Alma Mater Studiorum · Università di Bologna

**facoltà di scienze matematiche, fisiche e naturali**
**Corso di Laurea Triennale in Scienze di Internet**

# Implementazione di giochi per un calcolatore: problemi e soluzioni

**Tesi di Laurea in Programmazione di Internet**

**Relatore:**
**Chiar.mo Prof.**
**Antonio Messina**

**Presentata da:**
**Eugenio Ligabue**

**Seconda Sessione**
**Anno Accademico 2010/2011**

# Indice

# Elenco delle figure

# Capitolo 1

# Introduzione

Vi siete mai posti la domanda su quale sia il primo gioco per calcolatore realizzato al mondo? Circa mezzo secolo fa a Cambridge, nello stato americano del Massachusetts, un ricercatore universitario allora venticinquenne implementò il primo gioco per calcolatore che la storia ricordi. Nel febbraio del 1962 , il ricercatore statunitense *Steve Russel*, dopo circa 200 ore di lavoro diede alla luce *"Space War"*[1]. In questo semplice gioco, erano visibili nello schermo due navicelle spaziali di forma diversa, controllate da altrettanti giocatori. Le navicelle potevano ruotare in senso orario, antiorario e muoversi accelerando. Lo scopo del gioco, era quello di eliminare la navicella dell'avversario sparando un missile nell'intento di colpirla. Il calcolatore in possesso di Steve Russel era un PDP-1[1] dell'azienda statunitense DEC[2] . Le dimensioni erano pari a tre frigoriferi ed il prezzo si aggirava intorno ai 120.000 dollari (circa 880.000 attuali). Ne furono venduti solamente 50 esemplari destinati a centri universitari di ricerca e grandi aziende[2][3].

Figura 1.1: Steve Russel ed il PDP-1 da www.wikipedia.org/wiki/PDP-1

[1] Acronimo di Programmed Data Processor-1
[2] Acronimo di Digital Equipment Corporation

A cinquant'anni di distanza la situazione è ben diversa. Al giorno d'oggi sono disponibili tantissimi videogiochi ed i moderni calcolatori non sono più posseduti solo da una nicchia di utilizzatori. Il rapporto annuale sullo stato dell'industria videoludica italiana stilato dall'*AESVI* [3]nel 2010, riporta che sei famiglie su dieci possiedono almeno un calcolatore. Il giro d'affari italiano dei giochi per calcolatore invece è superiore ai 40 milioni di euro annuali[4].

In questa tesi, è stato progettato e sviluppato un gioco 2D [4] per calcolatore in linguaggio Java[5] chiamato *BarInvaders*. Saranno affrontate le problematiche legate alla realizzazione del progetto e verranno illustrate alcune scelte implementative che si sono rivelate molto utili. Il primo capitolo, affronta il tema dei giochi per calcolatore 2D in generale, mentre i capitoli successivi sono dedicati alla progettazione dell'applicazione che è stata implementata. Il capitolo due è dedicato all'analisi dei requisiti, mentre nel capitolo tre sono riportate le specifiche di progetto e la metodologia seguita per l'implementazione del gioco. Nel rimanente capitolo quattro, sono riportate alcune immagini del programma BarInvaders durante l'esecuzione.

## 1.1 Scenario del progetto BarInvaders

L'applicazione che è stata realizzata in questa tesi, è un gioco per calcolatore. L'ambientazione ricorda quella di un bar al cui interno abbiamo un barista, alcuni clienti ed un bancone. Una musica di sottofondo accompagna il giocatore per tutta la durata della partita. Lo scopo del gioco, è quello di difendere il bancone dall'invasione dei clienti che barcollanti si avvicinano ad esso. Per scongiurare l'invasione, il barista ha la possibilità di lanciare in direzione dei clienti una bottiglia, nell'intento di colpire uno di loro. Se il bersaglio viene colpito, quel cliente viene eliminato e l'utente guadagna dei punti. La difficoltà risiede nel fatto che i clienti non solo barcollano e quindi

[3] Acronimo di Associazione Editori Software Videoludico Italiana - www.aesvi.it

[4] Un gioco 2D utilizza immagini bidimensionali aventi una certa larghezza ed una certa lunghezza ma sono prive della terza dimensione, la profondità

[5] Java è un linguaggio di programmazione orientato agli oggetti - www.java.com

cambiano spesso direzione, ma se vengono eliminati, vengono sostituiti da altri più veloci dei precedenti. L'utente ha a disposizione un certo numero di "vite" ed ogni volta che si verifica uno di questi eventi:

- un cliente raggiunge il bancone
- un cliente raggiunge il barista

all'utente viene sottratta una "vita" se non sono esaurite. In quel caso la partita termina ed i migliori risultati degli utenti entrano a far parte di una classifica. Il barista è controllato dall'utente mentre il calcolatore è responsabile del movimento dei clienti nella schermata di gioco. L'applicazione perciò non è multigiocatore e non prevede la possibilità di giocarci in rete. L'applicazione è "standalone" cioè in grado di avviarsi senza necessitare di nessuna procedura di installazione.

## 1.2 Cos'è un gioco ?

Per poter classificare un'applicazione per calcolatore come un gioco, occorre che abbia delle caratteristiche che la differenzino dagli altri programmi. Innanzitutto, lo scopo per il quale l'utente decide di applicarvisi, è la ricerca del divertimento. Cliccando col mouse sull'icona di un gioco, l'utente è perfettamente cosciente che non realizzerà qualcosa di utile, ma riceverà soddisfazione dall'attività ludica. Una seconda caratteristica distintiva è il totale distacco dalla realtà dell'applicazione. Spesso nei giochi, l'utente deve controllare i movimenti di un personaggio in un luogo che è puro frutto della fantasia. Tutto ciò che appare nel monitor non esiste veramente, ed ha un periodo di vita circoscritto alla permanenza dell'utente al calcolatore. Una terza caratteristica distintiva rispetto alle altre applicazioni, è l'imprevedibilità del risultato dell'interazione. Spesso potremmo trovarci a sfidare il calcolatore in veste di secondo giocatore durante una partita. L'esito dell'interazione è ignoto, perchè dipende in questo caso dal comportamento di

entrambi i giocatori. L'ultima caratteristica importante di un gioco, è la presenza di regole ben precise che possono essere diverse da quelle che seguiamo nella vita di tutti i giorni.

## Com'è strutturato un gioco?

La parte fondamentale di un gioco 2D[6], è sicuramente quella legata alla gestione dell'animazione. Occorre simulare il movimento di ogni elemento che vogliamo animare nello schermo. Oltre a questo, occorre che si muovano in maniera coerente con i comandi del giocatore. Se l'utente ad esempio volesse spostare una pallina da tennis verso destra, l'applicazione dovrà simulare esattamente questa azione nello schermo. La tecnica utilizzata è la stessa utilizzata nella realizzazione di un cartone animato, nella quale l'animazione è un effetto ottico, dato dallo scorrimento veloce di una serie di immagini. L'animazione è perciò divisa in tre fasi distinte, che vengono continuamente eseguite in sequenza all'interno di un ciclo:

- Update
- Render
- Ritardo

Nella fase di Update, si aggiornano le posizioni degli elementi coinvolti nell'animazione. Tornando all'esempio della pallina da tennis, il programmatore dovrà decidere la quantità di spostamento da effettuare ad ogni fase di Update in una direzione o nell'altra. Se l'utente non la volesse spostare, la pallina deve rimanere nella propria posizione. Nella fase di Render, andremo a disegnare nello schermo del calcolatore tutti gli elementi coinvolti nell'animazione, nella nuova posizione. Nell'ultima fase, si introduce un ritardo nell'esecuzione dell'applicazione altrimenti la grafica avrebbe un aggiornamento troppo veloce e l'utente non sarebbe in grado di utilizzare il videogioco. Variando il ritardo, è possibile realizzare lo spostamento più rapido dei soggetti. Se diminuiamo il ritardo, la grafica sarà aggiornata

[6] Un gioco 2D utilizza immagini bidimensionali aventi una certa larghezza ed una certa lunghezza ma sono prive della terza dimensione, la profondità.

più frequentemente e questo darà la sensazione di una maggiore velocità dei soggetti nello spostamento.

Nel capitolo 3 verrà introdotta una tecnica di gestione dell'animazione chiamata *double buffering*.

## 1.3 Problemi di realizzazione : prestazioni e risorse

Due riflessioni importanti, che precedono la realizzazione di un gioco dovrebbero essere la gestione delle risorse impiegate e le prestazioni fornite all'utente. Il programmatore ha un ruolo fondamentale in questo, in quanto i linguaggi di programmazione attuali, forniscono tutte le tecnologie affinché si possano implementare applicazioni ad elevate prestazioni con un'utilizzo oculato delle risorse a disposizione. Una programmazione efficiente, è fondamentale quando le risorse sono scarse e dobbiamo garantire il massimo delle prestazioni. Vediamo ora qualche tecnologia utile al nostro scopo, facendo riferimento al linguaggio Java.

La prima tecnologia che occorre nominare è la tecnologia *HotSpot*[6, 7] che è parte integrante della JVM[7]. In seguito alla compilazione di un file con estensione ".java", viene generato dal nostro IDE[8] un file con estensione ".class" contenente il bytecode[9]. Per essere eseguito, il bytecode deve essere tradotto in linguaggio macchina dalla JVM installata nel nostro calcolatore. A tale scopo viene utilizzato un compilatore JIT [10] che traduce il bytecode nel linguaggio macchina durante la fase di esecuzione dell'applicazione, nell'istante precedente all'utilizzo di quel codice[8]. La JVM grazie alla tecnologia *HotSpot*, è in grado di ottimizzare l'esecuzione delle parti di

[7] Acronimo di Java Virtual Machine - componente software della piattaforma Java che esegue i programmi scritti in bytecode

[8] Acronimo di Integrated Development Environment - ambiente software per lo sviluppo di applicazioni

[9] Linguaggio intermedio tra il codice sorgente ed il linguaggio macchina del calcolatore

[10] Acronimo di Just-In-Time

codice che al run-time vengono utilizzate più frequentemente *(dynamic optimization)*. Vale anche il discorso contrario, in quanto è prevista la revoca dell'ottimizzazione ad una parte di codice se non sussistono più le condizioni per la quale era stata ottimizzata *(dynamic deoptimization )*. Cerchiamo ora di capire come funzioni l'ottimizzazione, analizzando l'esempio riportato nel tutorial *HotSpot* di Sun[11] che ricorre ad una tecnica di compilazione chiamata *inlining*[9].

Immaginiamo che in un programma, tra le varie classi ce ne sia una denominata A, contenente un metodo foo() come nel codice Java seguente.

```
class A
{
 final int foo()
 {
  return 3;
 }
}
```

Una qualsiasi JVM, all'atto della compilazione del programma in codice macchina, scambierebbe ogni chiamata al metodo foo() col valore intero 3. La parola chiave "final" garantisce infatti che nessuna sottoclasse possa sovrascrivere il metodo foo(). L'intento della JVM è perciò quello di evitare dispendiose chiamate ad un metodo che restituirà sempre il valore costante 3. Con la tecnologia *HotSpot* ci spingiamo oltre. Non occorre inserire la parola chiave "final" perchè avvenga l'ottimizzazione, ma viene eseguita automaticamente.

```
class B
{
 int foo()
 {
  return 3;
 }
```

[11]Sun Microsystems è un'azienda produttrice di software ideatrice del linguaggio Java

```
}
```

Non ci saranno problemi fino a quando, durante l'esecuzione, non verrà caricata una sottoclasse C che estende la classe B sovrascrivendo il metodo foo().

```
class C extends B
{
  int foo()
  {
   return 6;
  }
}
```

Ogni variabile di tipo B nel programma, potrebbe ora riferirsi a C oppure a B chiamando il metodo foo(). Quando siamo in presenza di questi casi che creano confusione, non viene più effettuata l'ottimizzazione per questo metodo.

Per fornire le migliori prestazioni possibili, la tecnologia *HotSpot* offre inoltre la possibilità di compilare in modo differente applicazioni con funzioni client o server. Le prime necessitano di ottimizzare il tempo di avvio e l'utilizzo della memoria, per le seconde invece è importante la velocità di esecuzione. Vediamo ora alcune delle tecniche di ottimizzazione utilizzate durante la compilazione:

- Deep inlining : applicazione della tecnica "*inlining*" in modo approfondito, questo migliora le prestazioni in termini di velocità di esecuzione
- Fast checkcast : vengono velocizzati i controlli dei "*casting*" tra oggetti. Il "*casting*" è una conversione di un oggetto Java da una tipologia ad un'altra
- Range check elimination : prima di ogni accesso ad un array[12] di dati, viene controllato se l'indice del dato a cui si vuole accedere non superi

[12]Struttura dati contenente una molteplicità di valori della stessa tipologia

la lunghezza dell'array stesso. Il compilatore elimina questo controllo, tenendo traccia della lunghezza dell'array quando gli accessi sono ripetuti nel tempo

- Loop unrolling: se possibile, vengono uniti due o più cicli di istruzioni per creare un unico ciclo più grande. L'effetto che si ottiene è una diminuzione del numero di iterazioni

Per quanto riguarda la memoria, l'introduzione di *HotSpot* ha portato due miglioramenti. Innanzitutto, la JVM ha la possibilità di avere un puntatore diretto alle classi degli oggetti che si stanno utilizzando nell'applicazione, eliminando quelle che vengono chiamate "maniglie"(handle )[13]. Togliamo un livello inutile di riferimento, portando ad uno solo il numero di accessi alla memoria per un riferimento ad un oggetto. Il *garbage collector* ha il compito di mantenere sempre aggiornati i puntatori diretti anche in seguito ad una pulizia della memoria. Vi è stata inoltre una riduzione dell'occupazione del descrittore di un oggetto(Object Header) in memoria da tre a due spazi(word). In questa maniera si riduce l'occupazione dell'heap[14] dell'applicazione. Per quanto riguarda i processi ed i thread, la gestione è maggiormente supportata dal sistema operativo del calcolatore in cui l'applicazione è in esecuzione. Questo permette una maggiore velocità degli stessi, in quanto è possibile utilizzare strumenti che si adattano maggiormente alla memoria ed al processore di quel calcolatore specifico. L'ultimo argomento che prendiamo in considerazione riguardo alla tecnologia *HotSpot*, è il *garbage collector* letteralmente il "raccoglitore di rifiuti". Il ruolo di questo thread "spazzino", è di liberare la memoria centrale da oggetti che non vengono più referenziati nell'applicazione e quindi che occupano spazio inutilmente. Vi sono linguaggi di programmazione, che lasciano che sia il programmatore a liberare la memoria (C e C++), mentre il Java prevede un garbage collector

[13] Il linguaggio Java non permette la gestione esplicita dei puntatori agli oggetti. Occorre utilizzare quelle che si chiamano maniglie (handle)

[14] Parte della memoria gestita dell'applicazione nella quale vengono allocate le istanze degli oggetti creati.

integrato nella JVM del tutto indipendente. Questa scelta è stata fatta in quanto l'utente potrebbe rilasciare inavvertitamente della memoria ancora in uso, causando degli errori in esecuzione quando un programma cerca di interagire con un oggetto non più presente in memoria. Il lato opposto della medaglia, sta nel fatto che potrebbe non essere mai rilasciata la memoria, causando la terminazione ed il blocco dell'applicazione. Ciò che è importante sapere, è che esiste un metodo per invocare il garbage collector,chiamato *System.gc()*[15]. Il programmatore può invocare il garbage collector nell'istante che ritiene opportuno, per esempio appena prima dell'esecuzione di codice in cui si farà un'utilizzo molto intenso di memoria. Il garbage collector introdotto dalla tecnologia *HotSpot*, è molto efficente in quanto tutti gli oggetti in memoria possono essere spostati e compattati, consentendo di eliminare la frammentazione e aumentando la località dei dati che sono così più facilmente accessibili . Molti degli oggetti creati da un programmatore in una applicazione, hanno normalmente una vita corta e sono invece pochi quelli che hanno una vita pari all'intero periodo di esecuzione dell'applicazione. *HotSpot* ha perciò appositamente introdotto due tipologie di garbage collector: *young* and *old*. Se abbiamo un calcolatore multiprocessore[16] possiamo utilizzare un garbage collector multi-thread chiamato *Parallel Young Generation Collector*. Vengono migliorate le prestazioni della nostra applicazione, perchè il thread "spazzino" non rallenta l'esecuzione come accadrebbe in un sistema monoprocessore. Oltretutto in seguito all'eliminazione di un oggetto dalla memoria, si cerca di mantenere al massimo la correlazione degli oggetti rimanenti durante lo spostamento. Il garbage collector *old*, utilizzata una tecnica chiamata *Mark-compact old object collector* che permette una gestione efficente dello spazio vuoto di memoria lasciato dopo l'eliminazione di un oggetto in memoria. Viene effettuato un compattamento dell'intero heap in modo da liberare al massimo la memoria.

