciso di creare i livelli manualmente sarebbe stato necessario definire per ogni livello le inizializzazioni e definire per ciascuno i valori quali posizione degli ostacoli, posizione dei personaggi ecc.



Si è invece deciso di creare un motore di gioco più flessibile che acquisisce le informazioni che definiscono un livello da file XML (datadialog.xml e datalevels.xml); In questa maniera è molto più rapido e immediato compilare un documento xml che verrà successivamente analizzato dall'utility dataGrabber e che ne estrapoleranno le informazioni. Queste informazioni verranno utilizzate dal motore di gioco per generare i livelli.

Come si può notare dallo schema affianco, è il DataGrabber specifico che prende le informazioni all'interno dei file xml contenuti dentro la cartella assets, crea gli elementi del livello e li restituisce al motore di gioco il quale li fa rappresentare sullo schermo.

Per rappresentare i livelli all'interno del documento datalevels.xml la struttura è questa:

```
<levels>
<level name="amsterdamhigh">
  <ostacoli>
    <ostacolo fisico="true" x="-1280" y="-1280" tipo="muro" frame="1">trasp</ostacolo>
    . . .
    <ostacolo fisico="true" x="6252" y="420" tipo="endlevel" frame="1">polo</ostacolo>
  </ostacoli>
  <background>bgsdamhigh</background>
  <personaggi>
    <personaggio x="4000" y="-700" frame="8" notify="true" dialogo="6d2">permetz</personaggio>
    . . .
    <personaggio x="4000" y="410" frame="4" notify="false" dialogo="6d3">dancer</personaggio>
  </personaggi>
  <player x="500" y="100" frame="8">p1</player>
  <base>grigiopece</base>
  <sounds>
    <sound loop="true" tipo="camminata">camminata</sound>
    <sound loop="false" tipo="salto">salto3</sound>
    <sound loop="true" tipo="ariacaduta">vento</sound>
    <sound loop="false" tipo="caduta">caduta</sound>
    <sound loop="true" tipo="background">spirit</sound>
  </sounds>
</level>
. . .
</levels>
```

Per rappresentare i dialoghi all'interno del documento `datadialog.xml` la struttura è questa:

```
<?xml version="1.0" encoding="UTF-8"?>
<dialogs>
<dialog nome="1d2">
  <battuta nomePers="Yoris" immPers="faceyorisfelice" scelta="">Ohi...</battuta>
  .
  .
  <battuta nomePers="Yoris" immPers="faceyorisfelice" scelta="">Non ce la faccio ...</battuta>
  <battuta nomePers="Cazzaro" immPers="facecazzarofelice" scelta="">No, le macchine ...</battuta>
  <battuta nomePers="Yoris" immPers="faceyorisfelice" scelta="">Ma dai non c...battuta>
  <battuta nomePers="Cazzaro" immPers="facecazzarofelice" scelta="-Cosa gli dico?-">
    <scelta>Stai zitto e dormi!</scelta>
    <scelta>Mi dispiace ma non torniamo indietro.</scelta>
  </battuta>
</dialog>
.
.
</dialogs>
```

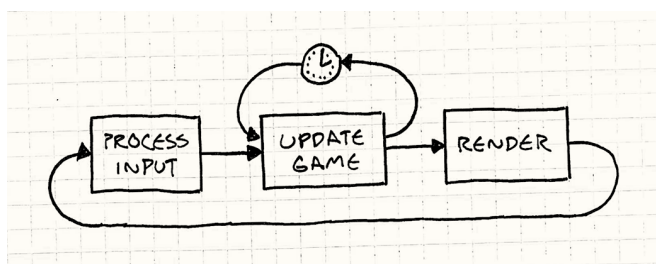
Si può notare a primo impatto che l'intuitività e la facilità di creazione di un livello ora è molto alta, grazie a questo metodo di lavoro abbiamo risparmiato molto tempo e anche in fase di debug è stato molto utile trovare eventuali errori e correggerli avendo le informazioni del livello rappresentati sotto forma di xml.

## GameLoop

Dall'alba della creazione dei videogiochi è sempre esistito il concetto di gameloop, un videogioco potrebbe essere considerato come un elemento multimediale fatto di immagini che si conseguono l'un l'altra e che variano in base ad un input del giocatore. In un videogioco sono presenti anche elementi audio quindi in definitivo possiamo paragonare un videogioco ad un video, che ha quindi una cadenza fissa di frame al secondo (FPS) e che mostra le immagini all'utente con un certo ordine e con una certa tempistica.

Per realizzare questo tutti i motori di gioco al mondo hanno al loro interno, implicitamente od esplicitamente il metodo `update()`; questo metodo contiene le operazioni che si devono richiamare a cadenza di frame garantendo in questo modo l'esecuzione di certe operazioni e la loro visualizzazione a schermo con una certa tempistica.

Immaginiamoci ancora il concetto di immagini concatenate che vengono visualizzate sullo schermo, e il metodo `update` che aggiorna le informazioni del videogioco, che a loro volta influenzano l'immagine che andrà ad essere visualizzata.



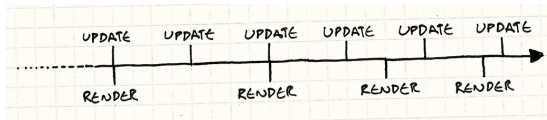
Quando però, per motivi di prestazioni, o di compatibilità con il dispositivo, il numero di FPS cala, questo determina una differenza di operazioni eseguite al secondo, cioè varia in definitiva la velocità con cui le immagini vengono visualizzate ed in maniera concreta la velocità del player.

L'occhio umano necessita minimo di 24 FPS per rilevare una immagine in movimento, noi abbiamo optato per 60 FPS per renderla anche fluida.

Per realizzare un gameloop il modo migliore è quello di realizzare un ciclo infinito in cui al suo interno si eseguono le operazioni necessarie al gioco, cioè il metodo `update()` e il metodo `draw()`.

Il metodo `update` serve per aggiornare le informazioni del gioco (posizioni, controllo collisioni, stato personaggi, ecc.) mentre il metodo `draw()` serve per disegnare i rispettivi elementi del gioco sullo schermo.

Generalmente queste due operazioni vengono eseguite insieme, ma nel caso in cui il numero di FPS varia si possono presentare dei problemi quali la variazione di velocità o peggio ancora una differenza sullo spazio percorso dai personaggi.



Per ovviare a questo problema si è deciso di dividere il metodo update e il metodo draw e non eseguirli sempre insieme; nel caso in cui si ha un numero di FPS minore al target preimpostato, si deve eseguire in maniera

proporzionale ancora n volte il metodo update() in maniera da avere lo stesso spazio percorso in un certo  $\Delta t$ , quindi in definitiva la stessa velocità. Mentre al contrario se si ha un FPS maggiore si deve semplicemente aspettare un determinato tempo t in maniera tale che i FPS tornino ad essere uguali al target preimpostato.

La classe GameThread ha al suo interno il gameloop ed esaminandola si può comprendere questo concetto.

```
@Override
public void run()
{
    Log.i("RTA", "Inizio RUN");
    long beginTime; // the time when the cycle begun
    long timeDiff; // the time it took for the cycle to execute
    int sleepTime; // ms to sleep (<0 if we're behind)
    int framesSkipped; // number of frames being skipped
    while(running) {
        //startTime = System.nanoTime();
        canvas = null;
        //Editing Del Canvas in modo da poter realizzare una immagine dinamica
        try {
            //Log.i("RTAloop", "try lookCanvas");
            canvas = this.surfaceHolder.lockCanvas();
            synchronized (surfaceHolder) {
                //Log.i("RTAloop", "Synchronized update and draw");

                beginTime = System.nanoTime();
                framesSkipped = 0; //resetto gli frame skippati
                // update game state
                this.engGame.update();
                this.engGame.draw(canvas);
                // calculate how long did the cycle take
                timeDiff = System.nanoTime() - beginTime;
                // calculate sleep time
                sleepTime = (int)(FRAME_PERIOD - timeDiff);
                if (sleepTime > 0 && !firstTime) {
                    // if sleepTime > 0 we're OK
                    firstTime=false;
                    try {
                        //Log.i("RTAloop", "Sleep");
                        Thread.sleep(sleepTime);
                    } catch (InterruptedException e) {Log.i("RTA", "Exception look canvas"+e);
                    }
                }
                averageFPS = MAX_FPS;
                while (sleepTime < 0 && framesSkipped < MAX_FRAME_SKIPS && !firstTime) {
                    firstTime=false;
                    // we need to catch up
                    // update without rendering
                    engGame.print("RTA skippp frame ancd update");
                    System.out.println("RTA skippp frame ancd update");

                    this.engGame.update();
                    // add frame period to check if in next frame
                    sleepTime += FRAME_PERIOD;
                    framesSkipped++;
                    averageFPS = MAX_FPS -framesSkipped;
                }
            }
        } catch (Exception e) {
            System.err.println(e.getMessage());
        } finally {
            if (canvas != null) {
                try {
                    surfaceHolder.unlockCanvasAndPost(canvas);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

## SurfaceView ed EngineGame

Una volta definito il gameloop bisogna rappresentare gli elementi di gioco sullo schermo, riprendendo il concetto di immagini concatenate spiegato prima, su Android esistono diversi modi per rappresentare le immagini a schermo. Il primo fra tutti è quello di utilizzare le View e preimpostare degli elementi all'interno dei Layout, in maniera tale che muovendoli appaiano animati.

La scelta per cui abbiamo optato noi è stata diversa invece, abbiamo deciso di utilizzare una SurfaceView, cioè una vista su cui disegnare con il metodo draw elementi al suo interno, generalmente elementi Bitmap quindi immagini. Abbiamo esteso la classe SurfaceView a EngineGame aggiungendo come campi tutti i tipi di elementi di gioco di cui abbiamo bisogno e all'interno dei metodi draw() e update() si richiamano corrispettivamente gli omonimi metodi anche di tutti gli elementi per aggiornare il loro stato, e disegnarli a schermo.

Esempio metodo update()

