

POLITECNICO DI TORINO

Laurea di 1° Livello in Ingegneria Gestionale
Classe L-8 Ingegneria dell'Informazione



Applicazione per software-house dell'industria videoludica

Relatore

Prof. Fulvio Corno

Candidato

Serra Alessio

Anno Accademico
2018/2019

INDICE

1	Proposta di progetto	4
1.1	Studente proponente	4
1.2	Titolo della proposta	4
1.3	Descrizione del problema proposto	4
1.4	Descrizione della rilevanza gestionale del problema	4
1.5	Descrizione dei data-set per la valutazione	4
1.6	Descrizione preliminare degli algoritmi coinvolti.....	5
1.7	Descrizione preliminare delle funzionalità previste per l'applicazione software	5
2	Descrizione del problema affrontato	6
3	Descrizione del data-set utilizzato	7
4	Descrizione degli algoritmi utilizzati.....	9
4.1	Primo algoritmo	9
4.2	Secondo algoritmo.....	11
5	Diagramma delle classi principali.....	13
6	Videate dell'applicazione con risultati sperimentali ottenuti	14
7	Link al video YouTube	18
8	Valutazione dei risultati e conclusioni.....	18

❖ 1 Proposta di progetto

1.1 Studente proponente

s233789 Serra Alessio

1.2 Titolo della proposta

Applicazione per software-house dell'industria videoludica

1.3 Descrizione del problema proposto

Si intende realizzare una applicazione che verifichi l'adeguatezza di un investimento riguardante lo sviluppo(e quindi le relative spese) di un videogioco da pubblicare in una determinata zona del mondo (Europa, America, Asia, Resto del mondo) basandosi su dati relativi a vendite, recensioni e anno di pubblicazione di videogiochi, presenti all'interno del database, appartenenti allo stesso genere o alla stessa saga, lasciando all'utente la libertà di settare le condizioni di analisi che ritiene più opportune.

1.4 Descrizione della rilevanza gestionale del problema

L'industria videoludica rappresenta oggi uno dei settori economici in più rapida espansione in grado di competere, per fatturato e per utili, con le tradizionali industrie dell'entertainment e del software.

E' quindi fondamentale riuscire ad avere una stima, basata su dati reali e concreti, riguardo la possibile buona riuscita del progetto in questione, visto le ingenti risorse che occorrono nelle fasi di studio, sviluppo e pubblicazione di un videogioco al fine di non sbagliare quello che oggi è ormai davvero un grosso investimento.

1.5 Descrizione dei data-set per la valutazione

Il database che sarà utilizzato dall'applicazione si può trovare al seguente link:

<https://www.kaggle.com/rush4ratio/video-game-sales-with-ratings/>

Esso è costituito da circa 17mila videogiochi pubblicati dopo il 1980, in cui ogni videogioco è caratterizzato da 16 colonne in cui possiamo trovare nomi, anno di pubblicazione, dati di vendita sia globali sia per zona, publisher e ratings estratti dal famosissimo sito MetaCritic, in cui sono sia utenti sia testate giornalistiche a determinare le valutazioni.

Il formato del file è in CSV quindi dovrà essere convertito in SQL e ripulito di eventuali dati errati/mancanti per essere utilizzato.

1.6 Descrizione preliminare degli algoritmi coinvolti

Gli algoritmi coinvolti sono due:

- L'utente (cioè la software-house) seleziona il genere di videogame che vuole produrre, immette l'anno da cui vuole far partire l'analisi e tramite uno slider sceglie quanto peso dare alle vendite e quale peso dare alle valutazioni. L'algoritmo restituisce la zona geografica dove la pubblicazione del videogioco restituisce il coefficiente più alto (coefficiente valutato tramite una media pesata di vendite e valutazioni);
- ALGORITMO RICORSIVO: caso in cui si vogliano pubblicare PIU' progetti, tenendo conto di un BUDGET limite. L'utente inserisce il numero di generi che vuole in output, la regione geografica e l'anno da cui far partire l'analisi e il budget massimo complessivo per lo sviluppo degli N progetti coinvolti. L'algoritmo ricorsivo restituisce la miglior combinazione di generi relativa a quella zona geografica (che può essere formata anche solo da 1 genere) tenendo conto degli input dell'utente e soprattutto del budget massimo disponibile.