Il terzo strumento che analizziamo, è il cosiddetto *Java Profiler*. Questo

[15] Metodo del package System della libreria di Java

[16] Calcolatore nel quale sono installati due o più processori operanti in parallelo

termine è utilizzato per identificare tutte quelle applicazioni gratuite o a pagamento che permettono al programmatore di monitorare le performance della propria applicazione.Una di queste, forse la più importante, è chiamata *JProfiler*[11]. Grazie ad esso, possiamo conoscere in quanto tempo viene eseguito un metodo, quanta memoria è occupata della JVM, quanto spazio occupano gli oggetti che abbiamo istanziato, quali parti di codice utilizzano eccessivamente la memoria e tante altre utili statistiche.

Uno strumento gratuito per controllare le performance e l'utilizzo di risorse di un'applicazione, è fornito durante l'installazione di Java nella cartella "bin" dove abbiamo installato il JDK. Si tratta di *JConsole* [12], uno strumento che permette di controllare le prestazioni della JVM, lo stato dei thread in esecuzione, l'utilizzo della memoria ed il garbage collector. Al run-time possiamo praticamente conoscere come il nostro applicativo sta utilizzando la JVM . Questa applicazione sgrava il programmatore dall'utilizzo diretto delle classi del package *java.lang.management* che è stato introdotto dalla versione 5 di Java per permettere al programmatore di interrogare la JVM in merito alle risorse utilizzate. Le classi di questa libreria sono:

- ClassLoadingMXBean: per il monitoraggio del caricamento delle classi
- CompilationMXBean : per il monitoraggio della compilazione
- MemoryMXBean : per il monitoraggio della memoria
- ThreadMXBean : per il monitoraggio dei thread
- RuntimeMXBean : per il monitoraggio del runtime
- OperatingSystemMXBean : per il monitoraggio del sistema operativo
- GarbageCollectorMXBean : per il monitoraggio del garbage collector
- MemoryManagerMXBean : per il monitoraggio e la gestione della memoria

Un altro argomento importante, soprattutto per la realizzazione di giochi, è la scelta tra l'utilizzo di oggetti grafici del package *java.awt* o *javax.swing*. La libreria *Swing* di Java, è stata introdotta successivamente all' *Awt*, in quanto vi era la necessità di oggetti grafici più sofisticati che fossero disegnati direttamente da Java e quindi indipendenti dal calcolatore utilizzato. Quest'ultima caratteristica, introduce perciò un livello supplementare di gestione sopra al sistema operativo, ed è per questo motivo che nei giochi, dove servono prestazioni efficienti, è preferibile l'utilizzo della libreria *Awt*.

Per quel che riguarda la grafica, occorre poi parlare della modalità *FSEM* [17] di Java [16]. Utilizzando questa modalità, disabilitiamo il classico sistema di visualizzazione a finestra, per disegnare direttamente nello schermo intero. Questa disabilitazione deve essere eseguita direttamente dall'applicazione. Utilizzando questa modalità, è possibile controllare la profondità e la grandezza di ogni bit dello schermo, oltre alla dimensione dello stesso. Le classi Java coinvolte sono le seguenti:

- java.awt.GraphicsEnvironment
- java.awt.GraphicsDevice
- java.awt.DisplayMode

Vediamo ora come utilizzarle. La modalità *FSEM* può essere abilitata chiamando innanzitutto il metodo *getDefaultScreenDevice()* della classe GraphicsEnvironment [17]. Esso restituisce un oggetto della classe GraphicsDevice[18], che nel nostro caso rappresenta lo schermo del computer. I metodi più importanti di questa classe sono i seguenti:

- setFullScreenWindow (Window w) : serve per entrare nella modalità *FSEM*
- isFullScreenSupported() : metodo che restituisce il valore booleano "true" se il GraphicsDevice supporta la modalità *FSEM*

[17] Acronimo di Full Screen Exclusive Mode

- setDisplayMode(DisplayMode dm ) : serve a settare la modalità di visualizzazione del GraphicsDevice

- getDisplayMode() : restituisce l'attuale modalità di visualizzazione per il GraphicsDevice

- isDisplayChangeSupported() : restituisce il valore booleano "true", solamente se è permesso effettuare cambiamenti sulla visualizzazione del GraphicsDevice. Non tutti i sistemi operativi lo permettono

- getConfigurations() : restituisce tutti gli oggetti della classe GraphicsConfiguration [19] associati al GraphicsDevice. Ogni oggetto di questa classe rappresenta una delle risoluzioni possibili per lo schermo del computer e quindi utilizzabili dalla nostra applicazione.

Un oggetto della classe DisplayMode[20] contiene informazioni importanti quali:

- larghezza e altezza del monitor espresso in pixel (Height,Width)

- numero di bit per pixel del monitor (bit depth)

- la frequenza di aggiornamento del monitor(RefreshRate)

La modifica di questi valori deve essere effettuata solamente in modalità *FSEM*. L'importanza di tutto questo è cruciale soprattutto per i giochi. Se le immagini visualizzate dall'applicazione, hanno la stesso *bit depth* dello schermo, l'esecuzione potrà essere eseguita molto rapidamente.

## 1.4 Videogiochi nei dispositivi mobili

Il 3 aprile del 1973 lo statunitense *Martin Cooper* fece la prima telefonata con un telefono cellulare nella storia dell'uomo[21].

Figura 1.2: Martin Cooper da www.2space.net

Negli anni il cellulare si è notevolmente evoluto, e la sua originaria funzione di effettuare chiamate è passata progressivamente in secondo piano. Gli attuali apparecchi telefonici portatili ora possono funzionare come un navigatore satellitare, accedere alla rete internet, ricevere mail ed effettuare videochiamate. Inoltre l'utente può installarvi tantissime applicazioni tra cui i giochi. Il cellulare ha praticamente le stesse funzioni di un calcolatore racchiuse nella grandezza di un pacchetto di fazzoletti. Questo tipo di apparecchi ha preso il nome di *smartphone*[22]. Ne esistono tantissimi modelli ed utilizzano diversi sistemi operativi. I più importanti di questi sono:

- Android sviluppato da Google Inc.
- Apple iOS sviluppato da Apple Inc.
- Blackberry OS sviluppato da Research In Motion Limited (RIM)
- Windows Mobile sviluppato da Microsoft Corporation
- Symbian OS sviluppato da Nokia

Millenium Media, uno dei più famosi network pubblicitari americani per dispositivi mobili, ha recentemente pubblicato alcuni dati riguardanti i principali produttori di smartphone e la diffusione dei sistemi operativi citati

precedentemente[23]. I dati sono stati raccolti tenendo traccia della tipologia di dispositivi mobili utilizzati dagli utenti per visualizzare gli spazi pubblicitari gestiti dall'azienda. *Ranked by impressions,* significa appunto classificazione in base al numero di visualizzazioni (impressions). Dall'analisi è risultato che nel 54% dei telefoni era installato il sistema operativo Android contro il 28% di Apple iOS. Appena un anno e mezzo prima, a marzo 2010, Android aveva totalizzato un deludente 3%.

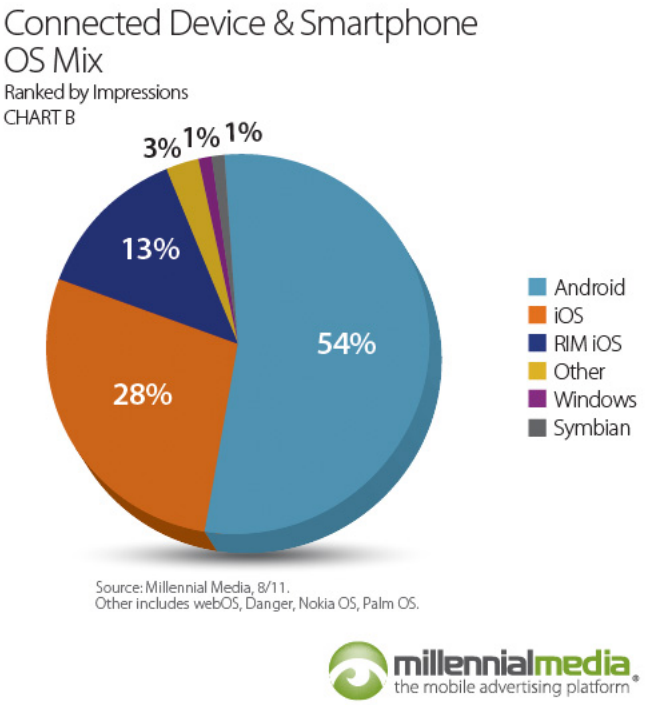
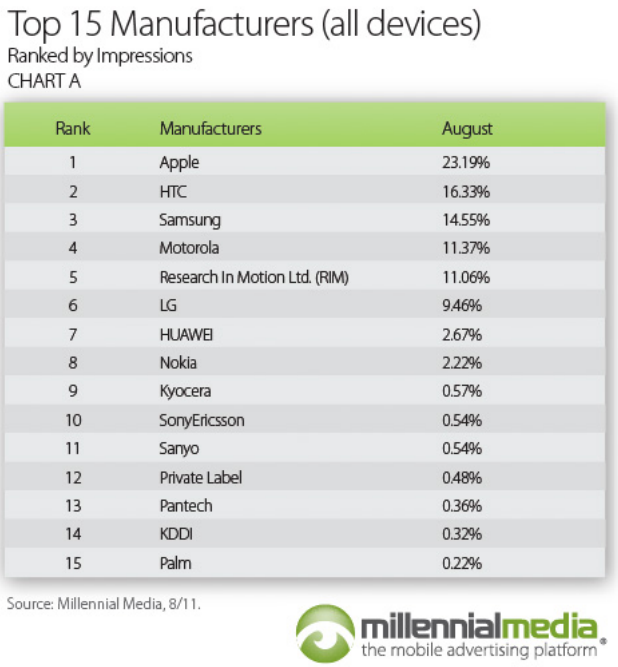



Figura 1.3: Percentuale di utilizzo dei sistemi operativi mobile
da www.millennialmedia.com

Per quanto riguarda i produttori di dispositivi mobili utilizzati nella visualizzazione della pubblicità, Apple è risultato il detentore del primato col 23% degli smartphone.



Top 15 Manufacturers (all devices)
Ranked by Impressions
CHART A

| Rank | Manufacturers | August |
|---|---|---|
| 1 | Apple | 23.19% |
| 2 | HTC | 16.33% |
| 3 | Samsung | 14.55% |
| 4 | Motorola | 11.37% |
| 5 | Research In Motion Ltd. (RIM) | 11.06% |
| 6 | LG | 9.46% |
| 7 | HUAWEI | 2.67% |
| 8 | Nokia | 2.22% |
| 9 | Kyocera | 0.57% |
| 10 | SonyEricsson | 0.54% |
| 11 | Sanyo | 0.54% |
| 12 | Private Label | 0.48% |
| 13 | Pantech | 0.36% |
| 14 | KDDI | 0.32% |
| 15 | Palm | 0.22% |

Source: Millennial Media, 8/11.



Figura 1.4: Produttori di dispositivi mobili da www.millennialmedia.com

I dati mostrano chiaramente che al momento, abbiamo una vera e propria guerra di mercato tra due sistemi operativi dominanti, Android ed iOS. Apple dal canto suo gode di una grande produzione, ma il sistema operativo iOS è installato solamente nei propri prodotti. Google invece può contare su una folta schiera di produttori che hanno scelto Android per i propri smartphone e la diffusione è in rapidissima ascesa. Il sistema operativo Android, ha perciò tutte le carte in regola per imporsi nel mercato. Inoltre, la caratteristica che utilizzi Java come linguaggio di riferimento, lo rende ideale per chi vuole cimentarsi nell'implementazione di giochi per smartphone. Alla stregua di Java, per poter sviluppare applicazioni Android, è necessario munirsi del *software development kit(SDK)* che include tra l'altro le librerie, un emulatore di dispositivo mobile ed alcuni progetti esemplificativi[24]. Un'applicazione, come ad esempio un videogioco che ha un'iterazione con l'utente, è detta attività *(Activity)*. Le operazioni che il sistema operativo svolge in supporto alle Activity, vengono chiamate *servizi (Service)*. Un accesso all'hardware del

telefono per esempio. Come detto prima, gli smartphone, che hanno adottato Android come sistema operativo, sono vari e diversi tra loro. Sviluppare ad esempio un'applicazione che si adatti graficamente ad ogni tipo di schermo, potrebbe essere un bel rompicapo per un programmatore. Android ha introdotto un sistema intelligente per evitare questo, adottando il concetto di classe *Frammento(fragment)*. La parte grafica è indipendente dalle altre parti di un'applicazione, svincolando così il programmatore da dover creare una versione diversa dell'applicazione per ogni apparecchio mobile. Utilizzando la stessa filosofia di Java, anche per Android abbiamo una Virtual Machine detta *Dalvik Virtual Machine (DVM)*. La compilazione di codice Java produce un file con estensione ".java" ed un altro invece ". xml". In quest'ultimo, sono contenute le caratteristiche della visualizzazione scelta, ad esempio il colore di sfondo dello schermo. Per completare un'applicazione, alla stregua di un archivio ".jar", dobbiamo creare un file *manifest*, nel quale inseriremo un'apposita descrizione dell'applicazione. Questi tre elementi devono poi essere inseriti in un file con estensione *".apk"* per poter essere installati in un dispositivo mobile Android.

# Capitolo 2

# Analisi dei requisiti

In questo capitolo, si preparano le fondamenta della progettazione e dell'implementazione del progetto BarInvaders. Occorre focalizzare l'attenzione sulle funzionalità della nostra applicazione e sul ruolo che il sistema ricopre cercando di soddisfare i bisogni dell'utente. Il punto di partenza è l'individuazione dei requisiti funzionali del sistema utili alla costruzione dei casi d'uso .

## 2.1 Casi D'uso

Nel paragrafo 1.1 abbiamo identificato l'ambito del progetto *BarInvaders*. Occorre che da quelle poche righe si focalizzi l'attenzione sui compiti che la nostra applicazione deve svolgere. Chiamiamo *requisiti funzionali,* quei requisiti che specificano quali funzioni il sistema deve fornire per soddisfare i bisogni dell'utente. Possono essere esplicitati sia a livello generale che in dettaglio. Vediamo ora quali sono i requisiti funzionali dell'applicazione che vogliamo realizzare:

- il sistema deve controllare la collisione dei clienti con la bottiglia
- il sistema deve controllare la collisione dei clienti con il bancone
- il sistema deve controllare la collisione dei clienti con il barista
- il sistema deve permettere all'utente di spostare il barista nella schermata
- il sistema deve permettere all'utente di gestire il movimento del barista ed il lancio di una bottiglia
- il sistema deve gestire l'animazione dei clienti nella schermata
- il sistema deve sostituire i clienti colpiti con altri clienti
- il sistema deve permettere all'utente di autenticarsi
- il sistema deve permettere all'utente di visualizzare la classifica

Identifichiamo ora i requisiti non funzionali, che definiscono proprietà del sistema ed eventuali restrizioni (ad es.adattabilità, tempo di risposta, richieste di memoria, capacità dei device di I/O, rappresentazioni di sistema ecc.). Essi non sono collegati direttamente con le funzioni implementate e fornite dal sistema, ma piuttosto alle modalità operative e di gestione definendo vincoli sullo sviluppo del sistema. Vediamo quali sono i requisiti non funzionali del nostro progetto:

- l'ambientazione deve essere quella di un bar
- occorre che vi sia una musica in sottofondo
- l'utente guadagna dei punti per ogni eliminazione
- la velocità di discesa dei clienti aumenta dopo ogni eliminazione
- ogni utente dispone di un certo numero di vite
- solo i migliori risultati entrano in classifica
- l'applicazione non è multigiocatore
- l'applicazione non è di rete
- l'applicazione deve essere standalone

Partendo dai requisiti funzionali, è possibile costruire quelli che in ingegneria del software vengono chiamati casi d'uso[27] . Sono una descrizione in forma narrativa delle interazioni tra l'utente ed il Sistema. Consentono di valutare ogni requisito incentrandosi sugli attori che interagiscono col sistema, valutandone tutti i possibili comportamenti. Sono importanti perchè quando si passerà alla progettazione ed all'implementazione, serviranno da traccia per riportare in forma di codice Java la sequenza di azioni così come è stata analizzata in questa fase. Vediamo in dettaglio quali sono i casi d'uso individuati:

- animazione
- update posizioni
- controllo collisioni
- fine partita
- nuovo livello
- lancio della bottiglia

- movimento barista
- visualizzazione classifica
- autenticazione

Ora analizziamo in dettaglio ogni caso d'uso utilizzando un linguaggio di modellazione dell'ingegneria del software chiamato UML [1]. Nella figura 2.1 è mostrata la tipologia di notazione UML testuale per i casi d'uso.