```
public void update()
{
    //Background
    bg.update();
    //Base
    base.update();
    //Ostacoli
    for (Ostacolo o : ostacoli) {o.update();}
    //Personaggi
    for (Personaggio p : personaggi) {p.update();}
    //Player
    pl.update();
    //Controlle
    control.update();
    //Movimento trascinamento touchscreen
    moveObjectTouch(dx,dy);
}
```

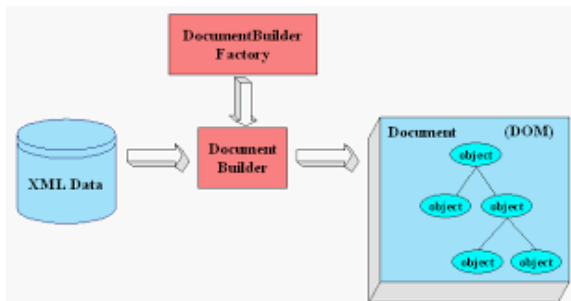
Ci sono altri metodi all'interno del EngineGame che servono comunque alla creazione e alla gestione degli elementi di gioco; è impostato un metodo che muovendo il dito sullo schermo fa traslare tutti gli elementi per un determinato spazio, oppure è impostato il metodo starView() che inizializza tutti gli elementi di gioco richiamandoli dalla utility DataGraberPlatform.

In definitiva il EngineGame è la classe più importante di tutti il gioco, essa fa creare, gestisce e muove tutti gli elementi di gioco.

L'EngineGame, essendo una SurfaceView, ha abbinato un Canvas, cioè una tela, quindi se il SurfaceView è l'area materiale dove disegniamo, il Canvas è la tela su cui poggiamo il pennello.

## DataGraber

Per estrapolare le informazioni dai due file XML `datalevels.xml` e `datadialog.xml`, vengono utilizzate le rispettive classi `DataXMLGraberDialog` ed `DataXMLGraberPlatform`; al loro interno sono presenti dei metodi che tramite la gestione JAVA/XML estrapolano delle informazioni dall'interno degli XML e inizializzano le astrazioni degli elementi che devono restituire.



Per elaborare le informazioni contenute nei file XML si è utilizzato il parser DOM `org.w3c.dom`; in Android sono diffuse anche altre metodologie per l'elaborazione di documenti XML ma la suddetta è stata la più immediata e che abbiamo già avuto modo di imparare in passato.

### DataGraber per Platform

Le informazioni acquisite dal DataGraber per il livello sono caratterizzate da info che definiscono gli elementi della fase platform in specifico:

Gli ostacoli, i personaggi, il player, la base, i suoni ed il suono di background.

Le informazioni riguardanti gli elementi visualizzabili a schermo in genere definiscono la loro posizione, grandezza, le loro proprietà fisiche, la loro texture e la loro animazione se presente.

Queste informazioni vengono usate dal DataGraber per inizializzare gli stessi elementi, i quali verranno poi restituiti al `EngineGame`.

Esempio di metodo DataGraber che ritorna al motore di gioco un tipo di elemento:

```
public Base getBase(String lvName)
{
    Node pl =
    getLevel(lvName).getFirstChild().getNextSibling().getNextSibling().getNextSibling().getNextSibling();
    ;
    int resId = context.getResources().getIdentifier(pl.getFirstChild().getTextContent(),
    "drawable", context.getPackageName());
    Bitmap img = BitmapFactory.decodeResource(context.getResources(), resId, options);
    Base base = new Base(img);
    Log.i("RTA", "- Base");
    return base;
}
```

### DataGraber per Dialog

Una volta che si vuole interagire con un personaggio l'Activity `DialogActivity` ne estrapola il suddetto Dialogo e lo restituisce alla rispettiva Activity sotto forma di Stack.

Esempio di calcolo del nodo Dialog, poi da dove ne verrà estrapolate le informazioni del dialogo e convertite in Stack di Dialoghi.

```
private Node getDialogNode(String infoDialog)
{
    NodeList dialogs= radice.getChildNodes();
    for (int i = 0; i < dialogs.getLength(); i++)
    {
        Node dialog = dialogs.item(i);
        if(dialog.getAttributes().getNamedItem("nome").getNodeValue().equals(infoDialog)) {
            //Log.i("RTA", "getDialogNode");
            return dialog;
        }
    }
    return dialogs.item(0);
}
```



## Elementi Responsive

Sia le coordinate che le grandezze degli elementi vengono definite dal DataGraber perché è questa utility che deve fornire al Motore di Gioco gli elementi già scalati e Responsive.

### Introduzione

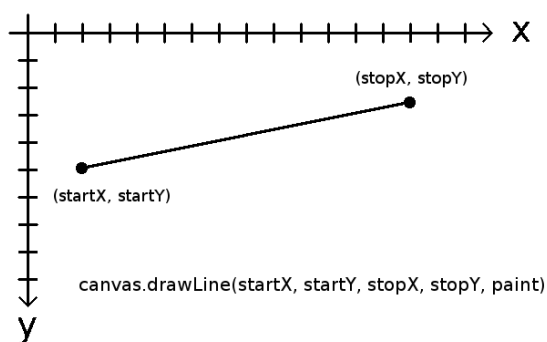
Una applicazione Android già di per sé ha come caratteristica l'essere eseguita su più device, essendo Android il sistema operativo più diffuso al mondo il numero di risoluzioni e di schermi sono numerosi e non si può preventivare come una View definitivamente apparirà a schermo.

È lo stesso problema che affligge le pagine web, con la differenza che HTML è dotata dei framework CSS che in automatico realizzano un layout responsive.

Per realizzarlo in Android Studio bisogna tener conto di alcune accortezze come l'utilizzare come unità di misura gli dpi e i spi, rispettivamente per gli elementi della view e per il testo.

In questa maniera tutti gli elementi della nostra View riusciranno a scalarsi automaticamente in base alla grandezza dello schermo.

Per scalare però gli elementi disegnati sulla SurfaceView ci sono alcuni problemi in quanto la SurfaceView ha una unità di misura spaziale tutta sua ed è solamente in PX.



Possiamo paragonare la superficie di disegno della SurfaceView come il piano cartesiano a sinistra; il punto 0,0 avrà come origini il punto in alto a sinistra, cioè determina una differenza sostanziale con tutte le altre unità di misura e sistemi di misura spaziali di Android Studio.

La cosa più importante da sapere quindi è che non è possibile utilizzare le unità di misura scalabili come i dpi ma sono disponibili solo i PX.

Per ovviare a questo problema abbiamo realizzato dei metodi resizer adattano la grandezza per ogni dispositivo in base alla risoluzione.

### Posizione Elementi

In fase di progettazione di un livello si deve tenere in considerazione che la risoluzione ideale sarebbe 1280/720, quindi a tutti gli elementi vengono date grandezze che necessitano la visualizzazione su uno schermo con questa risoluzione.

Successivamente tramite il metodo position Adapter vengono scalate le coordinate in base alla grandezza della risoluzione effettiva.

```
private int[] positionAdapter(int x, int y)
{
    int[] ret= new int[2];
    int screenWidth = EngineGame.WIDTH;
    int screenHeight = EngineGame.HEIGHT;
    if(x==0)
    {ret[0]=0;}
    else
    {ret[0]=(x*screenWidth)/1280;}

    if(y==0)
    {ret[1]=0;}
    else
    {ret[1]=(y*screenHeight)/720;}
    return ret;
}
```

## Grandezza Elementi



Una volta determinata la coordinata di un elemento può ancora essere definita la grandezza, cioè l'altezza e la larghezza, infatti ogni elemento del videogioco nella fase platform ha un box collider rettangolare.

Ciò significa che tutti gli ostacoli, i personaggi, il player, ecc, sono tutti rappresentati sotto forma di rettangoli e il loro perimetro di collisione è rettangolare.

Essendo loro comunque dei rettangoli nel file `dimen`, per le dimensioni, sono state impostate delle dimensioni prestabilite, ed in base al tipo di elemento definito nel file `xml datalevels.xml` vengono richiamati i suddetti valori memorizzati nel file `dimen.xml`. Questo comporta che queste costanti sono memorizzate in formato `dpi`, e vengono automaticamente tradotte in `PX` dal metodo `getDimensionPixelSize()`.