1.7 Descrizione preliminare delle funzionalità previste per l'applicazione software

L'applicazione avrà più interfacce grafiche sulle quali sarà possibile gestire tutti i metodi messi a disposizione del programma tramite l'uso di input testuali, box a scelta, bottoni, slider, ecc.

Il programma risponderà scrivendo i risultati in un'area di testo ed eventualmente con l'aggiunta di immagini e diagrammi.

Saranno presenti anche bottoni del tipo RESET per eliminare tutte le preferenze inserite dall'utente e altri bottoni per potersi spostare comodamente da un'interfaccia ad un'altra.

❖ 2 Descrizione del problema affrontato

Il processo tecnologico degli ultimi 30 anni ha portato benefici diretti a tutti i settori dell'intrattenimento, permettendo di raggiungere standard qualitativi che sembravano impensabili fino a pochi anni fa.

L'industria videoludica è proprio uno di questi settori: basti pensare ai videogiochi degli anni '80, creati da studi di programmazione relativamente piccoli, che si presentavano graficamente composti da pochi pixel e da scenari statici, tutto ciò dovuto alla poca potenza di elaborazione disponibile all'epoca. Oggi invece gli studi di programmazione sono composti da centinaia di persone: questo permette di sviluppare videogiochi che graficamente sfiorano il fotorealismo e permettono livelli di interazione altissimi all'utente, grazie alla potenza di elaborazione raggiunta in questi anni.

La qualità raggiunta non è però gratis: all'aumentare degli standard qualitativi raggiunti, sono aumentati in maniera proporzionale anche i costi da sostenere. Infatti, sviluppare un videogioco AAA raggiunge, e a volte supera, i costi di produzione di un colosso hollywoodiano: da questo dato possiamo comprendere la rilevanza gestionale del problema.

Riuscire ad avere una stima sui costi di produzione e sulla buona riuscita del progetto è fondamentale per non rischiare di avere un investimento sbagliato. In particolare, software-house molto grosse hanno la possibilità di sviluppare e pubblicare più videogiochi nello stesso periodo. Avere un'applicazione che tramite algoritmi di valutazione, basati su dati reali, riesce a suggerire quali tipi di investimenti (cioè quali generi di videogiochi sono più venduti e apprezzati in una certa area del mondo) possono avere un maggiore riscontro in termini costi/ricavi è davvero importante.

Grazie all'applicazione sviluppata è quindi possibile affrontare i seguenti problemi:

- *Stilare una classifica di preferenza di pubblicazione basata sulle diverse preferenze delle diverse macro-aree del mondo:* la classifica viene stilata in base alle preferenze inserite in input con cui viene calcolato un coefficiente che determina la classifica finale.
- *Ottenere la miglior combinazione di investimenti rispetto al numero di progetti da voler realizzare e il budget da impiegare disponibile:* la miglior combinazione viene ottenuta attraverso un algoritmo ricorsivo che confronta tutte le combinazioni in modo da ottimizzare il budget, cioè trovare la combinazione che minimizza lo scarto da budget di produzione e budget massimo disponibile.

❖ 3 Descrizione del data-sets utilizzato

Il data-sets utilizzato è disponibile all'indirizzo:

<https://www.kaggle.com/rush4ratio/video-game-sales-with-ratings/>, dove può essere scaricato in formato nativo CSV.