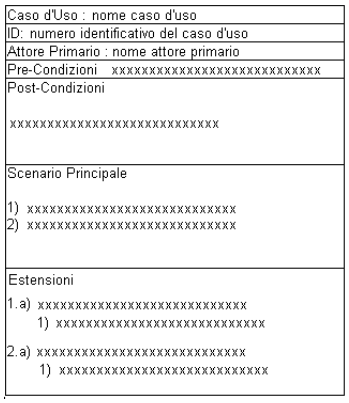



Figura 2.1: Rappresentazione UML testuale dei casi d'uso

La figura 2.1 mostra la rappresentazione UML testuale per un caso d'uso. Sono racchiuse diverse informazioni in questo semplice schema:

- il nome del caso d'uso
- il numero identificativo del caso d'uso
- l'attore primario : indica chi svolge il caso d'uso
- pre condizioni : condizioni che devono essere vere prima che il caso d'uso possa iniziare
- post condizioni : condizioni che devono essere vere una volta terminato il caso d'uso

[1] Acronimo di Unified Modeling Language - http://it.wikipedia.org/wiki/Unified_Modeling_Language

- scenario principale di successo : indica le azioni che vengono svolte nel caso d'uso dagli attori. Una di queste azioni potrebbe essere l'esecuzione di un'ulteriore caso d'uso.
- estensioni : sono delle eccezioni che si possono verificare nelle azioni dello scenario principale di successo

Utilizzeremo nei prossimi paragrafi questa rappresentazione per modellare i casi d'uso del progetto BarInvaders.

### Caso d'uso : Animazione

Questo è il caso d'uso più importante di tutti. Il cuore di un gioco come quello che stiamo progettando, è sicuramente rappresentato dalla gestione dell'animazione. Vengono inizialmente controllate le collisioni dei clienti con la bottiglia, il barista ed il bancone. Viene eseguito a tal proposito il caso d'uso "Controllo Collisioni". Successivamente vengono aggiornate le posizioni dei clienti e della bottiglia eseguendo il caso d'uso "Update Posizioni". L'ultimo passo che resta da fare, è disegnare i clienti, il barista e la bottiglia sullo schermo.

| Caso d'Uso : Animazione |
|---|
| ID: 1 |
| Attore Primario : Sistema |
| Pre-Condizioni |
| Post-Condizioni |
| Scenario Principale<br><br>1) Il Sistema esegue Controllo Collisioni<br>2) Il Sistema esegue Update Posizioni<br>3) Il Sistema disegna sullo schermo i clienti, il barista e la bottiglia |
| Estensioni<br><br>1.a) l'Utente ha terminato le "vite"<br>1)il Sistema esegue Fine Partita<br><br>3.a) il cliente è stato colpito precedentemente da bottiglia<br>1) il Sistema non disegna quel cliente<br><br>3.b) nessuna bottiglia in gioco<br>1) il Sistema non disegna la bottiglia<br><br>3.c) l'Utente ha eliminato tutti i clienti.<br>1)Il Sistema esegue Nuovo Livello |

Figura 2.2: Caso d'uso Animazione

### Caso d'uso : Update Posizioni

La tecnica che adottiamo per l'animazione, è la stessa utilizzata nella realizzazione di un cartone animato, sfruttiamo cioè un effetto ottico, dovuto dallo scorrimento veloce di una serie di immagini in successione. Nello scorrimento, le figure nelle immagini cambiano posizione simulando un movimento. Alla stessa maniera, in questo caso d'uso aggiorniamo la posizione della bottiglia e dei clienti ad ogni aggiornamento grafico.

| |
|---|
| Caso d'Uso : Update Posizioni |
| ID: 2 |
| Attore Primario : Sistema |
| Pre-Condizioni |
| Post-Condizioni<br><br>Sono state aggiornate tutte le posizioni. |
| Scenario Principale<br><br>1) Il Sistema aggiorna la posizione della bottiglia<br>2) Il Sistema aggiorna la posizione dei clienti |
| Estensioni<br><br>1.a) il barista non ha lanciato nessuna bottiglia<br>1) il Sistema non aggiorna la posizione della bottiglia<br><br>2.a) il cliente è stato colpito precedentemente da bottiglia<br>1) il Sistema non aggiorna la posizione di quel cliente<br><br>2.b) il cliente ha raggiunto il fondo della schermata<br>1) il Sistema non aggiorna la posizione di quel cliente<br><br>2.c)il cliente è entrato in collisione col barista<br>1) il Sistema non aggiorna la posizione di quel cliente |

Figura 2.3: Caso d'uso Update Posizioni

### Caso d'uso : Controllo Collisioni

Il sistema controlla le collisioni dei clienti con la bottiglia, il barista ed il bancone.

| Caso d'Uso : Controllo Collisioni |
|---|
| ID: 3 |
| Attore Primario : Sistema |
| Pre-Condizioni |
| Post-Condizioni<br><br>Sono state controllate tutte le collisioni. |
| Scenario Principale<br><br>1) Il Sistema controlla la collisione tra clienti e barista<br>2) Il Sistema controlla la collisione tra clienti e bancone<br>3) Il Sistema controlla la collisione tra clienti e bottiglia |
| Estensioni<br><br>1.a) il cliente è stato colpito precedentemente da bottiglia<br>1) il Sistema non controlla la collisione tra il cliente e barista<br><br>2.a) il cliente è stato colpito precedentemente da bottiglia<br>1) il Sistema non controlla la collisione tra il cliente e bancone<br><br>3.a) nessuna bottiglia in gioco<br>1) il Sistema non controlla la collisione tra i clienti e la bottiglia<br><br>3.b) il cliente è stato colpito precedentemente da bottiglia<br>1) il Sistema non controlla la collisione tra il cliente e la bottiglia |

Figura 2.4: Caso d'uso Controllo Collisioni

## Caso d'uso : Fine Partita

Quando l'utente termina le "vite" a sua disposizione, il sistema aggiorna la classifica ed esegue il caso d'uso "Autenticazione".

| Caso d'Uso : Fine Partita |
|---|
| ID: 4 |
| Attore Primario : Sistema |
| Pre-Condizioni<br><br>L'Utente deve ritrovarsi nella schermata principale del gioco ed aver terminato le "vite" |
| Post-Condizioni<br><br>L'Utente è riportato nella schermata di autenticazione |
| Scenario Principale<br><br>1) Il Sistema mostra all'Utente il risultato ottenuto nella partita<br>2) Il Sistema aggiorna la classifica<br>3) il Sistema esegue Autenticazione |
| Estensioni |

Figura 2.5: Caso d'uso Fine Partita

### Caso d'uso : Nuovo Livello

Questo caso d'uso viene eseguito in due casi:

- ogni volta che l'utente ha eliminato tutti i clienti della schermata
- all'inizio della partita per inizializzare il gioco

In entrambi i casi, successivamente viene eseguito il caso d'uso "Animazione".

| Caso d'Uso : Nuovo Livello |
|---|
| ID: 5 |
| Attore Primario : Sistema |
| Pre-Condizioni<br><br>L'Utente ha eliminato tutti i clienti oppure la partita è appena iniziata ed occorre inizializzare il livello.<br>. |
| Post-Condizioni<br><br>L'Utente interagisce con la schermata principale del gioco. |
| Scenario Principale<br><br>1) Il Sistema visualizza e posiziona il massimo numero di clienti nella schermata.<br>2) Il Sistema esegue ripetutamente Animazione |
| Estensioni |

Figura 2.6: Caso d'uso Nuovo Livello

### Caso d'uso : Lancio della Bottiglia

In questo caso d'uso, il sistema risponde ad un input da parte dell'utente che vuole far sparare il barista. Nella grafica occorre mostrare l'immagine di una bottiglia in corrispondenza della posizione del barista.

| Caso d'Uso : Lancio della Bottiglia |
|---|
| ID: 6 |
| Attore Primario : Utente |
| Pre-Condizioni |
| Post-Condizioni |
| Scenario Principale<br><br>1) l'Utente preme la barra spaziatrice della tastiera.<br>2) Il Sistema mostra l'immagine di una bottiglia immediatamente sopra a quella del barista. |
| Estensioni<br><br>2.a) c'é già una bottiglia in gioco<br><br>1) il Sistema non mostra nessuna ulteriore bottiglia al di fuori di quella in gioco |

Figura 2.7: Caso d'uso Lancio della Bottiglia

### Caso d'uso : Movimento Barista

In questo caso d'uso, il sistema risponde ad un input da parte dell'utente che vuole spostare il barista. Nella grafica occorre che l'immagine del barista cambi la propria posizione verso destra o verso sinistra a seconda della richiesta dell'utente.

| Caso d'Uso : Movimento barista |
|---|
| ID: 7 |
| Attore Primario : Utente |
| Pre-Condizioni<br><br>Il Sistema deve aver già posizionato l'immagine del barista nella schermata |
| Post-Condizioni<br><br>Il barista si è spostato verso destra o verso sinistra se non arrivato al limite dello schermo. |
| Scenario Principale<br><br>1) l'Utente seleziona con le freccie direzionali della tastiera la direzione verso il quale intende spostare il barista<br>2) Il Sistema aggiorna la posizione del barista nella schermata. |
| Estensioni<br><br>2.a) la posizione del barista ha raggiunto il limite destro o sinistro dello schermo<br>1) il Sistema non aggiorna la posizione del barista |

Figura 2.8: Caso d'uso Movimento Barista

### Caso d'uso : Visualizzazione Classifica

Questo caso d'uso è relativo alla visualizzazione della classifica. L'utente richiede al sistema di mostrare la classifica del gioco. Successivamente il sistema mostrerà nuovamente la schermata di gioco.

| Caso d'Uso : Visualizzazione Classifica |
|---|
| ID: 8 |
| Attore Primario : Utente |
| Pre-Condizioni |
| Post-Condizioni<br><br>L'Utente stà interagendo con la prima schermata del gioco |
| Scenario Principale<br><br>1) l'Utente seleziona dalla schermata l'opzione di visualizzazione della classifica<br>2) Il Sistema reperisce i dati relativi alla classifica e successivamente li mostra all'Utente.<br>3) l'Utente seleziona l'opzione di tornare alla schermata precedente |
| Estensioni<br><br>2.a) non esistono ancora dati in classifica<br>1) il Sistema notifica l'errore all'Utente |

Figura 2.9: Caso d'uso Visualizzazione Classifica

### Caso d'uso : Autenticazione

Questo caso d'uso viene eseguito in due momenti:

- all'avvio dell'applicazione
- quando l'utente ha terminato le "vite" a disposizione.

| Caso d'Uso : Autenticazione |
|---|
| ID: 9 |
| Attore Primario : Utente |
| Pre-Condizioni |
| Post-Condizioni<br><br>Il nome identificativo dell'Utente è stato registrato dal Sistema |
| Scenario Principale<br><br>1) il Sistema mostra la prima schermata dell'applicazione<br>2) l'Utente fornisce il proprio nome identificativo<br>3) il Sistema esegue Nuovo Livello |
| Estensioni<br><br>2.a) il nome identificativo ha una lunghezza di 0 caratteri<br>1) il Sistema notifica l'errore all'Utente<br>2) l'Utente deve fornire nuovamente il proprio nome identificativo<br><br>2.b) il nome identificativo ha una lunghezza maggiore di 15 caratteri<br>1) il Sistema notifica l'errore all'Utente<br>2) l'Utente deve fornire nuovamente il proprio nome identificativo |

Figura 2.10: Caso d'uso Autenticazione

## 2.2 Diagramma dei casi d'uso

Utilizzando la notazione grafica UML per i casi d'uso di figura 2.11, possiamo realizzare quello che viene chiamato "diagramma dei casi d'uso".

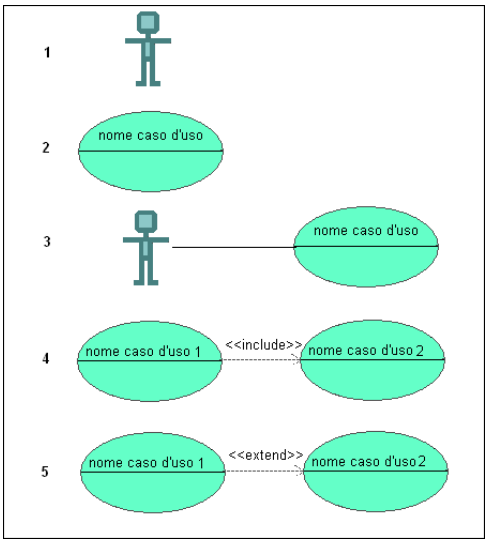



Figura 2.11: Rappresentazione UML grafica dei casi d'uso

La figura 2.11 mostra le convenzioni UML grafiche per rappresentare i casi d'uso. Sono illustrate le cinque convenzioni più importanti di questa rappresentazione:

- l'immagine al numero 1 rappresenta un attore cioè chi svolge le azioni del caso d'uso
- l'immagine al numero 2 rappresenta un caso d'uso
- l'immagine al numero 3 rappresenta un attore che prende parte ad un caso d'uso
- l'immagine al numero 4 rappresenta l'inclusione di un caso d'uso in un altro. Significa che durante lo svolgimento del caso d'uso numero 1 viene sempre svolto anche il caso d'uso 2

- l'immagine al numero 5 rappresenta un caso d'uso che ne estende un altro. Significa che durante lo svolgimento del caso d'uso numero 2 può essere che venga svolto anche il caso d'uso 1

Il diagramma dei casi d'uso, mostra le dipendenze che ci sono tra i vari casi d'uso ed è utile per avere una visione generale delle funzionalità del sistema. Il rettangolo nero in figura, delimita il confine che separa il sistema dall'ambiente esterno, dove troviamo l'utente che utilizza l'applicazione. In figura 2.12 viene riportato il diagramma dei casi d'uso del progetto BarInvaders.

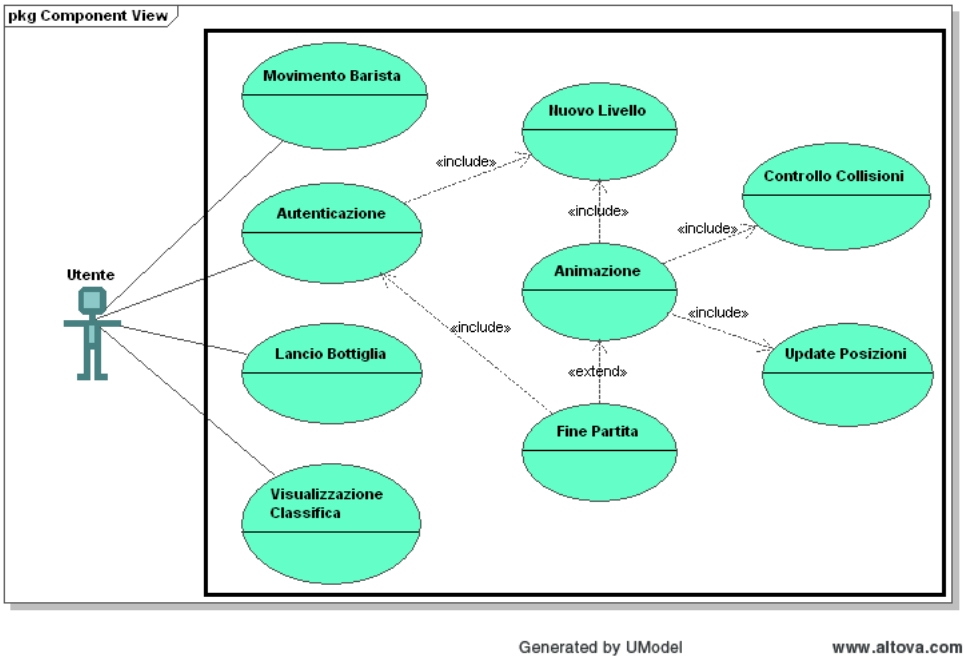



Figura 2.12: Diagramma dei casi d'uso

# Capitolo 3

# Bar invaders: progettazione

La progettazione è la fase di sviluppo precedente all'implementazione. L'analisi dei requisiti del capitolo 2 servirà ora ad introdurre una soluzione software alle funzionalità richieste dall'applicazione da parte dell'utente, modelleremo le classi e le rispettive responsabilità. Illustreremo inoltre alcune metodologie implementative dell'informatica legata ai videogiochi.

## 3.1 Specifiche di progetto

### Schermate del gioco

Prima di iniziare la progettazione vera e propria, è utile avere un'idea precisa della dinamica dell'applicazione. La figura 3.1, ha il compito di mostrare quale sia una possibile realizzazione della schermata principale del videogioco. Questa non sarà l'unica, poichè occorre introdurne un'altra supplementare nella quale l'utente avrà la possibilità di autenticarsi al sistema. Questo è necessario in quanto dobbiamo associare un nome identificativo ad ogni utente da inserire eventualmente nella classifica del gioco.