```
private int[] getGrandezza(String tipo)
{
    int width,height;
    switch (tipo) {
        case "grande":
            width = context.getResources().getDimensionPixelSize(R.dimen.ostacolo_grande_width);
            height= context.getResources().getDimensionPixelSize(R.dimen.ostacolo_grande_height);
            break;
        //...
    }
}
```

## Scalable Bitmap

Abbiamo detto che sul Canvas può essere disegnato qualsiasi cosa, linee, punti, forme geometriche ecc. ma quello che noi abbiamo disegnato maggiormente sono i Bitmap, cioè immagini esterne in formato PNG che vengono prese e disegnate sullo schermo a ripetizione formando quindi una animazione.

I Bitmap sono la cosa più disagiata in Android in quanto sono molto dispendiose di risorse quali per esempio la memoria.

Inoltre provando a disegnare una Bitmap su un canvas verrà disegnata con la grandezza effettiva letta dall'immagine PNG, quindi il concetto di Responsive non è più garantito.

Per ovviare a questo problema si utilizzano delle accortezze.

Innanzitutto tutte le immagini Bitmap vengono caricate risparmiando memoria non per intero ma per metà e se necessario vengono riadattate.

```
options = new BitmapFactory.Options();
options.inSampleSize = 2;
```

Poi successivamente prima di essere disegnate vengono scalate alla loro dimensione voluta, definita prima dalla grandezza degli elementi:

```
canvas.drawBitmap(img, new Rect(0,0,img.getWidth()-1, img.getHeight()-1), new Rect(x,y,x+width,
y+height), null);
```

La seguente linea di codice significa che l'immagine viene scalata grazie ai Rect, quindi ai rettangoli di riferimento, cioè dalla grandezza effettiva dell'immagine si passa a quella voluta con `x,y,width` e `height`.

## Gli Elementi di Gioco

Per elementi di gioco si intendono tutti gli elementi che risalgono nella fase Platform e NON nella fase Dialog, infatti la fase dove sono visualizzabili i dialoghi è realizzata in tutt'altra maniera e viene gestita diversamente dalla propria activity.

La fase platform ha il motore di Gioco, cioè il SurfaceView, acquisisce gli elementi dal DataGraber e li stampa sul canvas successivamente ad averli elaborati.

Gli elementi sono lo sfondo di background visibile su tutto lo schermo in entrambe le schermate.

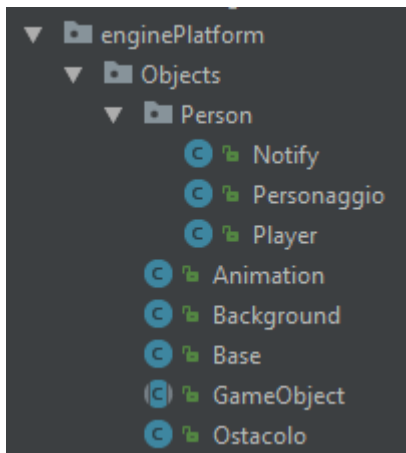


Successivamente è il Player, costituito da Cazzaro, che è l'unico elemento di gioco ad essere dinamico e muovibile dal giocatore.

Esistono successivamente gli ostacoli, un esempio sono i coni visibili nella schermata a sinistra e in ultimo esistono anche i Personaggi, cioè Salmaso visibile a destra. I personaggi inoltre se visualizzabili hanno anche l'oggetto Notify impresso sul canvas.

Sono presenti inoltre altri elementi di gioco quali la Base, i Suoni il suono di Background.

## Gerarchia Elementi



Come visibile nelle directory a sinistra c'è una certa gerarchia tra gli elementi del gioco.

In generale tutti gli elementi del gioco sono Objects, alcuni sono Person e da lì tramite un sistema di estensioni classi differiscono ciascuno in metodi e campi.

Come si può visualizzare nel Diagramma pseudo UML a destra

Le classi estendono l'un l'altra acquisendo specifiche caratteristiche.

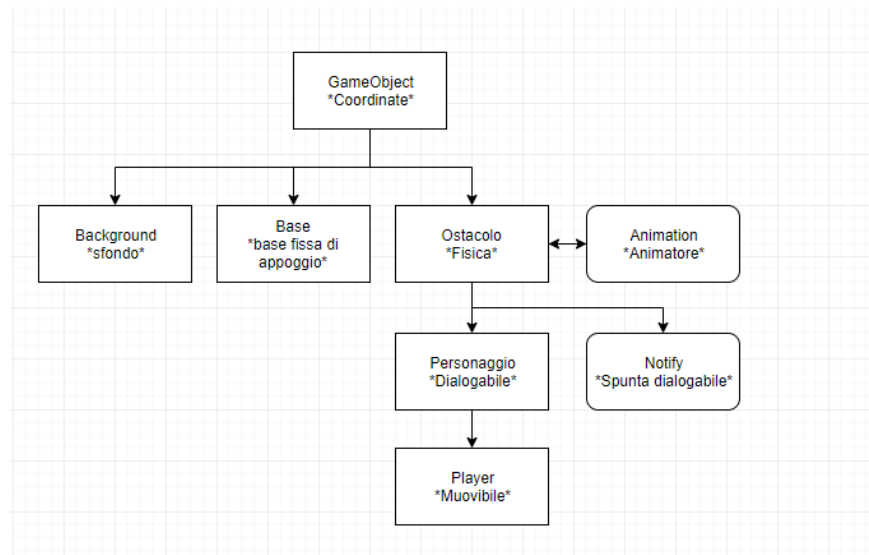
È importante specificare questo perché tutto il sistema di movimento e la gestione delle collisioni riprende il concetto e lo amplia utilizzandolo.

In questa relazione non esamineremo in specifico ogni classe e il suo codice interno in quanto porterebbe via moltissimo tempo e risorse, e non sarebbe essenziale.

Si deve solo tenere in considerazione che ciascuna ha un ruolo e se è diversa da una sua estensione "sorella" c'è un motivo.

Per esempio consideriamo la differenza tra Background e Ostacolo.

Esaminando il codice ci si può rendere conto che differiscono per il tipo di elemento e il come verranno disegnati sullo schermo, inoltre differiscono per il rispettivo ruolo che hanno nella gestione del movimento del Player.



La differenza principale è il come vengono disegnate tramite il metodo draw()

Metodo draw all'interno della classe Background:

```
public Background(Bitmap res,int dx,int dy)
{
    this.dx = 0;
    this.dy = 0;
    image = res;
    y=-EngineGame.HEIGHT*2;
    x=-EngineGame.WIDTH*2;

    coloreSfondo= new Paint();
    coloreSfondo.setColor(Color.WHITE);
    coloreSfondo.setStyle(Paint.Style.FILL);
}
public void draw(Canvas canvas)
{
    //Log.i("RTA",EngineGame.WIDTH+" "+EngineGame.HEIGHT);
    Rect src = new Rect(0,0,image.getWidth(), image.getHeight());
    Rect dest = new Rect(x,y,x+EngineGame.WIDTH*8, EngineGame.HEIGHT*3+y);
    canvas.drawPaint(coloreSfondo);
    canvas.drawBitmap(image, src, dest, null);
}
```

Metodo draw all'interno della classe Ostacolo:

```
public void draw(Canvas canvas)
{
    if(width>0 && x< EngineGame.WIDTH && height>0 && y <EngineGame.HEIGHT)
    {
        if(nframe>1) {
            img = animation.getImage();
        }
        canvas.drawBitmap(img, new Rect(0,0,img.getWidth()-1, img.getHeight()-1), new
Rect(x,y,x+width, y+height), null);
    }
}
```

La differenza è sostanziale infatti il background andrà ad essere disegnato sempre, mentre l'ostacolo no, in quanto se è fuori dall'area visualizzabile dal player non si è necessitati a disegnarlo fuori schermo.

Esaminando il metodo draw() del Background si può notare il fatto che esso ha una grandezza fissa, che è un rapporto tra la grandezza dello schermo moltiplicata per delle costanti.

## Personaggio

```
private String dialogo;  
private boolean notify = false;  
private Notify not;  
  
public Personaggio(Bitmap img, int x, int y, int height, int width, int nframe, String dialogo,  
boolean notify)  
{  
    super(img, x, y, height, width, nframe);  
    //tipo  
    this.setTipo("Personaggio");  
    //Dialogo  
    this.dialogo = dialogo;  
    //Se è Notified, allora è anche fisico, se non è Notified, non è fisico  
    if(notify) {  
        this.setNotify(true);  
        this.setFisico(true);  
        not = new Notify(img, getX(), getY(), getWidth(), getHeight());  
    }  
    else  
    {  
        this.setNotify(false);  
        this.setFisico(false);  
    }  
}
```

Un personaggio ha tutte le caratteristiche degli elementi ereditati prima, ciò significa che ha delle coordinate x,y, una grandezza height e width, un numero di frame per l'animazione ma in più ha anche un riferimento stringa a un dialogo e una variabile booleana che indica se è dialogabile o no.

Esaminando il costruttore si può notare come effettivamente si imposti il personaggio come fisico solamente nel caso in cui esso sia dialogabile.

Nel corso del gioco quando si avvierà il dialogo con esso, gli si imposterà il campo ereditato fisico come false, perché ci si è già dialogati, stessa cosa per il Notify.

## Player

```
public Player(Bitmap img, int x, int y, int height, int width, int nframe) {  
    super(img, x, y, height, width, nframe, null, false);  
    setTipo("Player");  
    this.img = img;  
    setFisico(true);  
    setNotify(false);  
    int delay2 = 50+nframe*10;  
    getAnimation().setDelay(delay2);  
}
```

Il Player è un'eredità di Personaggio, ciò significa che ne è identico solamente per il fatto che non è Notify, quindi non possiede dialoghi, ed ha un delay per l'animazione diverso.

In generale dal costruttore non ci si può rendere conto di come varia il Player, ma esaminando gli altri metodi ce lo si può rendere conto subito.

Il Player è l'unico elemento del gioco a potersi muovere in base ad un input del giocatore, nel senso che premendo dei Button il giocatore potrà muovere il Player, successivamente in base al movimento del Player si muoveranno automaticamente tutti gli altri elementi del gioco.

Il Player essendo destinato a muoversi ha anche delle animazioni diverse da tutti gli altri Ostacoli in quanto non è fermo ma in base al movimento ha un'animazione diversa.