Per essere utilizzato all'interno dell'applicazione è stato convertito in formato SQL con la seguente sintassi per la creazione delle tabelle:

```
CREATE DATABASE IF NOT EXISTS `datavg` /*!40100 DEFAULT CHARACTER SET latin1 */;
USE `datavg`;

-- Dump della struttura di tabella datavg.data
CREATE TABLE IF NOT EXISTS `data` (
  `Name` varchar(100) DEFAULT NULL,
  `Platform` varchar(50) DEFAULT NULL,
  `Year_Release` int(11) DEFAULT NULL,
  `Genre` varchar(50) DEFAULT NULL,
  `Publisher` varchar(50) DEFAULT NULL,
  `NA_Sales` double DEFAULT NULL,
  `EU_Sales` double DEFAULT NULL,
  `JP_Sales` double DEFAULT NULL,
  `OTHER_Sales` double DEFAULT NULL,
  `GLOBAL_Sales` double DEFAULT NULL,
  `CRITIC_Ratings` int(11) DEFAULT NULL,
  `USER_Ratings` double DEFAULT NULL,
  KEY `Name` (`Name`,`Platform`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Dump della struttura di tabella datavg.genreprice
CREATE TABLE IF NOT EXISTS `genreprice` (
  `Genre` varchar(50) DEFAULT NULL,
  `PriceForUnit` double DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

All'interno di **HeidiSQL** i dati della prima tabella vengono rappresentati nel seguente modo:

Name	Platform	Year_Release	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	OTHER_Sales	GLOBAL_Sales	CRITIC_Ratings	USER_Ratings
Grand Theft Auto IV	X360	2.008	Action	Take-Two Interactive	6,76	3,07	0,14	1,03	11,01	98	7,9
Grand Theft Auto IV	PS3	2.008	Action	Take-Two Interactive	4,76	3,69	0,44	1,61	10,5	98	7,5
Tony Hawk's Pro Sk...	PS	2.000	Sports	Activision	3,05	1,41	0,02	0,2	4,68	98	7,7
SoulCalibur	DC	1.999	Fighting	Namco Bandai Games	0	0	0,34	0	0,34	98	8,8
Grand Theft Auto V	PS3	2.013	Action	Take-Two Interactive	7,02	9,09	0,98	3,96	21,04	97	8,2

Si hanno a disposizione le statistiche riguardanti le **vendite**, sia suddivise per regioni geografiche (Nord America, Europa, Giappone, Resto del Mondo) sia per totale globale, e le **valutazioni**, sia assegnate dalle maggiori testate giornalistiche del settore sia votate dagli utenti attraverso il famoso sito MetaCritic.

Sono di rilievo inoltre gli attributi **nome** e **piattaforma**, che costituiscono la chiave primaria, oltre che **genere** e **data di rilascio** al pubblico.

La seconda tabella appare invece nel seguente modo all'interno di HeidiSQL:

Genre	PriceForUnit
Action-Adventure	80
Platform	75
Sports	70
Shooter	70
Puzzle	65
Action	60
Racing	60
Fighting	55
Misc	50
Role-Playing	50
Adventure	50
Music	40
MMO	40
Strategy	35
Simulation	30
Party	20

In questa tabella sono riportati i **generi** di videogames col relativo **costo** di produzione per unità.

In questo modo è possibile valutare il costo medio di produzione di un dato genere moltiplicando le vendite medie, in una determinata regione e a partire da un determinato anno, per il costo/unità.

❖ 4 Descrizione degli algoritmi utilizzati

4.1 Primo algoritmo

L'applicazione è stata sviluppata in linguaggio Java con il supporto delle interfacce **JavaFX**. Sono inoltre implementati il pattern **MVC** (*Model View Controller*) ed il pattern **DAO** (*Data Access Object*).

La struttura dell'applicazione software è divisa in tre package:

```

v VGDataTool [SerraAlessio master]
  v src
    v it.polito.tdp.vgdatatool.controller
      > BestMixController.java
      > DataAnalysisSalesController.java
      > Main.java
      > VGDataToolController.java
      application.css
      BestMix.fxml
      DataAnalysisSales.fxml
      VGDataTool.fxml
    v it.polito.tdp.vgdatatool.db
      > DBConnect.java
      > TestDBConnect.java
      > VideogameDAO.java
    v it.polito.tdp.vgdatatool.model
      > Genre.java
      > Model.java
      > Videogame.java
      > Zone.java
```

- **Controller**: contiene la classe **Main** per l'avvio dell'applicazione, le tre classi Controller: **VGDataTool**, **BestMix**, **DataAnalysisSales** che definiscono i metodi che permettono l'interazione tra l'utente e i rispettivi file **FXML** per la definizione dell'interfaccia grafica.

- **DB**: contiene la classe **DBConnect** utilizzata per la connessione al database e la classe **VideogameDAO** che attraverso query SQL permette il caricamento dei dati presenti nel database all'interno dell'applicazione Java. È inoltre presente la classe di prova **TestDBConnect** che ha il compito di testare la connessione al database.

- **Model**: contiene l'*omonima* classe **Model** che possiede tutta la logica applicativa del software. Sono inoltre presenti le classi Java Bean: **Genre**, **Model**, **Videogame**, **Zone**.

Il primo algoritmo viene richiamata dall'interfaccia grafica *DataAnalysisSales.fxml*.

Dopo aver selezionato il genere, l'anno di rilascio e settato lo slider di preferenza (che oscilla tra vendite e valutazioni), alla pressione del tasto *Analyze* viene invocato il metodo `public List<Zone> getBestZone()` che restituisce la lista di zone geografiche ordinate per indice, calcolato in base alle preferenze immesse in input dall'utente.

L'algoritmo (il cui codice è riportato nella pagina seguente) interroga il database facendosi restituire la lista di videogames, di un dato genere e pubblicati a partire da un dato anno. Poi vengono settati i pesi relativi alle vendite e alle valutazioni e creati gli *oggetti* rappresentanti le zone del mondo. Vengono poi estratti i dati di interesse per ciascuna zona geografica al fine di calcolare il relativo indice, pesato rispetto alle preferenze inserite in input. Infine, viene creata la lista di Zone e, prima di essere restituita, viene riordinata per indice decrescente.

```

public List<Zone> getBestZone(Genre g,int year, double preferences){

    VideogameDAO dao = new VideogameDAO();
    List<Zone> result = new ArrayList<>();

    //Get all videogames of that genre and following that year
    List<Videogame> videogames = dao.getVideogames(g, year);

    //Set the weight about ratings and sales
    double weightR = preferences;
    double weightS = 1-preferences;

    //Create different zones
    Zone na = new Zone("North America",0.0,0.0);
    Zone eu = new Zone("Europe",0.0,0.0);
    Zone jp = new Zone("Japan",0.0,0.0);
    Zone row = new Zone("Rest Of World",0.0,0.0);

    //Get datas for index
    for ( Videogame v : videogames) {
        //NA
        na.sumSales(v.getNA_sales());
        na.sumRatings(v.getCriticR()/10 + v.getUserR());
        //EU
        eu.sumSales(v.getEU_sales());
        eu.sumRatings(v.getCriticR()/10 + v.getUserR());
        //JP
        jp.sumSales(v.getJP_sales());
        jp.sumRatings(v.getCriticR()/10 + v.getUserR());
        //ROW
        row.sumSales(v.getOTHER_sales());
        row.sumRatings(v.getCriticR()/10 + v.getUserR());
    }

    //Calculating index
    na.setAvgSales(na.getAvgSales()/videogames.size());
    na.setAvgRatings(na.getAvgRatings()/(videogames.size()*2));
    double NAindex = na.getAvgSales()*weightS + na.getAvgRatings()*weightR;
    na.setIndex(NAindex);

    eu.setAvgSales(eu.getAvgSales()/videogames.size());
    eu.setAvgRatings(eu.getAvgRatings()/(videogames.size()*2));
    double EUindex = eu.getAvgSales()*weightS + eu.getAvgRatings()*weightR;
    eu.setIndex(EUindex);

    jp.setAvgSales(jp.getAvgSales()/videogames.size());
    jp.setAvgRatings(jp.getAvgRatings()/(videogames.size()*2));
    double JPindex = jp.getAvgSales()*weightS + jp.getAvgRatings()*weightR;
    jp.setIndex(JPindex);

    row.setAvgSales(row.getAvgSales()/videogames.size());
    row.setAvgRatings(row.getAvgRatings()/(videogames.size()*2));
    double ROWindex = row.getAvgSales()*weightS + row.getAvgRatings()*weightR;
    row.setIndex(ROWindex);

    result.add(na);
    result.add(eu);
    result.add(jp);
    result.add(row);