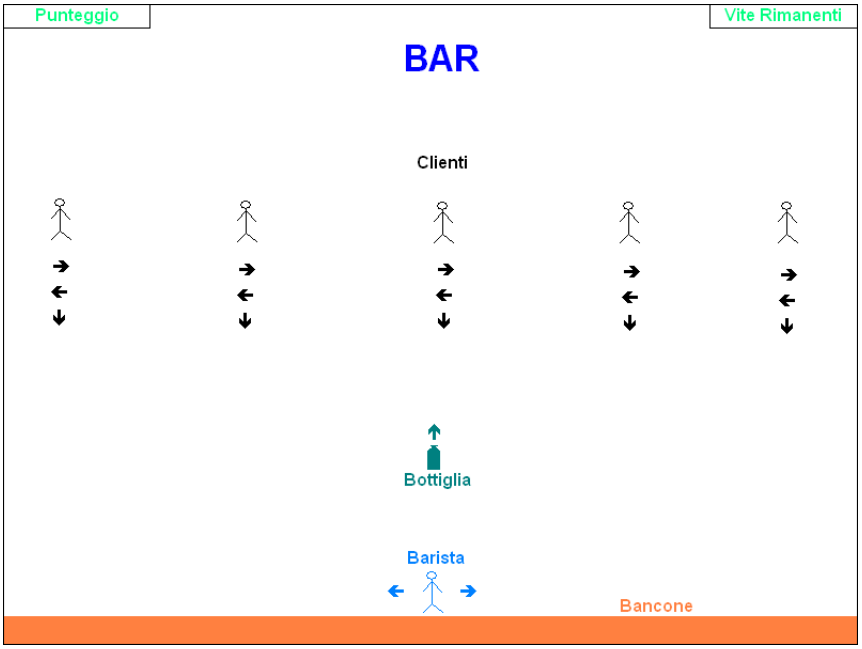



Figura 3.1: Bozza della schermata principale

### Scopo del gioco

Lo scopo del gioco, è quello di difendere il bancone del bar dal tentativo di invasione e dall'attacco dei clienti, che si avvicinano pian piano ad esso.

### Barista

Il ruolo del barista, controllato dall'utente, è quello di difendere il bancone. Il barista può lanciare delle bottiglie di birra nell'intento di colpire uno dei clienti. Le bottiglie lanciabili sono illimitate, ma il barista può lanciarne solo una per volta. Egli ha la possibilità di muoversi verso destra o verso sinistra orizzontalmente sul fondo dello schermo. Per fare ciò, l'utente non dovrà fare altro che premere sulla tastiera una delle freccie direzionali, scegliendo tra sinistra o destra. In seguito alla pressione della barra spaziatrice invece, il barista lancierà una bottiglia in direzione dei clienti.

### Clienti

L'obiettivo dei clienti è quello di "conquistare" il bancone del bar. Ad ogni aggiornamento grafico, si spostano dall'alto della schermata verso il basso ed in modo casuale verso destra o verso sinistra. Viene estratto un numero random tra 0 e 10 : se il numero è strettamente maggiore di 5, il cliente si sposta verso destra, altrimenti verso sinistra. Un cliente colpito da una bottiglia lanciata dal barista, è eliminato e non concorre più alla conquista del bancone. Inizialmente nel bar è presente un solo cliente che viene sostituito da un gruppo di due clienti quando è eliminato; i due saranno sostituiti da un gruppo di tre clienti e così via fino ad un massimo di cinque. Se anche quest'ultimo gruppo di clienti viene sconfitto, il ciclo appena descritto riparte, ma aumenta la velocità di avvicinamento al bancone. Il processo appena descritto è mostrato in figura 3.2.

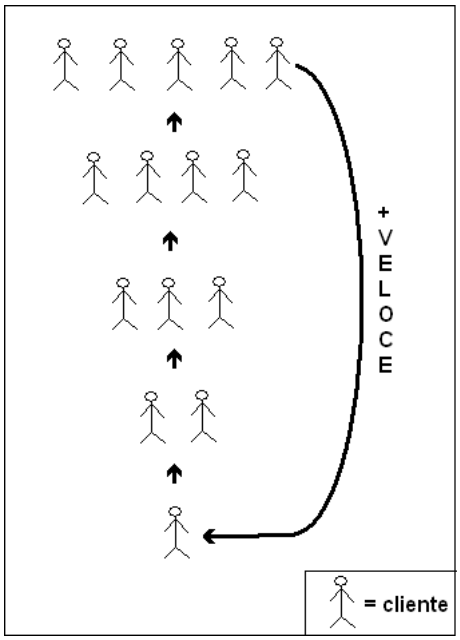



Figura 3.2: Aumento della velocità dei clienti

Se il cliente non viene colpito dalla bottiglia, continua il suo avanzamento discendente verso il bancone. In questo caso entrerà in contatto col barista oppure con il bancone del bar, posto in basso alla schermata di gioco. In seguito alla collisione, quel cliente verrà eliminato causando la perdita di una "vita" all'utente.

## Conclusione della partita

L'utente ha a disposizione tre "vite". Ogni volta che si verificherà uno di questi eventi:

- un cliente raggiunge il bancone
- un cliente raggiunge il barista

all'utente verrà sottratta una "vita", fino a quando non saranno ultimate. In quel caso la partita termina. Nel caso l'utente voglia terminare una partita anticipatamente, non deve fare altro che premere il tasto "q" della tastiera.

### Punteggio

Ogni cliente eliminato, permette all'utente di guadagnare 30 punti incrementando il punteggio della partita che inizialmente parte da zero. Il punteggio non può superare i 10000 punti. Se viene raggiunto tale punteggio, la partita termina .

### Classifica

La classifica del gioco è aggiornata alla fine di ogni partita e mostra soltanto i migliori cinque risultati ottenuti dagli utenti.

### Suoni

Una musica di sottofondo accompagna il giocatore per tutta la durata della partita ed anche le collisioni saranno seguite da effetti sonori.

## 3.2 Sprite e collisioni

Prima di procedere alla progettazione è necessario introdurre alcuni concetti preliminari che ci saranno utili successivamente. Il progetto del videogioco che si vuole realizzare, prevede la gestione di *"Sprite"*. In grafica informatica, lo Sprite è una figura bidimensionale mobile che può essere spostata rispetto allo scenario del videogioco, che rimane invece fisso[25]. Il barista, i clienti e la bottiglia sono esempi di figure bidimensionali che devono essere disegnate sopra ad uno sfondo che invece rimane fisso perchè non necessita di spostamenti.

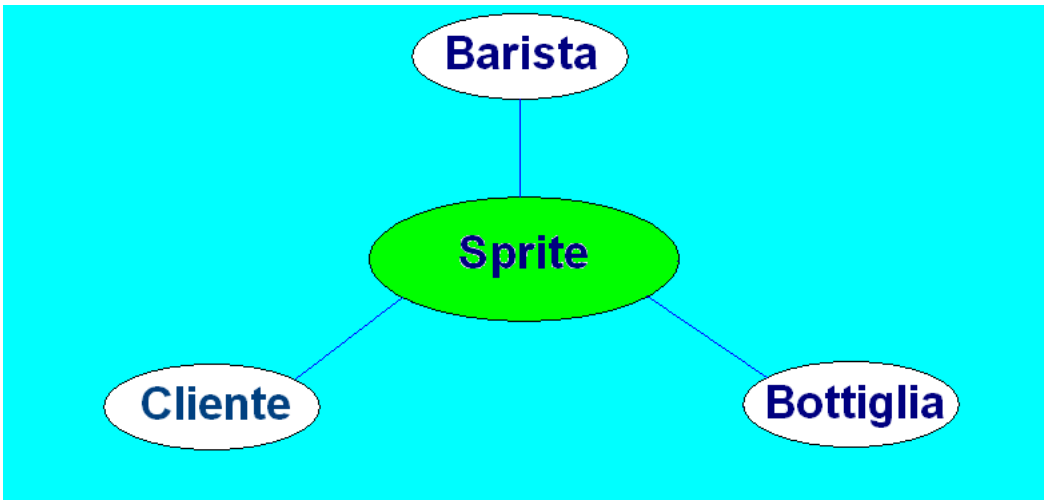



Figura 3.3: Sprite utilizzati nel progetto

Un altro aspetto fondamentale dei videogiochi, come quello che si vuole realizzare, sono le *"collisioni"*. Dalle specifiche, è infatti emerso per esempio che i clienti verranno colpiti da una bottiglia. Nell'informatica legata allo sviluppo dei videogiochi, si utilizza il termine "collisione" per identificare la situazione in cui due oggetti grafici, si ritrovano nella stessa posizione nella schermata[25]. Se provassimo ad immaginare un piano cartesiano, potremmo immaginare una collisione come due figure geometriche sovrapposte. Le collisioni da gestire nel videogioco sono le seguenti:

- tra una bottiglia ed un cliente
- tra i clienti ed il barista
- tra i clienti ed il bancone.

Come è facilmente intuibile da questo elenco, avemo collisioni dall'alto e dal basso. Vediamo una figura per capire meglio.

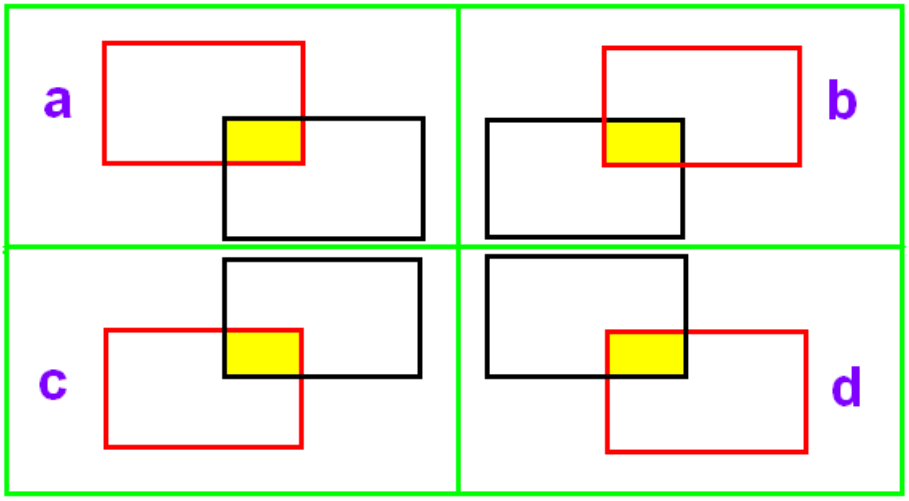



Figura 3.4: Tipologie di collisione

Il rettangolo nero è il nostro oggetto grafico di riferimento, mentre quello rosso è il rettangolo entrato in collisione. Le figure "a" e "b" mostrano una collisione dall'alto fra due "Sprite", rispettivamente con l'angolo in basso a destra e sinistra. Le figure "c" e "d" mostrano invece una collisione dal basso, rispettivamente con l'angolo in alto a destra e sinistra. Queste sono le tipologie di collisione che andremo a controllare nel videogioco.

## 3.3 Individuazione delle classi e schede CRC

Il linguaggio scelto per implementare l'applicazione è il Java, un linguaggio di programmazione orientato agli oggetti. Gli oggetti sono entità all'interno di un sistema dotate di responsabilità e caratteristiche. Queste entità interagiscono tra di loro tramite richieste di servizi da svolgere. Ogni oggetto appartiene ad una classe che ne definisce le responsabilità, le caratteristiche distintive ed i servizi che esso può svolgere. Dalle specifiche sono state individuate le seguenti classi:

- Cliente : questa è la classe rappresentante i clienti del bar

- Barista : questa è la classe rappresentante il barista del bar
- Bottiglia : questa è la classe rappresentante la bottiglia che il barista lancia in direzione dei clienti.

Facendo riferimento a quanto detto nel paragrafo 3.2 introduciamo una classe supplementare chiamata "Sprite" che racchiude le caratteristiche comuni per tutti gli oggetti grafici che vogliamo inserire nel videogioco. Questa classe deve essere poi estesa e modellata per ogni diverso tipo di figura che vogliamo disegnare, con le proprie caratteristiche che la diversificano dalle altre. Per uno "Sprite" ci interessano:

- la posizione nello schermo
- la larghezza
- l'altezza
- l'immagine caratteristica
- la quantità di spostamento da effettuare durante il movimento dello stesso

Oltre a queste classi, occorre aggiungerne altre che sono necessarie per quanto riguarda la gestione grafica dell'applicazione:

- Principale : è la classe da cui potremo avviare l'esecuzione dell'applicazione
- Frame : è la classe che funge da "contenitore" dei pannelli utilizzati nelle schermate dell'applicazione
- PannelloDiGioco : è la classe che racchiude e gestisce tutto il nucleo del videogioco ossia l'animazione dei clienti, della bottiglia e del barista
- PannelloAutenticazione : è la classe che racchiude sia l'autenticazione dell'utente nel gioco che la visualizzazione della classifica

- CaricatoreImmagini : è la classe incaricata della gestione di tutte le immagini utilizzate nel gioco

Per quanto riguarda le gestione degli effetti sonori occorrono le seguenti classi:

- CaricatoreSuoni : è la classe incaricata della gestione degli effetti sonori nelle collisioni
- CaricatoreMidi : è la classe incaricata della gestione della musica di sottofondo durante la partita
- MidiInfo : è la classe che rappresenta il file audio utilizzato nella musica di sottofondo
- Suono : è la classe che rappresenta il file utilizzato come effetto sonoro

Ora per comprendere meglio le responsabilità e le collaborazioni di queste classi, introduciamo una tecnica nell'ingegneria del software, le cosiddette schede CRC [1][28]. Utilizziamo una scheda per ogni classe, dove si riportano le proprie responsabilità e le collaborazioni esistenti. La figura 3.5 mostra una scheda CRC standard.

| **Nome** : nome classe | |
|---|---|
| | |
| **Responsabilità** | **Collaborazioni** |
| | |
| responsabilità 1 | nome classe 1 |
| responsabilità 2 | nome classe 2 |
| | nome classe 3 |

Figura 3.5: Scheda CRC standard

Nei paragrafi seguenti sono mostrate le schede CRC per ogni classe del progetto.

[1] Acronimo di Classe Responsabilità Collaborazioni

### Scheda CRC della classe Sprite

| **Nome** : Sprite | |
|---|---|
| | |
| **Responsabilità** | **Collaborazioni** |
| | |
| E' la superclasse che ogni classe relativa ad uno Sprite dovrà estendere | PannelloDiGioco |

Figura 3.6: Scheda CRC della classe Sprite

### Scheda CRC della classe Cliente

| **Nome** : Cliente | |
|---|---|
| | |
| **Responsabilità** | **Collaborazioni** |
| | |
| Si occupa della gestione dello Sprite raffigurante il cliente | PannelloDiGioco |

Figura 3.7: Scheda CRC della classe Cliente

### Scheda CRC della classe Barista

| **Nome** : Barista | |
|---|---|
| | |
| **Responsabilità** | **Collaborazioni** |
| | |
| Si occupa della gestione dello Sprite raffigurante il barista | PannelloDiGioco |

Figura 3.8: Scheda CRC della classe Barista

### Scheda CRC della classe Bottiglia

| **Nome** : Bottiglia | |
|---|---|
| | |
| **Responsabilità** | **Collaborazioni** |
| | |
| Si occupa della gestione dello Sprite raffigurante la bottiglia | PannelloDiGioco |

Figura 3.9: Scheda CRC della classe Bottiglia

### Scheda CRC della classe Principale

| **Nome** : Principale | |
|---|---|
| | |
| **Responsabilità** | **Collaborazioni** |
| | |
| Crea il frame nel quale verranno inseriti i pannelli | Frame |

Figura 3.10: Scheda CRC della classe Principale

### Scheda CRC della classe Frame

| **Nome** : Frame | |
|---|---|
| | |
| **Responsabilità** | **Collaborazioni** |
| | |
| Crea le istanze delle classi con cui collabora, in modo da poter graficamente interagire con pannelli diversi scambiabili all'occorrenza. | PannelloDiGioco<br>PannelloAutenticazione |

Figura 3.11: Scheda CRC della classe Frame

### Scheda CRC della classe PannelloDiGioco

| **Nome** : PannelloDiGioco | |
|---|---|
| | |
| **Responsabilità** | **Collaborazioni** |
| | |
| Si occupa della gestione della schermata principale del gioco aggiornando continuamente la schermata di gioco con gli Sprite in posizioni diverse. | Barista<br>Cliente<br>Frame<br>Bottiglia<br>Sprite<br>CaricatoreImmagini<br>CaricatoreMidi<br>CaricatoreSuoni |