```
//Metodo set PER le immagini  
public void setLeftAnim(Bitmap i){leftAnim = i;}  
public void setRightAnim(Bitmap i){rightAnim = i;}  
public void setJumpLAnim(Bitmap i){jumpLAnim = i;}  
public void setJumpRAnim(Bitmap i){jumpRAnim = i;}  
public void setUpAnima(Bitmap upAnima) {jumpAnim= upAnima;}
```

## Animazioni

Per animazioni di gioco non si intende l'immagine che fatta stampare più volte al secondo, con cadenza di FPS 60, dà l'impressione all'occhio umano di movimento, ma si intende una variazione della suddetta immagine, che oltre al movimento dà l'impressione anche di dinamicità e interattività, ma anche di diversità tra le diverse pose dell'immagine.

Questo concetto è stato realizzato grazie alle spreadshit e agli ArrayList di Bitmap.

Abbiamo pensato che per creare un'animazione la cosa ideale sarebbe stata caricare delle GIF che avrebbero dovuto animare un elemento del gioco.

L'utilizzo delle GIF in Android, oltre che essere difficoltoso, è poco consigliato, e abbiamo utilizzato in alternativa una tecnica molto usata nei videogiochi 2D tutt'oggi.

Abbiamo realizzato delle spreadshit, cioè delle immagini PNG che contengono tutti i Frame di animazione:



Successivamente, una volta saputo il numero di frame, in questo caso 4, lo abbiamo specificato nel documento xml che definiva l'elemento, in questo caso il personaggio:

```
<personaggio x="4000" y="-700" frame="4" notify="true" dialogo="6d2">permetz</personaggio>
```

Una volta definito il numero di Frame attraverso la Classe Animation, di cui si contiene una istanza come campo nella Classe Ostacolo, si può animare il suddetto elemento.

```
public class Ostacolo extends GameObject {  
    ...  
    private Animation animation = new Animation();  
    ...  
}
```

Quello che viene fatto all'interno della classe Ostacolo è, nel caso ci siano animazioni, di creare un Array di Bitmap, dividendo lo spreadsheet in base ai frame specificati:

```
//Animazione se presente
if(nframe>1) {
    Bitmap[] gif = getGif(img);
    animation.setFrames(gif);
    //Se 2 frame = 100 delay - Se 6 frame = 200 delay
    //Per ogni frame 25 di delay in più
    int delay = 50+nframe*25;
    animation.setDelay(delay);
}
```

Nella classe Animation successivamente si provvederà a capire ogni volta, in base al richiamo del metodo update nel tempo, che frame restituire in base al delay di tempo preimpostato e successivamente a restituirli tutti uno dopo l'altro, e creando l'effetto della animazione.

Metodo che restituisce il frame corrente nella classe Animation:

```
public Bitmap getImage(){
    return frames[currentFrame];
}
```



## Gestione del Movimento

Per la gestione del movimento si è utilizzato il metodo del Controller che regola il movimento di tutti gli elementi di gioco.

Abbiamo detto precedentemente che solo il Player si può muovere, il suo movimento non è regolato dalla classe Player implicitamente ma in verità dalla classe Controller, in quanto è essa a decidere se farlo muovere; il Player controlla a cadenza di update() se si può muovere ed esegue le operazioni dettate dal Controller.

Possiamo dire in maniera astratta che è il Controller che fa muovere il Player, ed è il Player che costantemente controlla se può muoversi verificando il Controller.

Il movimento viene effettuato quando il giocatore preme dei Pulsanti presenti sullo schermo.

Avendo impostato dei Listener nell'Activity Platform precedentemente viene registrato l'evento, e viene mandato al Controller il segnale che ci si vuole muovere.

Listener movimento up:

```
btn_up.setOnTouchListener(new View.OnTouchListener() {  
    public boolean onTouch(View v, MotionEvent event) {  
        if (event.getAction() == MotionEvent.ACTION_DOWN) {  
            control.setMUp(true);  
            btn_up.setBackgroundResource(R.drawable.upclick);  
        }  
        if (event.getAction() == MotionEvent.ACTION_UP) {  
            btn_up.setBackgroundResource(R.drawable.up);  
        }  
        return false;  
    }  
});
```

Listener movimento destra, simile a movimento sinistra:

```
btn_right.setOnTouchListener(new View.OnTouchListener() {  
    public boolean onTouch(View v, MotionEvent event) {  
        if (event.getAction() == MotionEvent.ACTION_UP) {  
            control.setMRRight(false);  
            btn_right.setBackgroundResource(R.drawable.right);  
        }  
        if (event.getAction() == MotionEvent.ACTION_DOWN) {  
            control.setMRRight(true);  
            btn_right.setBackgroundResource(R.drawable.rightclick);  
        }  
        return false;  
    }  
});
```

Si può notare che viene richiamato il controller e gli si modificano dei campi interni che indicano la volontà di muoversi.

Si può notare infine un'altra differenza tra i due Listener in quanto per muoversi a destra e a sinistra il movimento ed il tempo duraturo del movimento è determinato da quanto si tiene premuto sullo schermo, mentre per il salto che si tenga all'infinito, o poco tempo, è ininfluente in quanto il salto è di una durata costante e di una altezza costante.

In ogni caso questo è il primo passaggio per la descrizione del sistema di movimento.

Successivamente nella classe Controller avendo impostato dei vettori di movimento, viene mosso il Player.

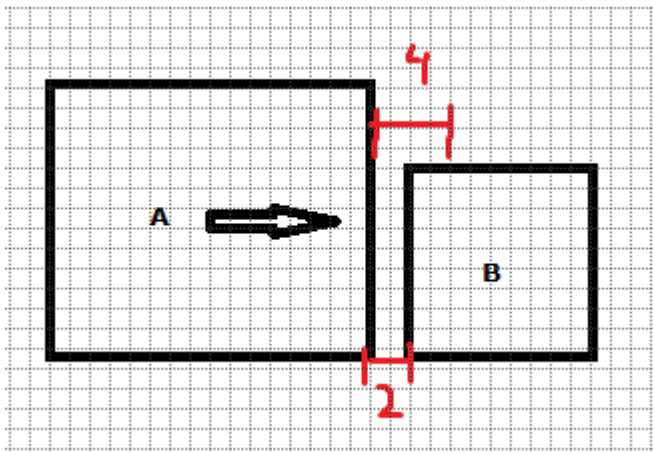
Prima però si deve controllare se il Player si può muovere, nel senso che non è mai garantito che lui possa muoversi, potrebbe incontrare un ostacolo e in quanto tale, nonostante il giocatore preme e ribadisca l'intenzione di muoversi, non glielo si deve concedere.

Per ovviare a questo problema intervengono la gestione delle collisioni.

### Gestione delle collisioni

La gestione delle collisioni avviene dentro il Controller ed utilizza il concetto delle Box Collider, concetto molto diffuso in ambito di sviluppo videogiochi.

Innanzitutto si deve capire come si può verificare la collisione di un elemento con un altro.



Se si vuole muovere l'oggetto A verso destra con un vettore che indica un movimento di + 4 PX, l'algoritmo di gestione della collisione se ne deve accorgere e non lo deve permettere in quanto tra l'oggetto A e B ci sono solo 2 PX e il movimento di + 4 PX indicherebbe una successiva sovrapposizione dei due oggetti.

Viene quindi creato un ArrayList di elementi con cui il Player potrebbe collidere, e si verifica per ogni update() se può collidere con ciascuno di essi.

Prima si verifica se collide con un determinato movimento, e successivamente si autorizza il movimento che viene effettuato; per fare questo basta controllare le posizioni del player sommare al vettore di movimento con l'effettiva posizione degli Ostacoli.