    //Order for the index
    for (int i=0;i<result.size()-1;i++) {
        if (result.get(i).getIndex()<result.get(i+1).getIndex()) {
            Zone zone = result.get(i);
            result.set(i, result.get(i+1));
            result.set(i+1, zone);
        }
    }

    return result;
}

```

4.2 Secondo algoritmo

Il secondo algoritmo implementato è un algoritmo ricorsivo che utilizza altri metodi al suo interno.

Il codice del metodo che avvia la *ricorsione* è il seguente:

```
public List<Genre> recursion(int lenght, int budget, int year, String zone){

    //Collego Model e DAO
    VideogameDAO dao = new VideogameDAO();
    this.best = new ArrayList<>();

    //Clear by old recursion
    if (!allGenres.isEmpty()) allGenres.clear();

    //Add all genres need for combination
    for (int i=0; i<lenght; i++)
        allGenres.addAll(dao.getGenresRecursion(year, zone));

    //Creo soluzione parziale vuota
    List<Genre> partial = new ArrayList<Genre>();

    //Azzero best
    this.best.add(new Genre("Default", 0.0, 0.0));

    //Start recursion (level 0)
    sub_recursion(partial, 0, lenght, budget);

    //ritorno la miglior sequenza trovata
    return best;
}
```

Per prima cosa viene ripulita la memoria da vecchie *ArrayList* utilizzate da algoritmi ricorsivi precedenti. In seguito, viene interrogato il database per ottenere le informazioni necessarie riguardo i *Genre*, i quali vengono inseriti *i-volte* all'interno dell'array che sarà passato al metodo ricorsivo **sub_recursion** per calcolare la combinazione di elementi *i-esima*: avviene quindi avviata la ricorsione di livello 0.

Infine, il metodo restituirà l'*ArrayList* **best**, il quale contiene la miglior combinazione trovata in base al budget inserito.

Il metodo ricorsivo vero è programmato con il seguente codice:

```
public void sub_recursion(List<Genre> partial, int level, int lenght, int budget){

    //FINAL CASE

    if (level == lenght) {

        //Choose the best that maximize the budget
        if (getListPrice(partial) > getListPrice(this.best)) {

            this.best = new ArrayList<>(partial);
            return;
        }
    }
}
```

```

    }

    //INTERMEDIATE CASE
    for (Genre gen : allGenres ) {

        if ( tryBudget(partial, budget, gen)==true ) {

            partial.add(gen);

            sub_recursion(partial, level+1, lenght, budget);

            //backtracking
            partial.remove(partial.size()-1);
        }

        else return;
    }
}

```

La condizione di terminazione imposta si basa sul livello della ricorsione e sulla lunghezza dell'array considerato: se sono uguali si entra nel primo *ciclo IF* e si verifica se il budget della combinazione trovata è *più ottimizzato* del budget della combinazione best attuale. Se quest'ultima condizione è verificata, si aggiorna il best con la nuova combinazione e si chiude il rispettivo ciclo ricorsivo.

Come condizione di verifica intermedia si verifica se il prossimo genere da aggiungere sfiora il budget: se la condizione è verificata l'algoritmo chiude l'attuale ciclo ricorsivo, altrimenti va avanti fino a soddisfare la condizione di terminazione.

Gli altri metodi coinvolti nell'algoritmo ricorsivo, per la gestione del budget, sono i seguenti:

```

public boolean tryBudget(List<Genre> partial,int budget,Genre test) {

    //First step of recursion
    if (partial.size()==0) return true;

    //Get price of the test
    double priceTest = test.getAvgSales()*1000*test.getPrice();

    //Get price of the partial
    double pricePartial = getListPrice(partial);

    if ( pricePartial+priceTest <= budget ) return true;

    //else
    return false;
}

public double getListPrice(List<Genre> listPrice) {

    double priceList = 0.0;