Figura 3.12: Scheda CRC della classe PannelloDiGioco

### Scheda CRC della classe PannelloAutenticazione

| **Nome** : PannelloAutenticazione | |
|---|---|
| | |
| **Responsabilità** | **Collaborazioni** |
| | |
| Registra e controlla il dato relativo al nome identificativo dell'Utente nel gioco<br><br>Permette la visualizzazione della classifica | <br><br>Frame |

Figura 3.13: Scheda CRC della classe PannelloAutenticazione

### Scheda CRC della classe CaricatoreImmagini

| **Nome** : CaricatoreImmagini | |
|---|---|
| | |
| **Responsabilità** | **Collaborazioni** |
| | |
| Il ruolo di questa classe è simile ad una banca dati contenente tutte le immagini utilizzate nell'applicazione. | Barista<br>Cliente<br>Frame<br>Bottiglia<br>PannelloAutenticazione<br>PannelloDiGioco<br>Sprite |

Figura 3.14: Scheda CRC della classe CaricatoreImmagini

### Scheda CRC della classe CaricatoreSuoni

| **Nome** : CaricatoreSuoni | |
|---|---|
| | |
| **Responsabilità** | **Collaborazioni** |
| | |
| Ha il compito di interfacciare la classe PannelloGioco alla classe Suono nell'esecuzione di file Wav | PannelloDiGioco<br>Suono |

Figura 3.15: Scheda CRC della classe CaricatoreSuoni

### Scheda CRC della classe CaricatoreMidi

| **Nome** : CaricatoreMidi | |
|---|---|
| | |
| **Responsabilità** | **Collaborazioni** |
| | |
| Ha il compito di interfacciare la classe PannelloGioco alla classe MidiInfo nell'esecuzione di file MIDI | PannelloDiGioco<br>MidiInfo |

Figura 3.16: Scheda CRC della classe CaricatoreMidi

### Scheda CRC della classe MidiInfo

| **Nome** : midiInfo | |
|---|---|
| | |
| **Responsabilità** | **Collaborazioni** |
| | |
| Questa classe contiene tutte le informazioni in merito al file Midi in esecuzione. | CaricatoreMidi |

Figura 3.17: Scheda CRC della classe MidiInfo

### Scheda CRC della classe Suono

| **Nome** : Suono | |
|---|---|
| | |
| **Responsabilità** | **Collaborazioni** |
| | |
| Questa classe contiene tutte le informazioni in merito al file Wav in esecuzione. | CaricatoreSuoni |

Figura 3.18: Scheda CRC della classe Suono

## 3.4 Diagramma delle classi

Nel paragrafo 3.3 abbiamo individuato le classi software del nostro sistema. Il secondo passo è stato quello di assegnare delle responsabilità a queste classi. Ora, grazie all'ausilio della notazione standard UML[2] [28], costruiamo il cosiddetto "diagramma delle classi". In figura 3.19 sono mostrate tutte le classi del nostro sistema alle quali sono stati assegnati attributi ed operazioni. Gli attributi rappresentano le proprietà di una classe. Le operazioni, sono i compiti che possono portare a termine gli oggetti istanziati appartenenti a quella classe. Le operazioni permettono di modificare gli attributi dell'oggetto, oppure effettuare azioni legate alla responsabilità della classe a cui appartiene.

[2] Acronimo di Unified Modeling Language - http://it.wikipedia.org/wiki/Unified_Modeling_Language

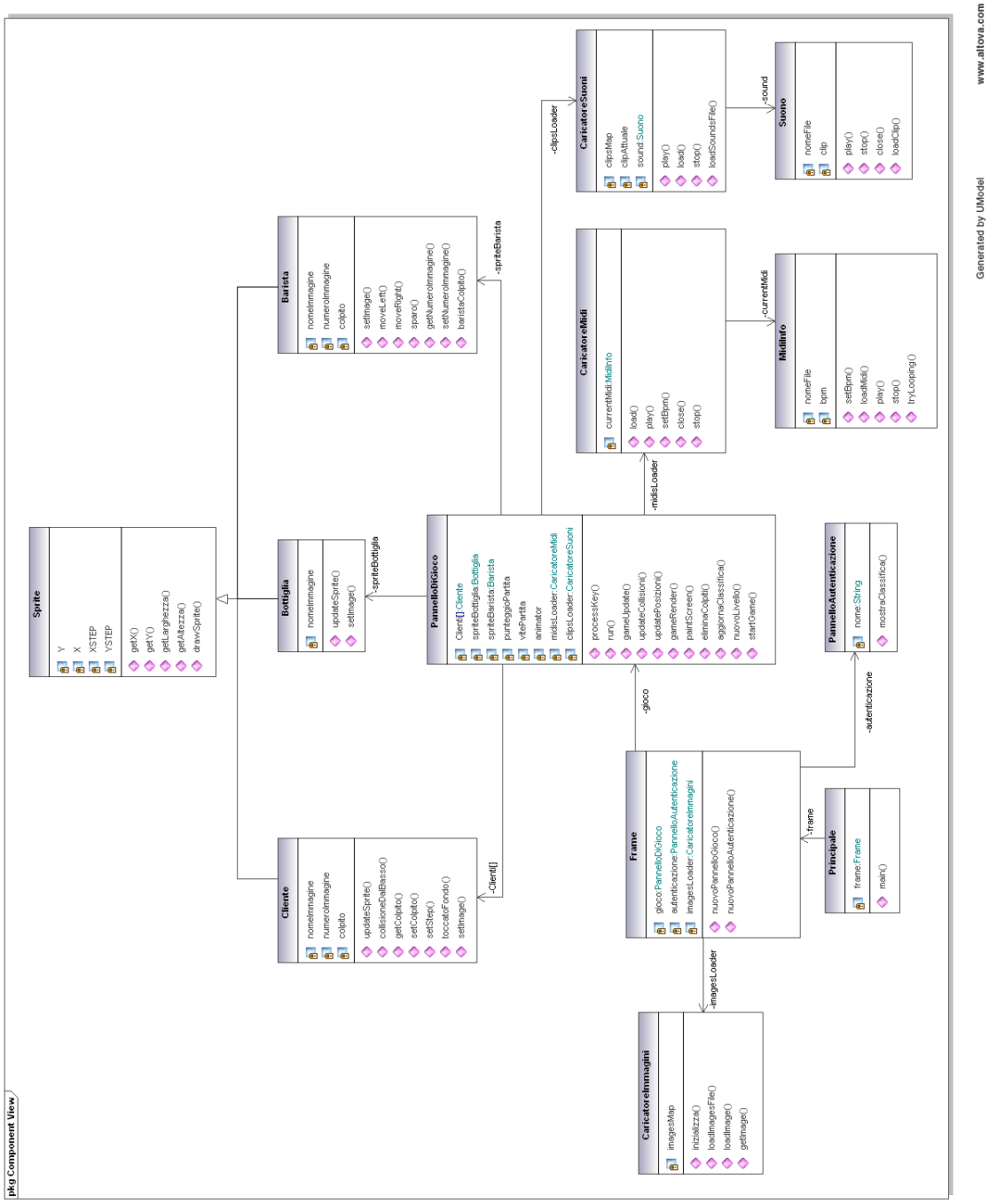



Figura 3.19: Diagramma delle classi

## 3.5 Diagrammi di sequenza

I Diagrammi di Sequenza vengono utilizzati per descrivere i flussi di messaggi tra diversi oggetti o entità che collaborano all'interno di uno scenario. Uno scenario è una determinata sequenza di azioni che compiono diversi oggetti collettivamente durante l'esecuzione di un caso d'uso. Questi diagrammi fanno parte della notazione standard UML [28].

### Diagramma di sequenza : "Update Posizioni"

Il metodo updateSprite() è utilizzato dai vari "Sprite" per modificare la posizione nello schermo dell'immagine caratteristica ad essi associata. La quantità di spostamento e la direzione dello spostamento, variano da una classe all'altra, in quanto, la bottiglia si sposta verso l'alto mentre i clienti verso il basso, destra o sinistra.

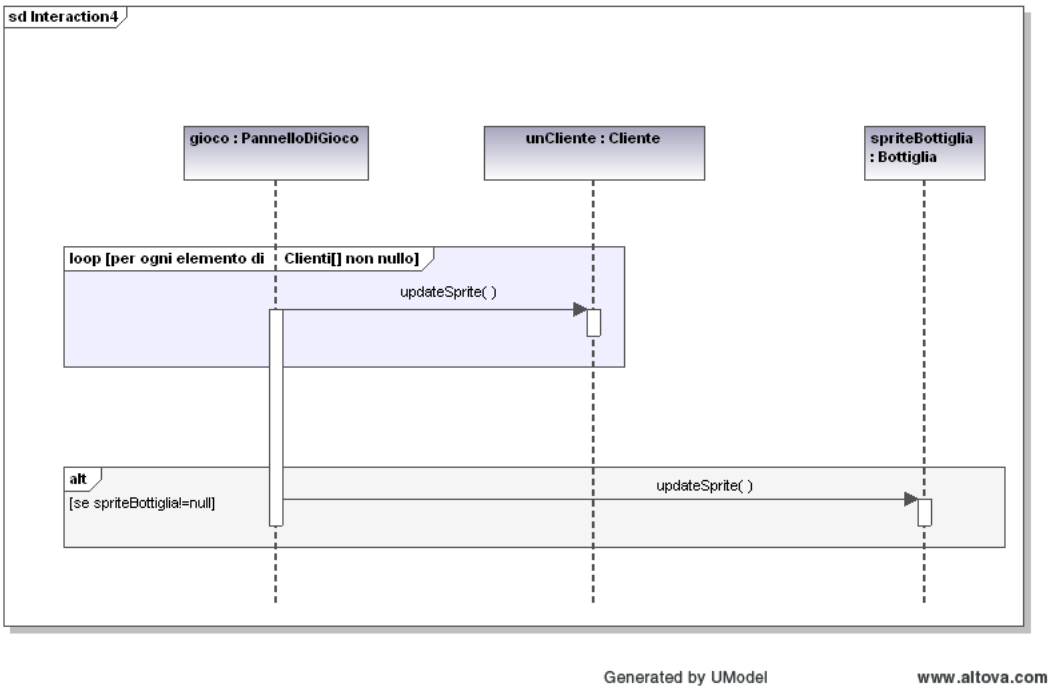



Figura 3.20: Diagramma di sequenza Update Posizioni

### Diagramma di sequenza : "Controllo Collisioni"

Il medodo collisioneDalBasso() implementa il concetto di collisione tra "Sprite" introdotto nel paragrafo 3.2. Effettua un controllo sulle coordinate grafiche di due elementi per constatare o meno una sovrapposizione delle relative immagini associate. In caso sia rilevata una collisione, il metodo setColpito() e baristaColpito() rispettivamente delle classi Cliente e Barista, settano il proprio l'attributo "colpito" di tipo Booleano[3] al valore "true". Ogni nuovo oggetto della classe Cliente e Barista ha impostato di default il valore "false" per quell'attributo. In questo diagramma di sequenza vengono controllate tre tipologie di collisione:

- tra i clienti e la bottiglia : in caso di collisione, eliminiamo lo sprite della Bottiglia perchè ha colpito un cliente
- tra i clienti ed il barista : in caso di collisione, assegno all'attributo "colpito" del barista e del cliente coinvolto il valore "true"
- tra i clienti ed il bancone: il metodo toccatoFondo() della classe Cliente ci informa sulle coordinate grafiche del cliente. Se ha raggiunto il bancone, assegno all'attributo "colpito" della classe Cliente il valore "true" perchè è entrato in collisione.

[3] Un tipo di dato Booleano può assumere solamente due valori : true o false

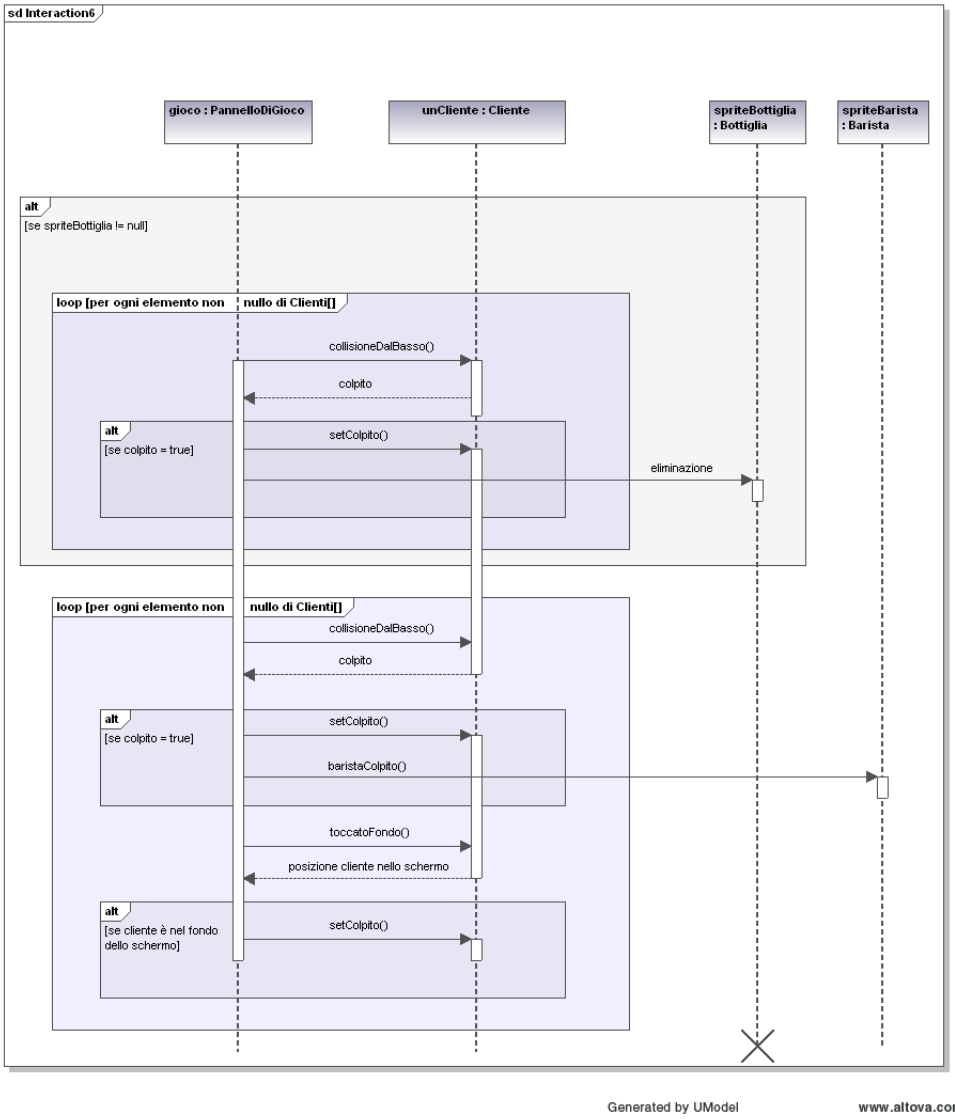



Figura 3.21: Diagramma di sequenza Controllo Collisioni

### Diagramma di sequenza : "Fine Partita"

La classe Frame ha il compito di sostituire il pannello grafico in cui si svolge il gioco, con il pannello di autenticazione e viceversa. Se l'utente ha terminato le "vite" a sua disposizione nella partita, si succedono in sequenza queste azioni :

- l'oggetto "gioco" di tipo "pannelloDiGioco" mostra il risultato ottenuto all'utente nella partita nella GUI[4]
- l'oggetto "gioco" di tipo "pannelloDiGioco" aggiorna la classifica chiamando il metodo aggiornaClassifica()
- l'oggetto "frame" della classe "Frame" crea un nuovo oggetto della classe "PannelloAutenticazione" ed elimina l'oggetto "gioco" di tipo "PannelloDiGioco".