```
private boolean verCol(int dx, int dy)
{
    boolean ret = false;
    for(GameObject g : objColl) {
        if((g.getWidth()+g.getX())>50 && g.getX()<EngineGame.WIDTH+5
            && (g.getY()+g.getHeight())>50 && g.getY()<EngineGame.HEIGHT+50) {
            ret = (collision(new Ostacolo(play.getX() + dx, play.getY() + dy, play.getHeight(), play.getWidth()), g));
            if (ret){
                ...
            }
        }
    }
}
```

Una volta fatto questo viene effettuato un controllo tra l'area del rettangolo del Player e l'area del eventuale oggetto.

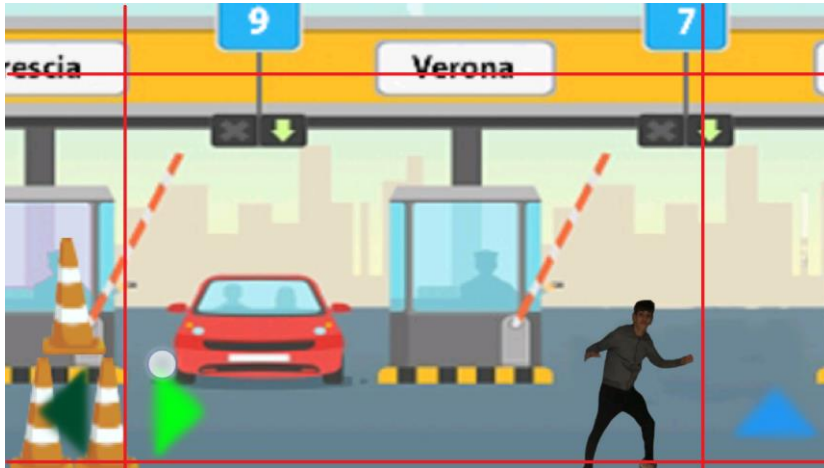
```
public boolean collision(GameObject a, GameObject b)
{
    if(Rect.intersects(a.getRectangle(), b.getRectangle())) {
        return true;
    }
    return false;
}
```

Una volta che il Controller ha autorizzato il movimento del Player, esso nel suo metodo update() aggiorna la sua stessa posizione, e al successivo richiamato del metodo draw() verrà disegnato con la posizione aggiornata, quindi simulando il movimento.

### Gestione del Movimento Altri Elementi

Abbiamo detto che l'unico elemento che può muoversi è il Player, esso si muove a destra, a sinistra, in alto e in basso seguendo una certa fisica.

Se si prova a muovere il Player a fine schermo, ci si potrà rendere conto che esso non esce mai dallo schermo, ma dopo un certo punto, saranno tutti gli elementi a traslare, lui resterà fermo, e questo simulerà un movimento del Player sullo sfondo che trasla.



Superando le relative coordinate visibili a sinistra, tutti gli elementi traslano del vettore al negativo con cui il Player si vuole muovere.

Per esempio come in questo caso, il Player si vuole muovere a destra, lui rimarrà fermo e saranno tutti gli altri elementi a muoversi del vettore negativo, quindi a sinistra.

Lui rimane fermo perché il vettore di movimento suo viene annullato

da quello negativo dettato per tutti gli elementi.

Entriamo nello specifico:

Il movimento del Player è seguito dal background, è esso infatti a muoversi del vettore negativo nel caso il Player superi le coordinate raffigurate nell'immagine.

Tutti gli elementi del gioco, quindi gli Ostacoli seguono a loro volta il background, Player compreso, ciò significa che nel caso il Player voglia fare un movimento che implichi il superamento delle coordinate di limite, automaticamente il background si muoverà del rispettivo vettore negativo, e a loro volta tutti gli Ostacoli, quindi tutti gli elementi seguiranno il background; per il Player seguire il background significa applicare al suo vettore positivo anche il rispettivo vettore negativo donatogli dal background, che provocherebbe l'annullamento del vettore.

Lui rimane infatti fermo, ed è tutto il resto a traslare alle sue spalle.

Nel metodo update del Background:

```
public void update()  
{  
    verifyMovementPlayer();  
    y+=this.dy;  
    x+=this.dx;  
}
```

Qui viene controllato se il Player effettua movimenti che prevedono il superamento delle coordinate di limite:

```
public boolean verifyMovementPlayer()  
{  
    boolean ret = true;  
    //Movimento del Background in base al Player  
    int xm = ((pl.getX()+pl.getWidth())/2+pl.getX())/2;  
    if(xm>(EngineGame.WIDTH/4)*3)  
    {dx = -pl.getDX();ret=false;}  
    else if((xm)<(EngineGame.WIDTH/4))  
    {dx = pl.getDX();ret=false;}  
    else { dx=0;}  
  
    int ym = ((pl.getY()+pl.getHeight())/2+pl.getY())/2;  
    int ymb = (pl.getY()+pl.getHeight());  
    if(ymb > EngineGame.HEIGHT)  
    {dy = -pl.getDDown();ret=false;}  
    else if(ym<(EngineGame.HEIGHT/3))  
    {dy = pl.getDY();ret=false;}  
    else { dy=0;}  
    return ret;  
}
```

L'algoritmo scritto sopra appartiene alla classe Background e rappresenta il sistema per il quale il suo vettore varia in base al volere del Player.

Successivamente variando il vettore del Background, sarà anch'esso a muoversi come visto nel metodo update().

Ora invece esaminiamo quello che succede nella classe Ostacolo:

```
public void update()  
{  
    this.x += bgCoord.getDX();  
    this.y += bgCoord.getDY();  
    if(width>0 && x< EngineGame.WIDTH && height>0 && y <EngineGame.HEIGHT){***}  
}
```

Qui dentro viene controllato il vettore del background, e aggiornate le proprie posizioni in base ai vettori del background.

Ricordando l'ereditarietà degli elementi di gioco, il Player sommando il vettore del background al proprio vettore, ritornerà una variazione nulla.

Facciamo un esempio numerico con traslazione verso un solo lato, a destra.

Il Player si vuole muovere a destra di 10 PX, facendolo supera il limite delle coordinate, il background si muove a sinistra di -10 PX, tutti gli elementi Ostacoli rilevano il movimento e si muovono a loro volta a sinistra di -10 PX, il Player stesso è un ostacolo e muovendosi a sinistra di -10 PX il suo movimento totale è di 0 PX.

## I suoni

I suoni sono un aspetto fondamentale nel nostro videogioco in quanto fanno immedesimare meglio l'utente; possiamo distinguere due tipi di suoni: i suoni di movimento e il suono di background.

Per ogni livello è infatti disponibile un suono di background, che consiste in una colonna sonora, una canzone che si sente mentre si gioca, viene eseguita sia nella Activity Platform che nella Activity Dialog, quindi in entrambe le fasi di gioco.

Le risorse musicali ed in generale gli audio, devono essere fornito sotto formato mp3 oppure wav nel percorso raw.

### SoundBG

Sound Background è una classe utility che viene utilizzata per eseguire un brano in background, sono infatti utilizzati i MediaPlayer a differenza dei suoni in cui vengono utilizzati i SoundPool.

```
/**Cstruttore*/
public SoundBG(int resId, Context context)
{
    mp = MediaPlayer.create(context, resId);
    mp.setLooping(true);
}

/**Metodo che fa partire l'audio*/
public void play()
{
    mp.start();
}

/**Metodo che fa rimanere in pausa l'audio*/
public void stop()
{
    mp.pause();
}
```

Provando ad eseguire una data risorsa musicale nella Activity platform, quindi all'avvio di un livello, ci si renderà conto che appena si avvia un Dialog con un personaggio, il suono viene stoppato, in quanto si sta avviando una nuova Activity.

Il nostro intento era quello di garantire il funzionamento fluido della musica di Background sia nella Activity Platform che nella Activity Dialog; per ovviare a questo problema esistono parecchie soluzioni abbastanza elaborate e dispendiose di risorse. Quello per cui abbiamo optato noi è creare una Activity madre che sarebbe poi stata estesa dalle due Activity Dialog e Platform, riuscendo quindi ad avere i campi statici in comune, e garantendo il mantenimento dei riferimenti di memoria delle istanze una volta che si fa il passaggio da Activity Platform a Dialog e viceversa.

```
public class SoundBackgroundActivity extends AppCompatActivity {

    /**Campo SoundBG contenente il riferimento SoundBG adebito al play o al pause della musica di background*/
    private static SoundBG soundBG;
    /**Campo che indica se la musica di background è stata già avviata oppure no*/
    private static boolean soundBGplayed = false;
```

```
public class DialogActivity extends SoundBackgroundActivity {
public class PlatformActivity extends SoundBackgroundActivity {
public class GameComposerActivity extends SoundBackgroundActivity {
```

Ed eventualmente anche la classe che gestisce il cambiamento di livello, che nella fase di caricamento deve comunque gestire l'audio.

L'esecuzione effettiva della musica viene gestita dalla Activity SoundBackground che al suo avvio col metodo onResume() controlla se sia necessario l'avvio della musica, e allo spegnimento definitivo della Activity, quindi onStop() viene spenta anche la canzone.

Metodi Activity SoundBackground:

```
/**
 * Metodo play che avvia la musica di background
 */
public static void play() {
    if(!soundBGplayed) {
        //Log.i("RTAbg", "playcanzone");
        soundBGplayed = true;
        soundBG.play();
    }
}

/**
 * Metodo stop che stoppa la musica di background
 */
public static void stop()
{
    if(soundBGplayed) {
        //Log.i("RTAbg", "stopcanzone");
        soundBGplayed = false;
        soundBG.stop();
    }
}