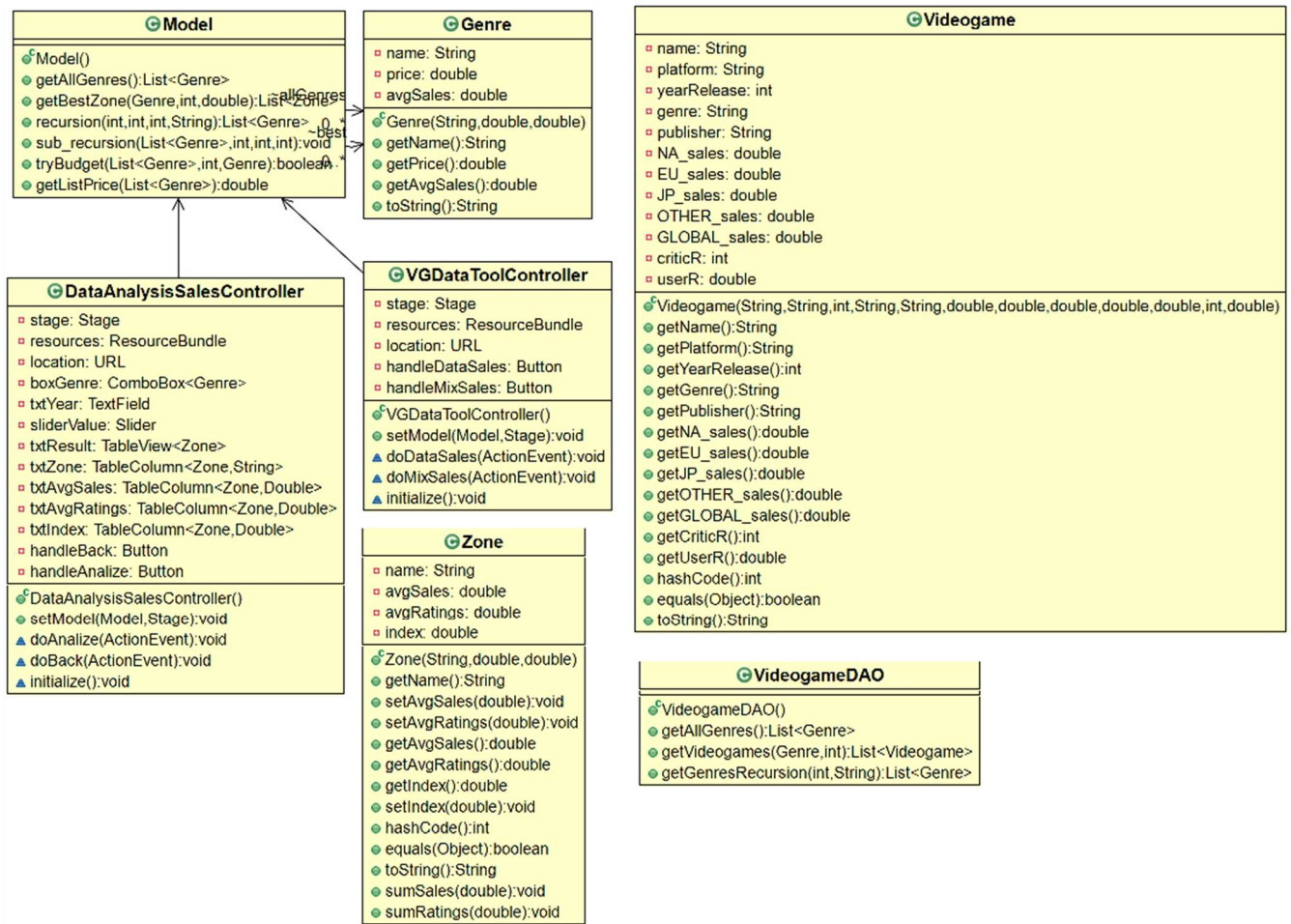
    //First case for best
    if (listPrice.isEmpty()) return 0.0;