[4]Acronimo di Graphical User Interface - rappresenta l'interfaccia grafica dell'applicazione

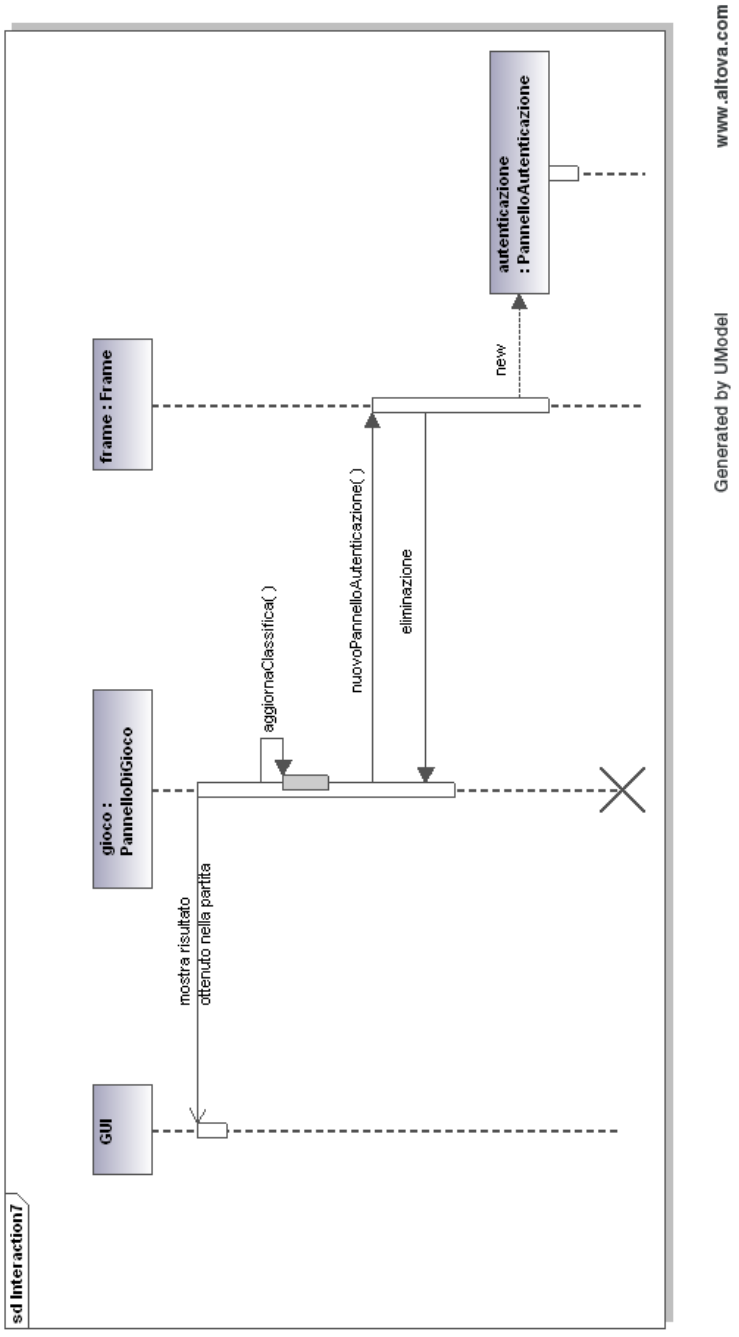



Figura 3.22: Diagramma di sequenza Fine Partita

### Diagramma di sequenza : "Lancio della Bottiglia"

Alla pressione della barra spaziatrice sulla tastiera, l'oggetto "gioco" di tipo "PannelloDiGioco" invoca il metodo processKey(). Il sistema riconosce quale tasto è stato premuto e crea un oggetto della classe Bottiglia che rappresenta lo Sprite della bottiglia da lanciare.

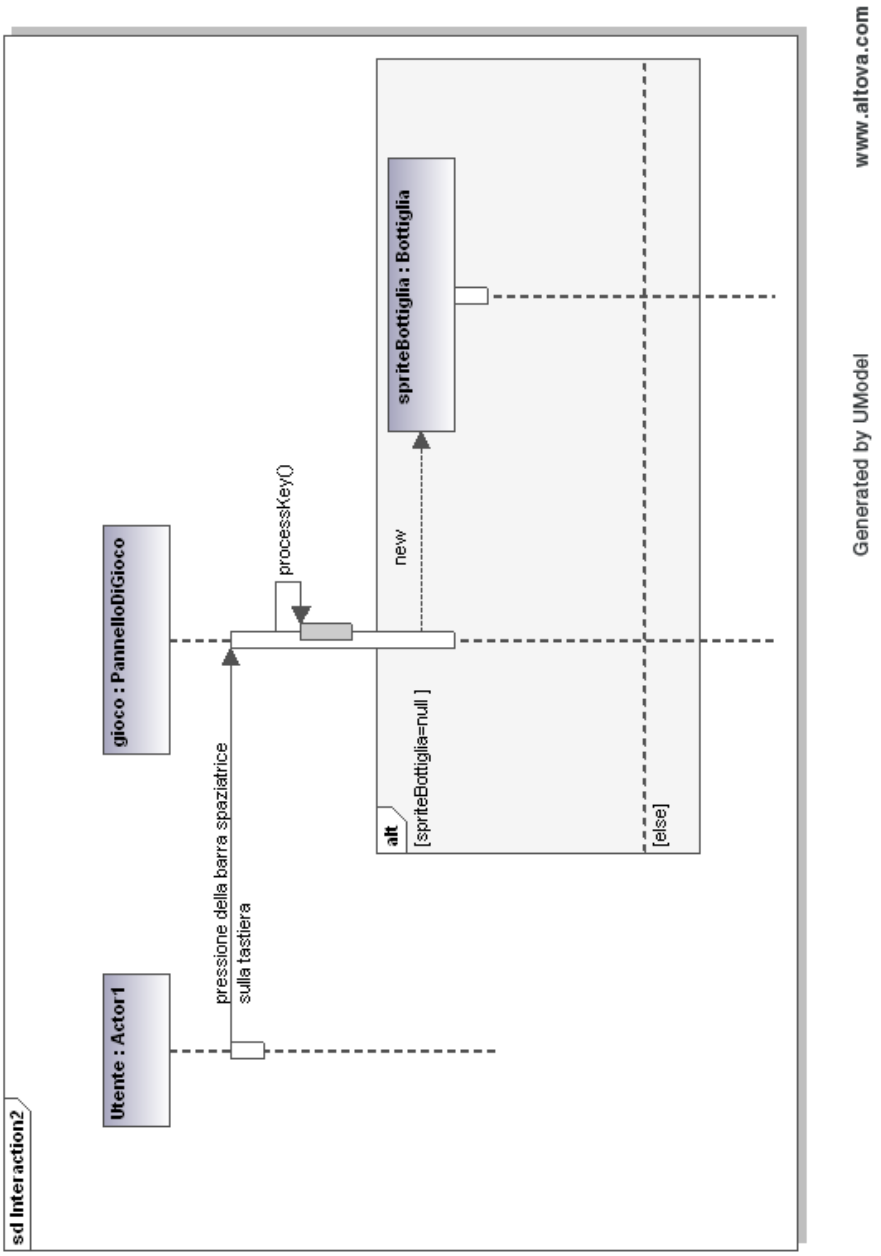



Figura 3.23: Diagramma di sequenza Lancio della Bottiglia

## Diagramma di sequenza : "Movimento Barista"

Il metodo processKey() della classe PannelloDiGioco, è invocato dall'ascoltatore di eventi della tastiera ogni volta che viene premuto un tasto. Se il tasto premuto è la freccia direzionale destra o sinistra, vengono invocati rispettivamente il metodo moveRight() o moveLeft() dell'oggetto "SpriteBarista" di tipo "Barista". La funzione dei metodi appena citati, è di modificare la posizione del barista verso destra o sinistra, sempre che non siano stati raggiunti i limiti della schermata. In questo caso non viene aggiornata la posizione.

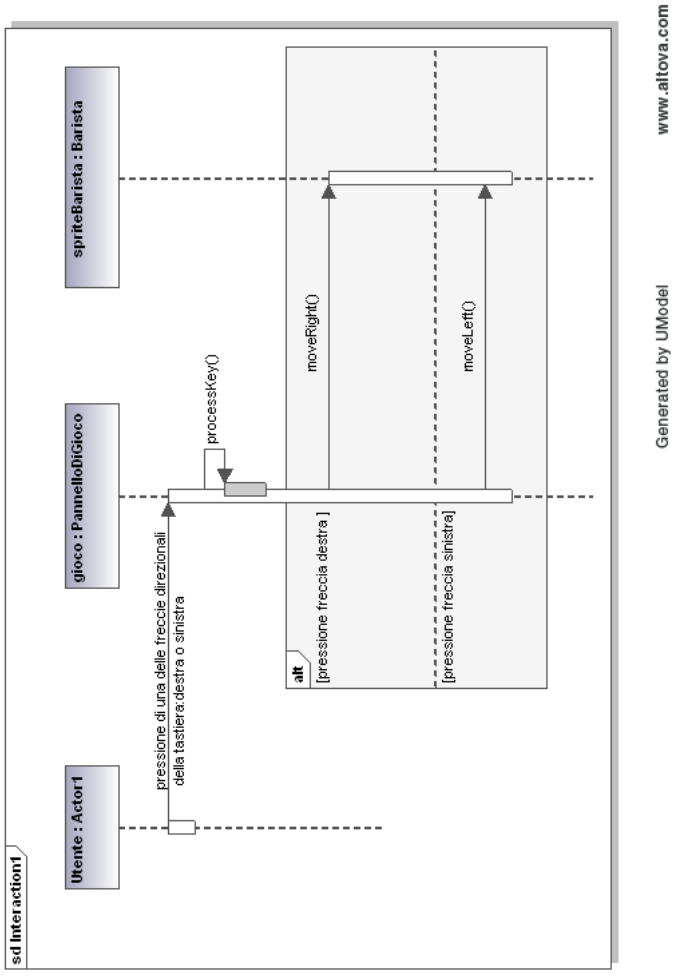



Figura 3.24: Diagramma di sequenza Movimento Barista

### Diagramma di sequenza : "Visualizza Classifica"

Dal pannello di autenticazione, rappresentato dalla classe "PannelloAutenticazione", è possibile visualizzare la classifica. Il metodo leggiFileClassifica(), preleva le informazioni dal file dove sono contenuti i cinque migliori risultati ottenuti dagli utenti nel videogioco. In seguito viene mostrata la classica all'utente nella GUI[5].

[5] Acronimo di Graphical User Interface - rappresenta l'interfaccia grafica dell'applicazione

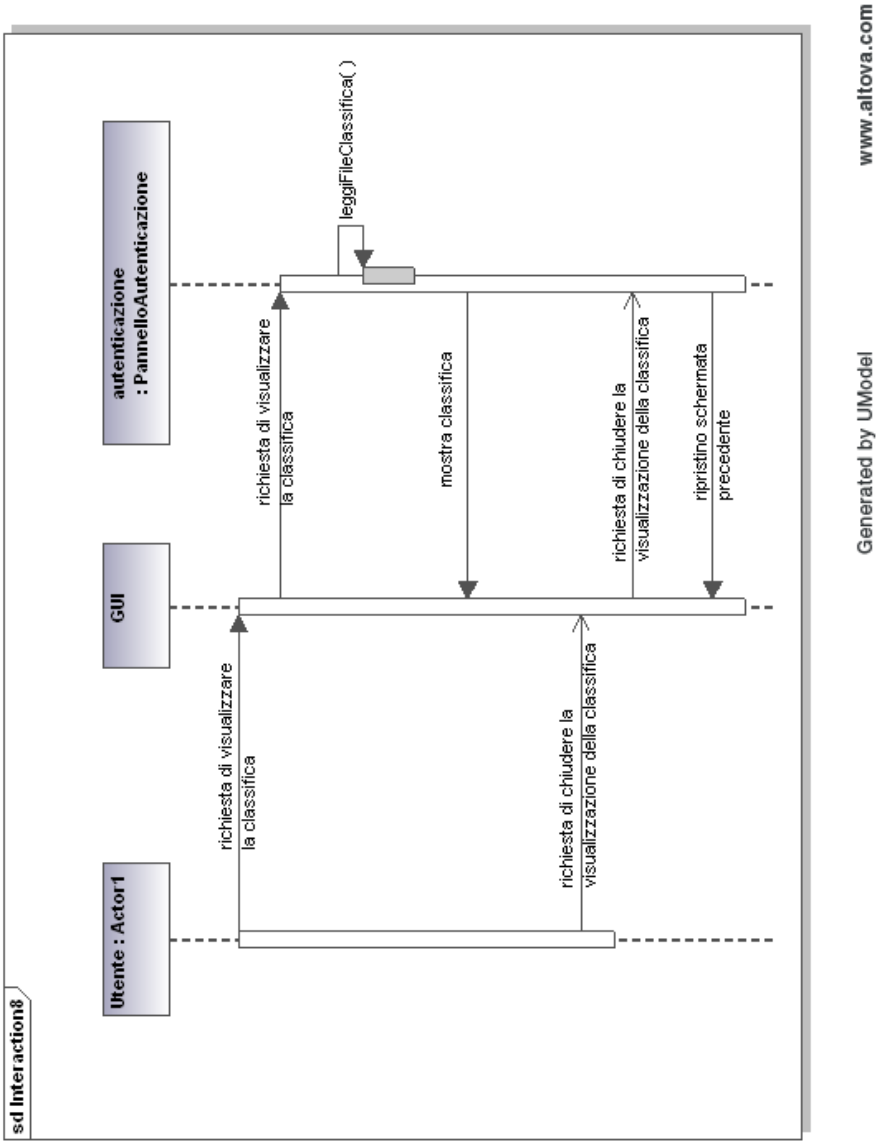



Figura 3.25: Diagramma di sequenza Visualizza Classifica

### Diagramma di sequenza : "Autenticazione"

Il pannello di autenticazione rappresentato dalla classe "PannelloAutenticazione" controlla i dati inseriti dall'utente nella GUI [6]. Se il nome identificativo scelto dall'utente per il gioco ha una lunghezza compresa tra 1 e 15 caratteri, viene creato un nuovo pannello di gioco rappresentato dalla classe "PannelloDiGioco". In caso contrario il sistema segnala all'utente la necessità di ripetere l'inserimento dei propri dati.

[6] Acronimo di Graphical User Interface - rappresenta l'interfaccia grafica dell'applicazione

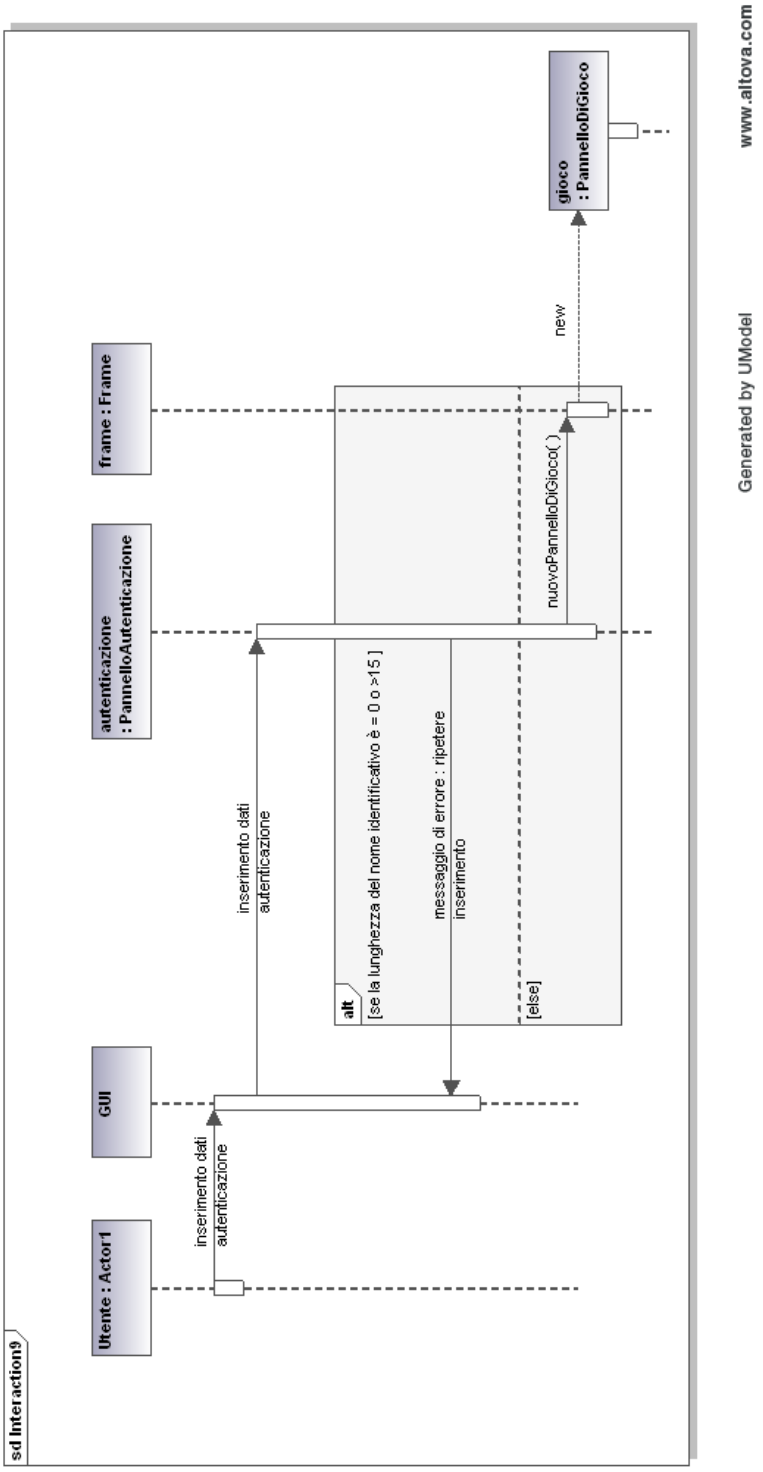



Figura 3.26: Diagramma di sequenza Autenticazione

### Diagramma di sequenza : "Elimina Colpiti"

L'eliminazione dei "colpiti", serve ad evitare che graficamente vengano disegnate nella schermata di gioco, gli "Sprite" già precedentemente colpiti da una bottiglia oppure entrati in collisione con il bancone . Ogni volta che uno "Sprite" è stato colpito, l'attributo Booleano[7] "colpito" di ogni oggetto di tipo Cliente viene posto al valore "true". Il metodo getColpito() della classe Cliente, restituisce il valore di quell'attributo. Se il valore è pari a "true" quel cliente viene eliminato.