/**
 * Metodo onPause richiamato automaticamente dall'activity Life Cycle
 */
@Override
protected void onPause()
{
    super.onPause();
    if(soundBGplayed) {
        Log.i("RTAbg", "onPause bg sound");
        soundBG.stop();
        soundBGplayed = false;
    }
}

/**
 * Metodo onResume richiamato automaticamente dall'activity Life Cycle
 */
@Override
protected void onResume()
{
    super.onResume();
    if(!soundBGplayed && soundBG!=null)
    {
        Log.i("RTAbg", "onResume bg sound");
        soundBG.play();
        soundBGplayed = true;
    }
}
```

## Suoni

Per la gestione dei suoni di animazione, viene utilizzata la tecnologia SoundPool, la differenza con i Media Player è proprio il loro target di utilizzo, infatti i SoundPool sono realizzati per suoni di piccola taglia come brevi effetti audio.

Viene utilizzata l'utility Sound, molto simile nella costruzione a SoundBG, tranne per la tecnologia utilizzata.

I suoni delle animazioni funzionano solamente nella fase di Platform, e vengono gestiti in base al movimento, è il Controller infatti che in base al movimento oppure alla eventuale collisione, fa partire il suono.

Per esempio per il suono del salto, nel metodo getMUp() del Controller:

```
public boolean getMUp() {
    boolean ret = (!verCol(0,-dy) && mUp);
    if(ret)
    {
        accelerateDY();
        playSoundUp();           //FACCIO PARTIRE IL SUONO
        pauseSoundRL();
        alreadyCrush=false;
        alreadyDown=false;
    }
    . . .
}
```

Allora nel metodo playSoundUp() avrò:

```
private void playSoundUp() {
    if(!alreadyUp) {
        alreadyUp=true;
        for (Sound s: sounds) {
            if(s.getTipoSound().equals("salto"))
            {s.replay();break;}
        }
    }
}
```

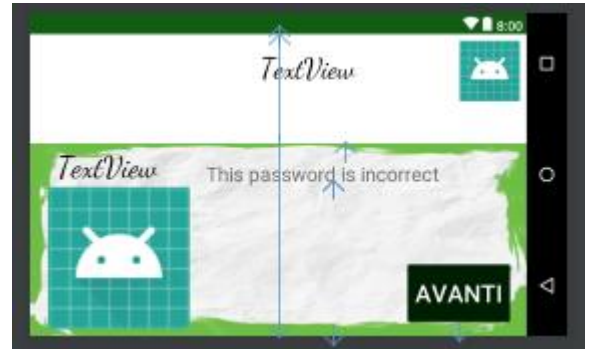
In questa maniera vengono eseguiti i suoni in base al movimento, come avveniva per il Player che era costantemente in ascolto di comandi per muoversi dettati dal Controller, anche i suoni sono assoggettati a questa classe.

Il controller gestisce infatti anche i suoni di animazioni oltre che le animazioni.

## Dialoghi



Per la gestione dei dialoghi viene utilizzata l'Activity Dialog, la quale ha associata una View fatta in xml con un sistema di layout per i quali a scomparsa e comparsa vengono visualizzati elementi diversi del layout.



Fondamentalmente la View dei dialoghi deve solamente mostrare due tipi di dialogo, il testo semplice botta e risposta, e la scelta tramite due Radio Button.

L'Activity Dialog utilizza questa View per mostrare in maniera consecutiva i dialoghi.

Per fare ciò, una volta presi dal DataGraberDialog, esso li restituisce sotto forma di Stack Dialoghi.

Un discorso verbale nel videogioco è inteso come un dialogo dove sono presenti più battute botta e risposta tra due interlocutori e che a volte mettono il protagonista nella condizione di fare una scelta invece di rispondere con una battuta.

```
<dialog nome="1d2">
  <battuta nomePers="Yoris" immPers="faceyorisfelice" scelta="">Ohi...</battuta>
  <battuta nomePers="Cazzaro" immPers="facecazzarofelice" scelta="">Ciao Yoris, come va?</battuta>
  <battuta nomePers="Yoris" immPers="faceyorisfelice" scelta="">Sono un pò stanco...</battuta>
  . . .
  <battuta nomePers="Cazzaro" immPers="facecazzarofelice" scelta="-Cosa gli dico?-">
    <scelta>Stai zitto e dormi!</scelta>
    <scelta>Mi dispiace ma non torniamo indietro.</scelta>
  </battuta>
</dialog>
```

Viene quindi in aiuto la classe Dialogo:

```
public Dialogo(String nomeDialogo, String nomePers, String nomeOtherPers, String nomeImmPers, String
nomeImmOtherPers, String battuta, String scelta, ArrayList<String> scelte)
{
  this.nomeDialogo=nomeDialogo;
  this.nomePers = nomePers;
  this.nomeOtherPers = nomeOtherPers;
  this.nomeImmPers = nomeImmPers;
  this.nomeImmOtherPers=nomeImmOtherPers;
  this.battuta=battuta;
  this.scelta = scelta;
  this.scelte=scelte;
}
```

Essa ha quindi il riferimento a una battuta di un dialogo e l'eventuale successiva scelta da effettuare.

Essendo i dialoghi botta e risposta, il tipo di struttura dati ideale per il loro utilizzo è la pila, infatti una volta creata una pila dove si vanno a interscambiare dei dialoghi botta e risposta, seguendo la politica LIFO, se ne va a prendere quelli superiori, andando ad estrarli dalla struttura dati.



Questo viene fatto dal metodo `applyDialog()` presente dentro l'Activity Dialog:

```
public boolean applyDialog(Stack<Dialogo> dialoghi)
{
    if(dialoghi.size()>0) {
        Dialogo d = dialoghi.pop();
        . . .
    }
}
```

In definitiva abbiamo l'effetto sperato, cioè il dialogo verrà esaminato, fatto abbinare alle rispettive View, e mostrato all'utente; verrà animato brevemente tramite la classe `scrollTest()` e aspettato il successivo input del giocatore sul pulsante avanti.

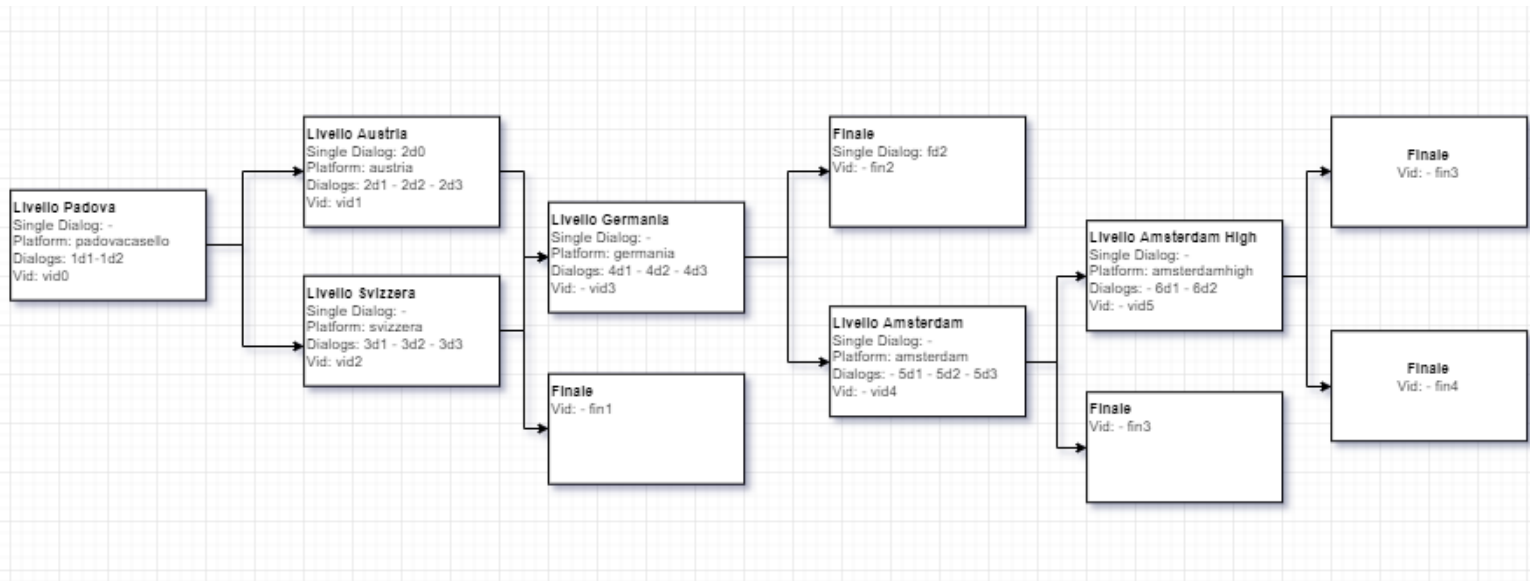
Una volta estrapolati tutti i Dialoghi dallo Stack l'Activity viene chiusa e si ritorna all'Activity precedente, seguendo l'Activity Stack System, quindi all'Activity Platform il più delle volte.

## Gestione Passaggio tra livelli

Per la gestione del passaggio tra i livelli, va innanzitutto definito cos'è un livello, è infatti un insieme di 3 fasi ciascuna delle quali opzionali fatte da: fase video, fase Single Dialog e fase Platform.

Tutte e 3 le fasi sono opzionali e attenzione, la fase di Single Dialog non è la fase dialoghi che si potrebbe pensare normalmente; non sono i dialoghi che partono quando si tocca un altro personaggio, è invece un dialogo opzionale introduzione che è assistente.

Si nota quindi lo schema dei Livelli:



Esaminandolo si può notare come ci siano 2 situazioni particolari:

- Il livello Austria ha un Single Dialog
- Il finale numero 2 ha anch'esso un Single Dialog.

Per rappresentare un sistema di livelli del genere si è dovuto creare una Struttura dati in grado di contenere il riferimento Stringa a ciascuna delle 3 parti di un livello.

### EnvironmentContainer