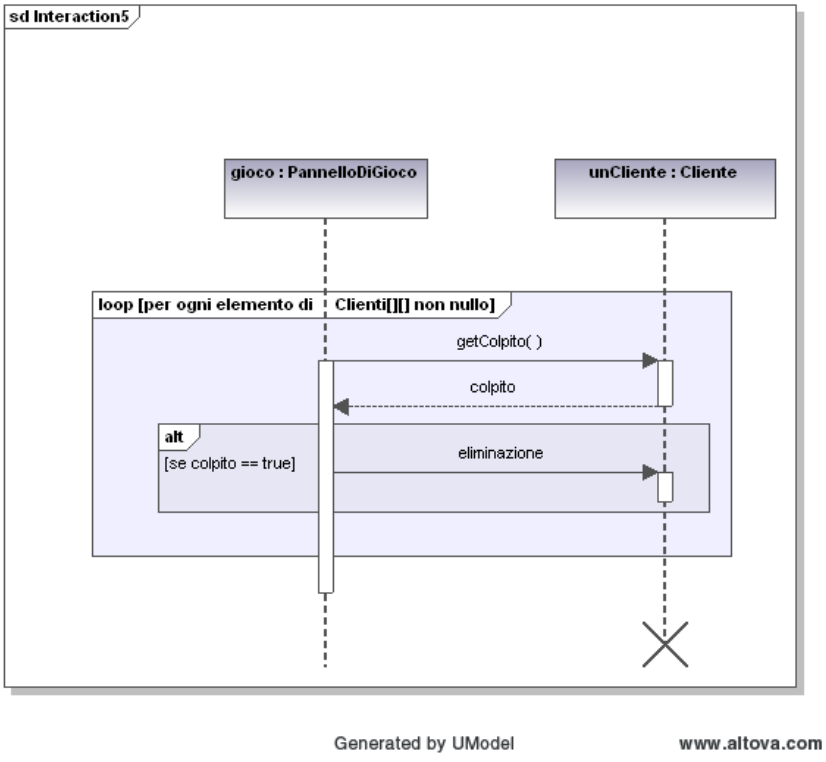



Figura 3.27: Diagramma di sequenza Elimina Colpiti

[7]Un tipo di dato Booleano può assumere solamente due valori : true o false

### Diagramma di sequenza : "Render"

Il metodo drawSprite() di ogni "Sprite" del videogioco, è invocato per implementare il double buffering discusso nelle considerazioni iniziali della progettazione. Sintetizzando,andremo a disegnare l'immagine caratteristica di ogni "Sprite", nell'oggetto grafico di una immagine avente le stesse dimensioni della schermata di gioco.

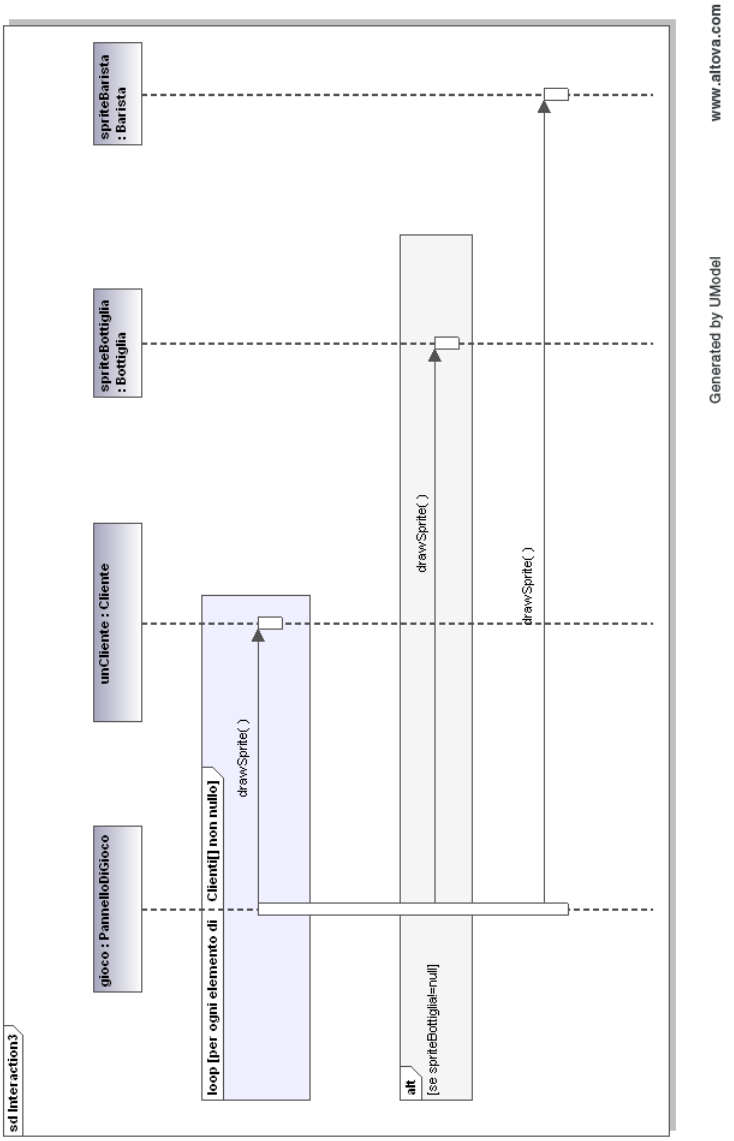



Figura 3.28: Diagramma di sequenza Render

## 3.6 Utilizzo dei thread

Per capire cosa sia un thread, proviamo ad immaginarci come lavora un avvocato. Di norma, egli dovrà seguire una serie di casi giudiziari contemporaneamente, che possono durare parecchi anni. Se l'avvocato aspettasse invece la fine di un processo prima di lavorare ad un altro caso, sarebbe efficiente? Assolutamente no, perchè rimarrebbe senza far nulla per parecchio tempo in attesa del successivo appello giudiziario. Passando al calcolatore, possiamo paragonare il processore ad un avvocato, ed i thread ai casi giudiziari che esso segue. La CPU[8] passa continuamente da un lavoro all'altro per evitare di essere inutilizzata durante i periodi di attesa. I thread sono sostanzialmente parti di codice in esecuzione appartenenti ad applicazioni diverse oppure alla stessa. Ad ogni programma in esecuzione corrisponde almeno un thread. Vediamo ora quali strumenti ci mette a disposizione il linguaggio Java per gestirli. Vi sono due modi di utilizzare i thread in java:

- estendere la classe Thread[9] dalla classe che ci interessa :

  ```
  nomeDellaClasse extends Thread
  ```

- implementare l'interfaccia Runnable nella classe che ci interessa :

  ```
  nomeDellaClasse implements Runnable
  ```

Utilizzando la prima modalità potremo creare ed avviare un thread utilizzando il seguente codice:

```
nomeDellaCasse nome = new nomeDellaClasse();
nome.start();
```

Nella seconda modalità invece:

[8] Acronimo di Central Processing Unit - è l'unità centrale di elaborazione di un calcolatore

[9] – java.lang.Thread appartenente alla libreria Java

```
nomeDellaCasse nome = new nomeDellaClasse();
new Thread(nome).start();
```

Entrambi i metodi "start()" invocano un metodo chiamato *run()* che deve essere presente nella nostra classe. All'interno di questo metodo, dobbiamo inserire la parte di codice da mandare in esecuzione all'avvio del thread. Nell'applicazione *BarInvaders* realizzata in questa tesi, è stato utilizzato un thread per gestire in un ciclo continuo tutte le fasi dell'animazione spiegate nel paragrafo 3.7.

## 3.7 Animazione con Double Buffering

Il "motore" del gioco, ovvero la parte principale, è sicuramente concentrata nell'animazione delle immagini dei vari "Sprite" che si spostano nello schermo. Analizzando il videogioco in dettaglio occorre gestire:

- il movimento dei clienti che si spostano verso il basso a destra o sinistra
- il movimento del barista sul fondo dello schermo verso destra e sinistra
- il movimento della bottiglia lanciata dal barista verso l'alto

Per realizzare tutto ciò, si andrà ad implementare lo stesso metodo utilizzato nei videogiochi commerciali 2D. L'animazione è divisa in quattro fasi distinte, che vengono continuamente eseguite in sequenza all'interno di un ciclo:

- Update
- Render
- Paint Screen
- Ritardo

Nella fase di Update si controllano le collisioni tra i vari Sprite e si aggiornano le posizioni degli stessi facendoli muovere della quantità di movimento

relativa. Nella fase di Render occore creare un'oggetto della classe Java Image[10][37] della stessa grandezza della schermata di gioco, ed utilizzo poi il contesto grafico di questa immagine per disegnarci lo sfondo della schermata ed ogni "Sprite". Nella fase di Paint Screen,si disegna l'immagine realizzata nella precedente fase sullo schermo. Nell'ultima fase di Ritardo, si introduce un ritardo nell'esecuzione dell'applicazione altrimenti, la grafica avrebbe un aggiornamento troppo veloce e l'utente non sarebbe in grado di utilizzare il videogioco.Variando il ritardo, è possibile realizzare lo spostamento più rapido dei clienti mano a mano che si avvicinano al bancone. Se abbassiamo il ritardo, la grafica sarà aggiornata più frequentemente e questo darà la sensazione di una maggiore velocità dei clienti nello spostamento. La tecnica appena spiegata prende il nome di "*Double Buffering*[25]. Il vantaggio di usare questo tipo di tecnica, risiede nel fatto che si utilizza un oggetto Java "Image" come un buffer virtuale di memoria al cui interno, andremo a disegnare gli Sprite nelle posizioni aggiornate nella fase di Update. Solamente al termine di tutta queste modifiche all'immagine, essa verrà copiata nella memoria video per essere visualizzata dall'utente. La memoria video è lenta, e questo metodo evita il fastidioso sfarfallio che otterremmo disegnando tutto direttamente in essa.

## 3.8 Riproduzione di file audio

In questo paragrafo verranno analizzati gli strumenti che il linguaggio Java mette a disposizione dei programmatori per la riproduzione di file audio. Nel progetto è stata gestita la riproduzione di due formati audio: un file in formato MIDI[11] come colonna sonora del gioco ed alcuni file WAW[12] per gli effetti sonori. Come primo esempio immaginiamo di voler riprodurre un file audio MIDI chiamato sinfonia.mid. Il package *javax.sound.midi* della libreria Java contiene tutti gli strumenti che ci servono a tal proposito.

[10] java.awt.Image appartenente alla libreria di Java
[11] Acronimo di Musical Instrument Digital Interface
[12] Acronimo di WAVEform Audio File Format

Le classi coinvolte nella riproduzione di un file MIDI sono le seguenti[34]:

- Sequencer : ha il compito di mettere in esecuzione le Sequence MIDI
- Sequence : è una struttura dati contenente i dati del brano da riprodurre. Una Sequence è riprodotta dal Sequencer.
- MidiSystem : fornisce accesso alle risorse del calcolatore per la riproduzione MIDI

Vediamo ora come utilizzare questi strumenti. Innanzitutto, dobbiamo creare un oggetto della classe File [13], che rappresenta un riferimento alla locazione del file contenente la nostra traccia musicale :

```
File mioFile =new File("sinfonia.mid")
```

Il secondo passo è quello di creare una sequence MIDI del file che abbiamo appena definito. A tale scopo utilizziamo il metodo getSequence() della classe MidiSystem:

```
Sequence sequenza = MidiSystem.getSequence(mioFile)
```

Ora abbiamo bisogno di un Sequencer . Possiamo ottenerlo invocando il metodo getSequencer() della classe MidiSystem in questa maniera:

```
Sequencer sequencer = MidiSystem.getSequencer()
```

Gli ultimi passi da compiere sono tre:

- attivare il sequencer chiamando il metodo open() della classe Sequencer
- associare la sequence al sequencer utilizzando il metodo setSequence della classe Sequencer
- avviare la riproduzione utilizzando il metodo start della classe Sequencer

Quanto detto si può scrivere in questa maniera:

[13] java.io.File nella libreria di Java

```
sequencer.open();
sequencer.setSequence(sequenza);
sequencer.start();
```

Immaginiamo ora di voler riprodurre un file WAV chiamato sinfonia.wav. Il suono analogico è formato da onde sonore mentre il segnale digitale è costruito mediante campioni prelevati(*sample*) dell'ampiezza del segnale analogico. Avremo la migliore qualità mano a mano che il numero di campionamenti per secondo aumenta (*sample rate*)[35]. Nella libreria di Java esiste un package per gestire i suoni campionati come i WAV che si chiama *javax.sound.sampled*. Innanzitutto, dobbiamo creare un oggetto della classe File[14] , che rappresenta un riferimento alla locazione del file contenente la nostra traccia musicale :

```
File mioFile =new File("sinfonia.wav")
```

Ora utilizzeremo alcune classi presenti nel package java nominato prima [36]:

- AudioSystem : permette l'accesso alle risorse del calcolatore interessate dalla riproduzione di file audio e ne consente anche la conversione degli stessi in formati audio diversi.
- AudioInputStream : permette la lettura di un flusso di dati audio in un formato particolare e con una propria lunghezza espressa in numero di campionamenti.
- AudioFormat : serve a specificare alcune informazioni riguardo al flusso di dati audio da leggere come la frequenza di campionamento, la dimensione di ogni campione ed il numero di canali: uno se mono due se stereo
- Clip : è una interfaccia multimediale che rappresenta un particolare tipo di linea audio in cui i dati possono essere caricati prima della riproduzione, invece di essere trasmessi in tempo reale

[14] java.io.File nella libreria di Java

Passando al codice java, vediamo ora come utilizzare questi strumenti. Indico allo stream, qual'è il file da cui leggere i dati.

```
AudioInputStream stream;
stream = AudioSystem.getAudioInputStream(mioFile)
```

Ora occorre raccogliere informazioni sul formato audio del file che vogliamo riprodurre. Utilizzo il metodo getFormat() della classe AudioInputStream.

```
AudioFormat formato = stream.getFormat().
```

Con la parola "*formato*" indichiamo queste informazioni relative al file audio:

- il sample rate : numero di campionamenti per secondo del file campionato
- il numero di canali : l'audio può essere mono o stereo a seconda che sia differenziato o meno per l'autoparlante sinistro e destro del calcolatore
- il numero di byte per rappresentare ogni sample (*frame size*)

Ora che abbiamo un riferimento al file da riprodurre e le informazioni in merito al suo formato, introduciamo il concetto di *Line*[36]. Questa è la nostra interfaccia al sistema audio del calcolatore, possiamo ricevere e mandare dati ad esso. Un particolare tipo di Line, è rappresentata dall'interfaccia *Clip*[36], al cui interno i dati possono essere pre caricati e non letti man mano in tempo reale come una normale Line. In questa maniera la lunghezza dei dati è nota, posso eseguire il brano da qualsiasi punto e ciclicamente il numero di volte che voglio. Per creare un tipo di Clip adeguata al formato del file da riprodurre, devo utilizzare un oggetto della classe DataLine.Info [36] creato in questa maniera:

```
DataLine.Info info = new DataLine.Info(Clip.class,formato)
```

I parametri passati sono il tipo di linea da creare, in questo caso Clip, ed il formato del file da riprodurre. Non ci resta ora che creare la Clip ed avviare

l'esecuzione del file. Utlizziamo il metodo getLine() della classe AudioSystem che restituisce una Clip specifica per la tipologia di file da riprodurre. Il metodo open della classe Clip permette di assegnare il nostro file alla Clip che lo eseguirà una volta chiamato il metodo start() della stessa classe.

```
Clip clip = (Clip) AudioSystem.getLine(info);
clip.open(stream);
clip.start();
```

## 3.9 Gestione degli eventi della tastiera e del mouse

Le applicazioni che prevedono un'interazione con l'utente, devono gestire di norma gli eventi generati da due periferiche: la tastiera oppure il mouse del calcolatore. Nei videogiochi questo argomento è di cruciale importanza, in quanto si deve coinvolgere il giocatore interattivamente nelle fasi del gioco. Nel videogioco *BarInvaders* per esempio, l'utente dovrà calarsi nei panni di un barista che deve difendere il bancone del suo bar. Gli eventi legati alla pressione della barra spaziatrice della tastiera e le frecce direzionali, devono essere gestiti per permettere al barista di lanciare una bottiglia. Vediamo ora gli strumenti che il linguaggio Java mette a disposizione dei programmatori. Tutte le classi che estendono la classe Component[15] ereditano da essa dei metodi molto utili per gli eventi[29]:

- addKeyListener(KeyListener l) : associa ad un oggetto un ascoltatore di eventi relativo alla pressione di tasti della tastiera
- addMouseListener(MouseListener l) : associa ad un oggetto un ascoltatore di eventi relativo alla pressione di tasti del mouse

Analizziamo la gestione degli eventi della tastiera. KeyListener è un'interfaccia che prevede tre metodi[30]:

[15] java.awt.Component appartenente alla libreria Java

- keyPressed(KeyEvent e) : è invocato alla pressione di un tasto della tastiera
- keyReleased( KeyEvent e) : è invocato al rilascio di un tasto della tastiera
- keyTyped( KeyEvent e) : è invocato alla pressione prolungata di un tasto della tastiera

Per poter utilizzare questi metodi in una classe, essa deve implementare l'interfaccia KeyListener e ridefinire al suo interno i tre metodi appena mostrati a seconda delle azioni che vogliamo eseguire al verificarsi degli eventi. Grazie all'oggetto "e" di tipo KeyEvent [31], possiamo risalire al tasto premuto grazie al metodo getKeyCode(), che restituisce il numero intero associato per convenzione al tasto premuto.