```
public class EnvironmentContainer {
    private String video;
    private String dialogo;
    private String platform;
    private Boolean scelta;//IF True A Else False B
    private EnvironmentContainer contA;
    private EnvironmentContainer contB;
```

Esaminando questa struttura dati si può notare che si ha il riferimento a ciascuna delle tre parti di un livello, e successivamente si ha una variabile Booleana scelta.

Quest'ultima indica la scelta effettuata nel Dialog veritiero incontrato toccando un personaggio all'interno della fase platform.

Ciò implica che il campo Boolean può acquisire 3 valore:

- Null non si ha ancora effettuato la scelta (manca ancora la fase platform da essere completata)
- True
- False

Detto ciò, una volta effettuata la scelta, quindi avendo la variabile Booleana impostata si passa al EnvironmentContainer Successivo in base alla scelta, si ha quindi ai campi contA e contB il riferimento al EnvironmentContainer successivo.

Il passaggio e l'esame di questa struttura dati e delle relative istanze viene effettuato dall'Activity GameComposer.

Nel metodo createEnvironments() si può notare come vengano create le istanze dei livelli.

Successivamente viene avviato il gioco, quindi il primo livello, oppure il livello salvato tramite il metodo startGame(). Questo metodo viene richiamato nel onResume() e si richiama da solo in maniera ricorsiva.

```
public void startGame() {
    //Inizio operazioni di scelta parti del EnvironmentContainer
    if (!checkVideo && contPrincipale.getVideo() != null) {
        checkVideo = true;
        Intent i = new Intent(this, VideoActivity.class);
        i.putExtra("video", contPrincipale.getVideo());
        startActivity(i);
    } else if (!checkDialogo && contPrincipale.getDialogo() != null) {
        checkDialogo = true;
        Intent i = new Intent(this, DialogActivity.class);
        i.putExtra("nomeDialogo", contPrincipale.getDialogo());
        startActivity(i);
    } else if (!checkPlatform && contPrincipale.getPlatform() != null) {
        checkPlatform = true;
        Intent i = new Intent(this, PlatformActivity.class);
        i.putExtra("platform", contPrincipale.getPlatform());
        startActivityForResult(i, 1);
    } else {
        contPrincipale.setScelta(scelta);
        if (contPrincipale.verifyScelta()) {
            //Prelevare il livello successivo
            Log.i("RTA", contPrincipale.toString());
            contPrincipale = new EnvironmentContainer(contPrincipale.getNext());
            //Aggiornamento savegame ocn il livello
            SharedPreferences.Editor editor = prefs.edit();
            editor.putInt("savegame", contPrincipale.getId());
            editor.apply();
            int savegame = prefs.getInt("savegame", 0);
            Log.i("RTA", "SaveGame Aggiornato:" + savegame);
            //Annullamento dei vari check
            checkPlatform = checkDialogo = checkVideo = false;
            //Ripartire con il metodo ricorsivo
            startGame();
        } else {
            Log.i("RTA", "\n\t@END GAME");
            //Aggiornamento savegame ocn il livello
            SharedPreferences.Editor editor = prefs.edit();
            editor.putInt("savegame", 0);
            editor.apply();
            int savegame = prefs.getInt("savegame", 0);
            Log.i("RTA", "SaveGame Aggiornato:" + savegame);
            //fish
            finishAffinity();
            System.exit(0);
        }
    }
}
```

è importante esaminare il metodo startGame() in quanto in esso sono racchiusi molti concetti indispensabili per la realizzazione del gioco. Si nota innanzitutto come tramite le prime 3 condizioni if si controlla se sono presenti ciascuna delle 3 fasi di gioco, nel caso siano presenti vengono avviate tramite gli intent:

```
Intent i = new Intent(this, VideoActivity.class);
i.putExtra("video", contPrincipale.getVideo());
startActivity(i);
```

Una volta che si ha eseguito tutte e 3 le fasi, scatta l'ultimo else, che verifica la scelta, e avvia quindi il livello successivo.

Nel caso non siano più presenti successivi Nodi, cioè contA e contB siano nullable, si finisce il gioco.

### Salvataggio del livello

L'Activity GameComposer si occupa anche del salvataggio dei livelli.

Per salvataggio si intende il mantenere il percorso già fatto e le scelte già fatte quando eventualmente si decide di chiudere l'applicazione, per poi ripristinare il tutto al riavvio.

Per salvare dei dati in Android, si possono utilizzare molte tecnologie, dai database SQLite, che sono abbastanza complessi nell'utilizzo, alla gestione dei file di scrittura, che richiederebbe permessi alti.

Per ovviare a questo problema abbiamo deciso di utilizzare le SharedPreferences, cioè un file xml preimpostato in Android destinato a contenere informazioni di piccola taglia, quali preferenze ed impostazioni delle applicazioni, da memorizzarsi in locale.

Attenzione, le sharedPreferences sono dei file che non sono dipendenti dalla suddetta applicazione che li utilizza, se si disinstallasse l'applicazione le sharedPreferences rimarrebbero, ciò significa che quando si aggiorna l'applicazione o si installa una nuova versione i salvataggi rimangono.

# Osservazioni

## Problematiche

### GarbageCollector di JAVA

Durante l'esperienza abbiamo avuto modo di confrontarci con problematiche nuove e imparare cose nuove, specialmente sul mondo JAVA.

Nella applicazione è presente un grande problema, che riteniamo vada segnalato, non è un problema funzionale ma di carattere prestazionale.

Passando da un livello all'altro, l'applicazione utilizza sempre più risorse, avendo una crescita nei consumi lineare; questo è dovuto al fatto che nonostante i riferimenti agli elementi utilizzati in precedenza vengano cancellati implicitamente ed esplicitamente, il GarbageCollector non riesce a pulire le celle di memoria e questo comporta un memory leak, cioè una perdita di memoria, per la quale le prestazioni calano.

Questa problematica è data dal linguaggio JAVA e dalla JVM, è risaputo che abbia questo problema; la risoluzione di questo problema va fuori tema con lo svolgimento e la consegna assegnata per questo progetto.

Noi ci teniamo comunque a presentare il problema e ricordare di tenerlo in considerazione in ambito lavorativo.

### Road To Amsterdam e Huawei

Come spiegato nel primo capitolo di introduzione di Android, ciascuna versione di Android rilasciata dai produttori di device differisce. La casa Huawei, quindi anche Honor, produce i propri dispositivi con il firmware proprietario EMUI.

Esiste però una problematica con tale firmware, consiste nel fatto che esso ha una gestione diversa della memoria e dei Canvas, per il quale l'applicazione ha una presente e costante presenza di rallentamento, in ogni fase di gioco.

Questo è un problema grave, ma che va riconosciuto in quanto per risolverlo, bisognerebbe ristrutturare tutto il progetto con l'utilizzo di librerie più adatte ad alta compatibilità tra i dispositivi e non l'utilizzo di Android Studio "Puro" come il nostro progetto ha adoperato.

# Conclusioni

## Riuscita progetto

Tutto sommato il progetto che abbiamo realizzato ci è riuscito bene, ma soprattutto rispetta pienamente quello che abbiamo progettato all'inizio dell'esperienza;

Siamo stati quindi in grado di immaginare e progettare un'applicazione, e realizzarla garantendo il funzionamento ed una buona esperienza di gioco.

Inizialmente volevamo fare più livelli, ma ci siamo resi conto che 6 livelli e 5 finali potevano bastare in quanto abbiamo garantito circa 15 minuti di Gioco Totale, e tenendo conto della rigiocabilità che l'applicazione ha, si arriva quindi a 30 minuti di gioco.

Il nostro obbiettivo era realizzare qualcosa che ci serva ad approfondire le nostre conoscenze didattiche in ambito Android (Cresciute in maniera esponenziale) anche quello di darci una soddisfazione personale nel creare una applicazione che ci veda noi alunni della 5IA come protagonisti.

## Video GamePlay

Il progetto racchiude molte componenti e molte parti che con una relazione non si possono spiegare pienamente, per garantire l'esaminazione della applicazione a tutti abbiamo realizzato due gameplay che indicano due finali diversi del gioco.

[Gameplay Finale Amsterdam High parte 1](#)

[Gameplay Finale Amsterdam High parte 2](#)

[Gameplay Finale Svizzera Schiavi](#)

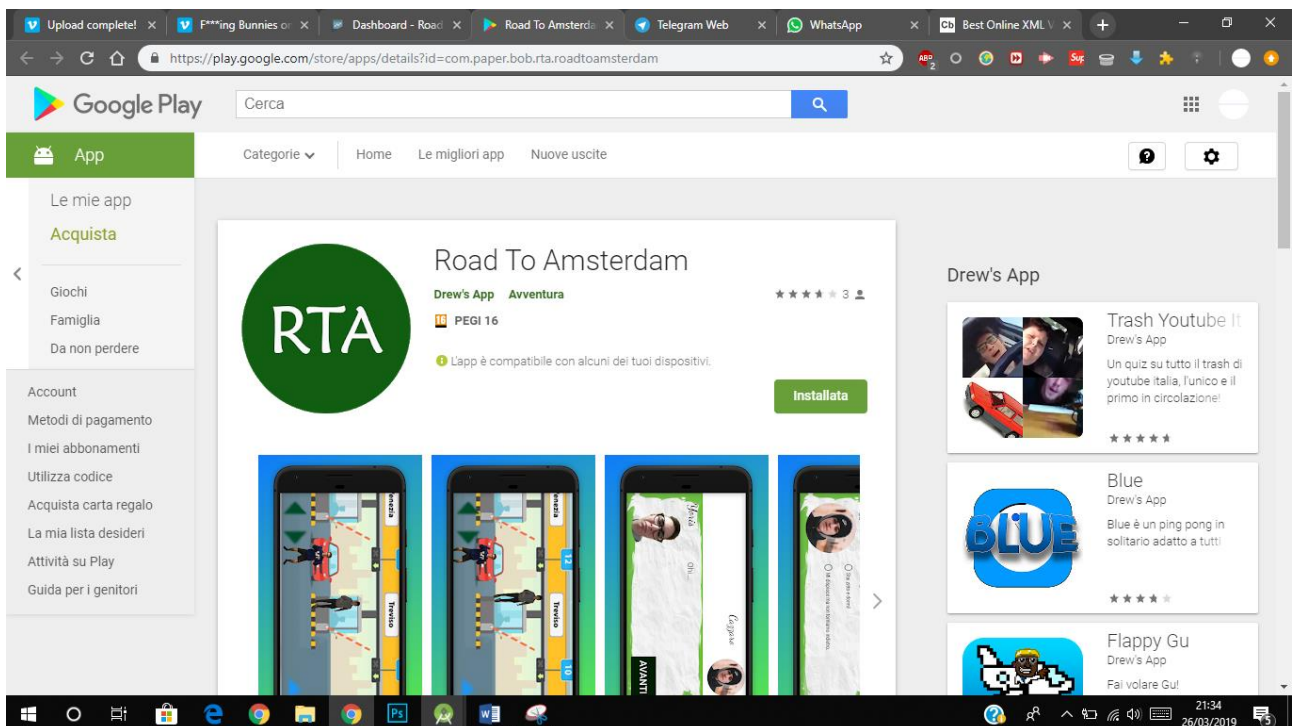
# PlayStore

Abbiamo pensato che per garantire un utilizzo a tutti i membri della classe il modo migliore sarebbe stata la pubblicazione nel playstore.

Per la pubblicazione abbiamo gestito nuovi concetti di Android Studio cioè la produzione di APK tramite chiavi firmate, che garantissero una firma digitale univoca.

Successivamente, dopo l'iscrizione al playstore e la sottoscrizione di un abbonamento di 20 €, abbiamo pubblicato la nostra applicazione rendendola pubblica al mondo.

<https://play.google.com/store/apps/details?id=com.paper.bob.rta.roadtoamsterdam>



# Riassunto Commit GitHub

25/01/2019 INIZIO

CREAZIONE PLATFORM MAIN ACTIVITY, ENGINE GAME E LEVEL CREATOR. CREAZIONE OSTACOLO E GESTIONE DEI TRIGGER

26/01/2019

CREAZIONE DEL GAME LOOP COMPLETO, SISTEMA DI LETTURA DI UN FOGLIO XML (INFO LIVELLO) ATTRAVERSO IL DOM PARSER E IMPLEMENTAZIONE DEL METODO GETOSTACOLI(). AGGIUNTA DELL'ONLY LANDSCAPE E DEL NOTILEBAR

27/01/2019

IMPLEMENTAZIONE SCROLLING DELLO SFONDO E WIDTH E HEIGHT DELLA CLASSE ENGINE GAME. CREAZIONE DELLA CLASSE ANIMATION

28/01/2019

RISOLUZIONE PROBLEMA DELLE GRANDEZZE CHE NON SI ADATTAVANO A QUALSIASI IMMAGINE TRAMITE L'INTRODUZIONE DELL'UNITÀ DI MISURA DP. IMPLEMENTAZIONE DELLE ANIMAZIONI OSTACOLI, IL FILE PNG NON DEVE SUPERARE I 100/200KB

29/01/2019

PERFEZIONAMENTO DEI PERSONAGGI, MIGLIORAMENTO DEGLI OSTACOLI, CONCETTO DELL'OSTACOLO FISICO E FALLIMENTO PERSONAGGIO NOTIFY.

5/02/2019

MODIFICA SISTEMA BACKGROUND E MOVIMENTO OSTACOLI E IMPLEMENTAZIONE DEL MOVIMENTO DEI PERSONAGGI CON IL BACKGROUND

06/02/2019

PICCOLE MODIFICHE GENERALI (FRAMERATE, MOVIMENTO, NOTIFY) E BOZZA SPLASH SCREEN. IMPLEMENTAZIONE CONTROLLER E PLAYER

07/02/2019

FALLIMENTO NELL'IMPLEMENTAZIONE DELLE COLLISIONI

08/02/2019

AGGIUNTA DELLA DOCUMENTAZIONE E PERFEZIONAMENTO DELLE COLLISIONI (RIUSCITA EGREGIAMENTE)

13/02/2019

KEEPSCREEN ABILITATO E MIGLIORAMENTO DATAGRABER

14/02/2019

AGGIUNTA DEL toString() IN OGNI CLASSE TRAMITE PLUGIN E MODIFICA DEL SISTEMA DI CONTROLLER

15/02/2019

IMPLEMENTAZIONE ACCELEROMETRO E SISTEMA SAVEGAME. MODIFICA DELLE DIRECTORY ED ELIMINAZIONE DEL toString() PERCHÉ CAUSAVA PROBLEMI NELL'APERTURA DELL'APP NEI DISPOSITIVI POCO PERFORMANTI

16/02/2019

IMPLEMENTAZIONE SUONI

18/02/2019

BOZZA DEL LAYOUT PER I DIALOGHI

19/02/2019

IMPLEMENTAZIONE DIALOGHI, DEBUG MODE VIEW.

MODIFICA DEL CAMBIO COLORE PER I BTN E SPLASHSCREEN.



BOZZA VIDEOACTIVITY.

20/02/2019

CREAZIONE MAINACTIVITY CON LA QUALE:

- SI AVVIERANNO VIDEO (DI PRESENTAZIONE O DI CONGIUNZIONE TRA I LIVELLI)
- LIVELLI PLATFORM, DA CUI SI PUÒ GIOCARE E ATTIVARE DIALOGHI CON ALTRI PERSONAGGI
- AVVIARE DIRETTAMENTE UN DIALOGO PRIMA DEL GIOCO PLATFORM, NEL CASO IN CUI IL LIVELLO ABBA BISOGNO DI UN PROLOGO

MODIFICA DEL GAMECOMPOSERACTIVITY E DEL ENVIRONMENTCONTAINER

21/02/2019

JAVADOC E CONCLUSIONE GAMEOBJECT (MECCANISMO DELLE SCELTE TRAMITE INTENT)

22/02/2019

IMPLEMENTAZIONE SCROOL TOUCH E COSTRUZIONE DEL PRIMO LIVELLO

23/02/2019

MIGLIORAMENTO DELLE ANIMAZIONI

25/02/2019

IL MAINTHREAD ORA SI CHIAMA GAMETHREAD

26/02/2019

IMPLEMENTAZIONE FISICA DI ACCELERAZIONE E DECELERAZIONE E MENU

2/03/2019

MIGLIORAMENTO DATALEVEL

INTRODUZIONE DEL GAMECOMPOSER DI ALTRI LIVELLI

PROBLEMA CON IL NOTIFY, ORA NON È PIÙ PRESENTE

3/03/2019

RISOLTO BUG SUONO BACKGROUND

5/03/2019

COMPLETATO IL SAVEGAME, RIPRISTINO DEL SAVEGAME E LIVELLO 1

RISOLTO BUG DIALOGO

6/03/2019

INIZIO LIVELLO 2 (AUSTRIA) E IMPLEMENTAZIONE DEI VIDEO DI INTRODUZIONE

09/03/2019

RISOLUZIONE BUG SALVATAGGI E COMPLETAMENTO PLATFORM LIVELLO SVIZZERA

10/03/2019

DOCUMENTAZIONE

11/03/2019

COPYRIGHT

16/03/2019

IMPLEMENTAZIONE DEL MESSAGGIO "CERCA E DIALOGA CON TUTTI I PERSONAGGI " PER PROSEGUIRE AL LIVELLO SUCCESSIVO

20/03/2019

MODIFICA GAMELOOP, RISCONTRATO PROBLEMA CON LE VERSIONI ANDROID CUSTOM EMUI

22/03/2019

COMPLETAMENTO APPLICAZIONE