La gestione degli eventi relativi al mouse è speculare a tutto quello appena detto per la tastiera. MouseListener[32] è un'interfaccia che prevede cinque metodi:

- mouseClicked(MouseEvent e) : invocato quando clicchiamo un tasto del mouse
- mouseEntered(MouseEvent e) : invocato quando la freccia del cursore passa sopra un componente grafico
- mouseExited(MouseEvent e) : invocato quando la freccia del cursore esce da un componente grafico
- mousePressed(MouseEvent e) : invocato quando premiamo un tasto sopra ad un componente grafico
- mouseReleased(MouseEvent e): invocato all'atto del rilascio di un tasto sopra un componente grafico

Per poter utilizzare questi metodi in una classe, essa deve implementare l'interfaccia MouseListener e ridefinire al suo interno i cinque metodi appena mostrati a seconda delle azioni che vogliamo eseguire al verificarsi degli

eventi . Grazie all'oggetto "e" di tipo MouseEvent[33], possiamo ottenere utili informazioni grazie ai metodi di questa classe quali getX() e getY() che restituiscono le coordinate del puntatore del mouse nello schermo.

# Capitolo 4

# Bar invaders: implementazione

L'implementazione è l'ultima fase del processo di sviluppo di un'applicazione. In questo capitolo sono riportate alcune immagini del programma BarInvaders durante l'esecuzione.

## 4.1 BarInvaders : esecuzione

All'avvio dell'applicazione, viene mostrata la schermata iniziale di figura 4.1 che ha la funzione di mostrare la classifica e di permettere all'utente di autenticarsi.


Figura 4.1: BarInvaders : avvio dell'applicazione

Se dalla schermata iniziale di figura 4.1 l'utente seleziona il pulsante "CLASSIFICA", gli verrà mostrata la classifica riportante i cinque migliori risultati ottenuti nel gioco. La visualizzazione può essere interrotta selezionando il pulsante "CHIUDI CLASSIFICA".


Figura 4.2: BarInvaders : la classifica

Se dalla schermata principale di figura 4.1 l'utente seleziona il pulsante "GIOCA", gli verrà richiesto di inserire il proprio nome identificativo prima di poter proseguire nel gioco.


Figura 4.3: BarInvaders : autenticazione di un utente

Il nome scelto per l'autenticazione deve avere una lunghezza compresa tra 1 e 15 caratteri. Nel caso non venga rispettata questa regola, un messaggio come quello di figura 4.4 avvertirà l'utente che sarà costretto a ripetere l'inserimento dei propri dati.


Figura 4.4: BarInvaders : errore nell'autenticazione

In figura 4.5 è mostrata la schermata di gioco. I clienti stanno scendendo in direzione del bancone ed il barista ha appena lanciato una bottiglia. Nella parte più in alto della schermata vi sono due contatori: uno per le "vite" rimaste all'utente ed uno per il punteggio della partita.


Figura 4.5: BarInvaders : schermata principale del gioco

La figura 4.6 mostra la collisione di un cliente con la bottiglia lanciata dal barista. Viene simulata un'esplosione.

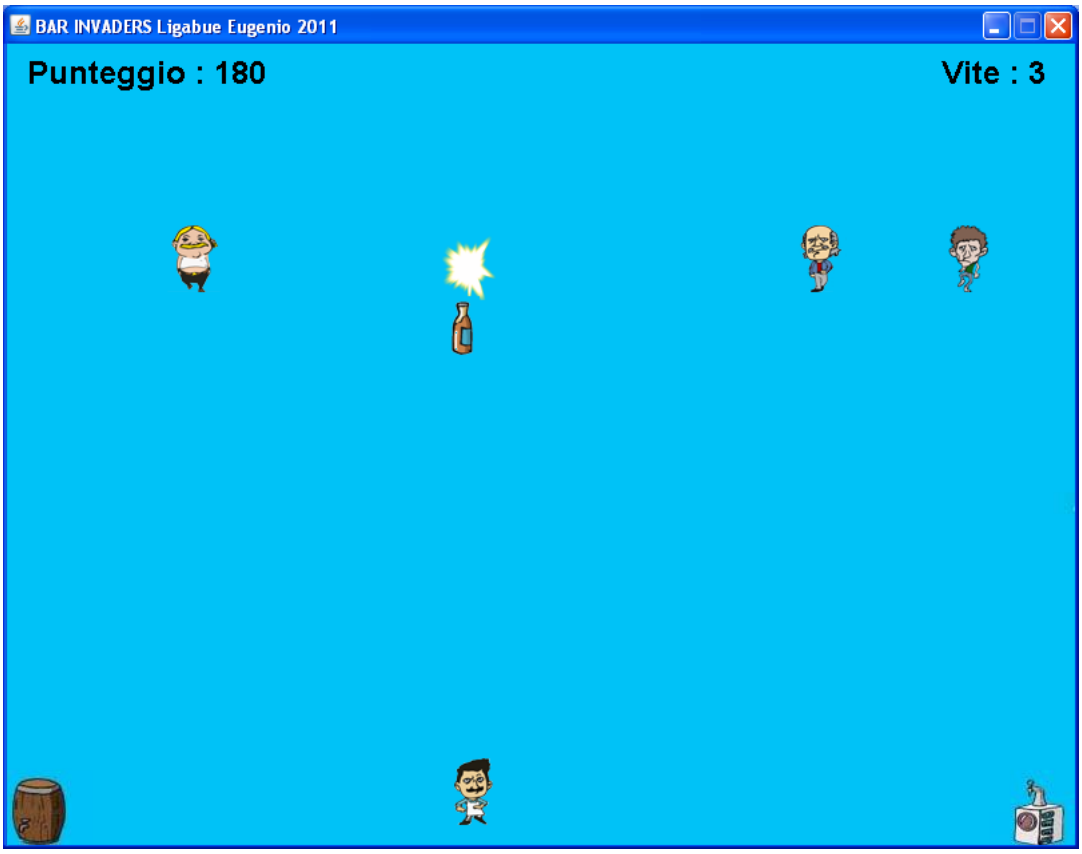



Figura 4.6: BarInvaders : eliminazione di un cliente

La figura 4.7 mostra una collisione tra un cliente ed il barista. In questo caso, a differenza dell'immagine 4.6, nell'esplosione è coinvolto anche il barista.


Figura 4.7: BarInvaders : collisione di un cliente col barista

L'immagine 4.8 mostra una collisione dei clienti con il bancone del bar. La partita in questo caso termina perchè l'utente ha terminato le "vite" a propria disposizione. Un messaggio segnala all'utente che la partita è finita. Premendo il pulsante "OK" , l'utente verrà riportato nella schermata di figura 4.1.

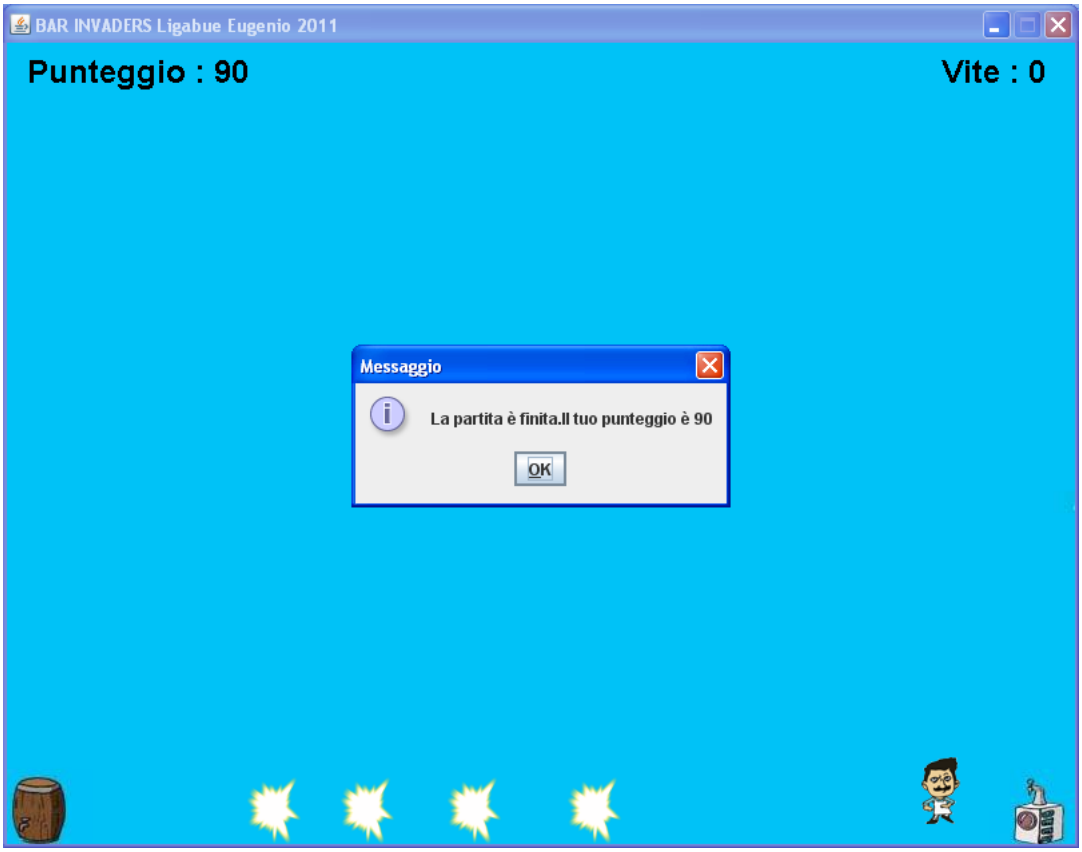



Figura 4.8: BarInvaders : collisione di clienti col bancone del bar

# Capitolo 5

# Conclusioni

Al termine di tutto il lavoro che il progetto di questa tesi ha comportato, è stato implementato un gioco per calcolatore scritto in linguaggio Java chiamato *BarInvaders*. Una caratteristica peculiare di questo linguaggio che lo differenzia da altri come il celebre C++, risiede in uno degli argomenti di cui si è parlato all'interno della tesi: la Macchina Virtuale Java, in inglese Java Virtual Machine( JVM). Ogni calcolatore che abbia installato la JVM, potrà eseguire il gioco BarInvaders, garantendo così la portabilità dell'applicazione in piattaforme diverse. Non a caso, uno degli slogan di Sun Microsystem, l'azienda americana ideatrice di Java, è "Write once, run everywhere" ossia "scrivi una volta ed esegui ovunque". Per quanto riguarda le prestazioni, l'introduzione della tecnologia *HotSpot* ha notevolmente migliorato le performance per quelle applicazioni che richiedono tempi di esecuzione molto rapidi. Andrew Davison, autore di diversi libri e diverse pubblicazioni riguardanti i videogiochi, afferma che la versione 1.0 del linguaggio Java, era dalle venti alle quaranta volte più lenta del C++[25]. La versione 1.6 invece sarebbe più lenta di sole 1.1 volte. Davison sostiene che il merito di questo enorme aumento di velocità è dovuto all'introduzione della tecnologia HotSpot da parte di Java.

Affinché un'applicazione garantisca elevate prestazioni, occorre anche che il programmatore si impegni ad utilizzare in modo efficente le tecnologie a

disposizione. L'utilizzo oculato delle risorse a disposizione in un sistema, specialmente se scarse, deve essere una priorità nelle scelte implementative in fase di progetto. Per monitorare quali parti di codice utilizzano eccessivamente la memoria, e tante altre utili statistiche sulle risorse impiegate da una applicazione, è fondamentale l'utilizzo degli strumenti chiamati *Java Profiler*. Questo termine è utilizzato per identificare tutte quelle applicazioni gratuite o a pagamento che permettono al programmatore di monitorare le performance della propria applicazione.

Per quando riguarda le prospettive future del progetto, il gioco BarInvaders può essere modificato adattando l'applicazione per essere eseguita in dispositivi mobili, quali gli smartphone oppure i tablet pc. Il sistema operativo di queste apparecchiature deve però supportare le applicazioni Java. Come abbiamo avuto modo di vedere nel corso della tesi, il sistema operativo Android ha tutte le carte in regola per imporsi nel mercato. La caratteristica che utilizzi Java come linguaggio di riferimento, lo rende il candidato ideale per cui adattare il gioco BarInvaders nei dispositivi mobili.

# Bibliografia e fonti dal Web


[1] http://it.wikipedia.org/wiki/Spacewar!

[2] http://en.wikipedia.org/wiki/PDP-1

[3] http://pdp-1.computerhistory.org/pdp-1/index.php

[4] http://www.aesvi.it/cms/attach/editor/Rapporto_Annuale_2010.pdf

[5] Cay S. Horstmann - Concetti e fondamenti di Java, Apogeo , 2010

[6] http://www.oracle.com/technetwork/java/whitepaper-135217.html

[7] http://java.html.it/articoli/leggi/3371/

[8] http://it.wikipedia.org/wiki/Macchina_virtuale_Java

[9] http://java.sun.com/developer/technicalArticles/Networking/HotSpot/inlining.html

[10] http://download.oracle.com/javase/6/docs/api/java/lang/System.html#gc()

[11] http://www.ej-technologies.com/products/jprofiler/overview.html

[12] http://java.sun.com/developer/technicalArticles/J2SE/jconsole.html

[13] http://download.oracle.com/javase/6/docs/api/java/lang/management/package-summary.html

[14] http://download.oracle.com/javase/6/docs/api/javax/swing /package-summary.html

[15] http://download.oracle.com/javase/6/docs/api/java/awt /package-summary.html

[16] http://download.oracle.com/javase/tutorial/extra/fullscreen/index.html

[17] http://download.oracle.com/javase/6/docs/api/java/awt /GraphicsEnvironment.html

[18] http://download.oracle.com/javase/6/docs/api/java/awt /GraphicsDevice.html

[19] http://download.oracle.com/javase/6/docs/api/java/awt /GraphicsConfiguration.html

[20] http://download.oracle.com/javase/6/docs/api/java/awt /DisplayMode.html

[21] http://it.wikipedia.org/wiki/Martin_Cooper

[22] http://it.wikipedia.org/wiki/Smartphone

[23] http://www.millennialmedia.com/research/

[24] http://it.wikipedia.org/wiki/Android

[25] Andrew Davison - Killer Game Programming in Java, O'Reilly ,2005

[26] David Brackeen, Bret Barker, Laurence Vanhelsuwé - Developing Games in Java , New Riders , 2004

[27] http://it.wikipedia.org/wiki/Caso_d%27uso_%28informatica%29

[28] Craig Larman - Applicare UML e i pattern , Pearson , Terza Edizione

[29] http://download.oracle.com/javase/6/docs/api/java/awt
/Component.html

[30] http://download.oracle.com/javase/6/docs/api/java/awt
/event/KeyListener.html

[31] http://download.oracle.com/javase/6/docs/api/java/awt
/event/KeyEvent.html

[32] http://download.oracle.com/javase/6/docs/api/java
/awt/event/MouseListener.html

[33] http://download.oracle.com/javase/6/docs/api/java/awt
/event/MouseEvent.html

[34] http://download.oracle.com/javase/6/docs/api/javax/sound
/midi/package-summary.html

[35] http://it.wikipedia.org/wiki/WAV

[36] http://download.oracle.com/javase/6/docs/api/javax/sound
/sampled/package-summary.html

[37] http://download.oracle.com/javase/6/docs/api/java/awt/Image.html