    for (Genre g : listPrice) {
        priceList = priceList + (g.getAvgSales()*1000*g.getPrice());
    }
    return priceList;
}

```

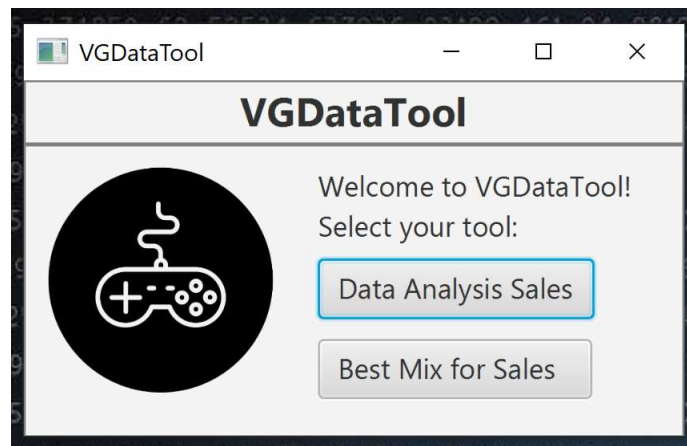

❖5 Diagramma classi principali

L'applicazione è stata sviluppata secondo il pattern MVC (Model-View-Controller) ed il pattern DAO (Data Access Object). Nell'immagine seguente viene illustrato il diagramma delle classi principali:

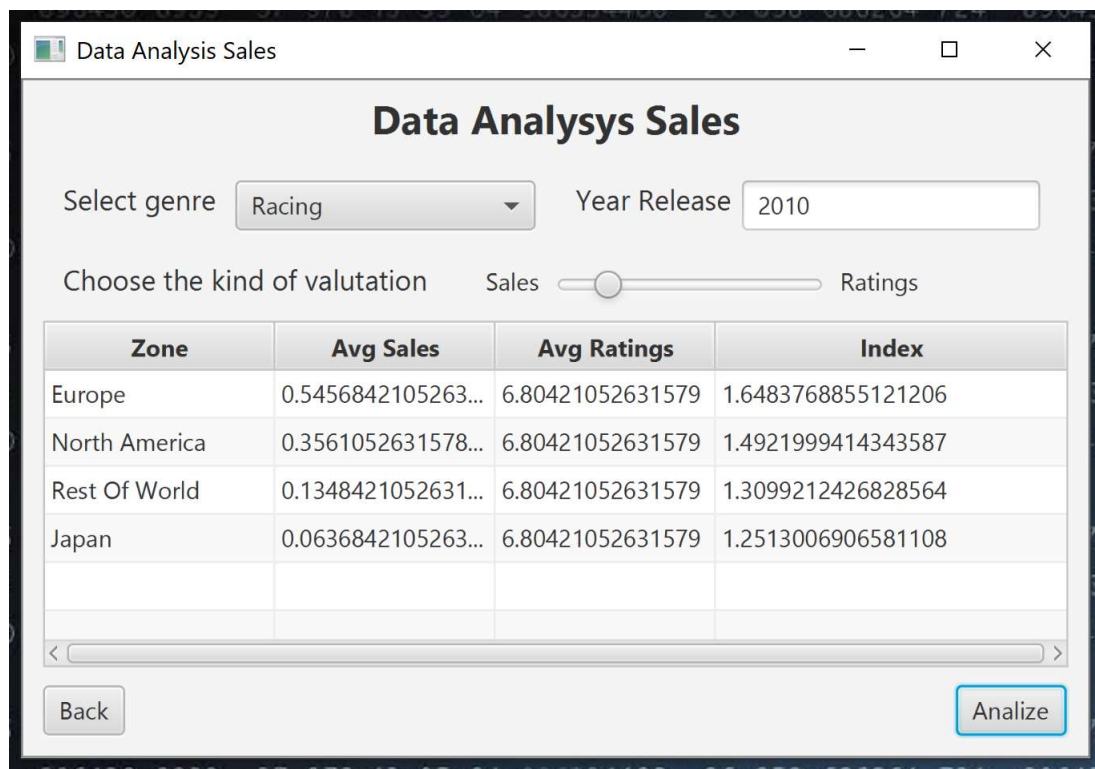


❖6 Videate dell'applicazione con risultati sperimentali ottenuti

All'avvio dell'applicazione si presenta la prima interfaccia grafica, la quale permette di accedere ad uno dei due algoritmi presenti nel programma tramite la pressione dei tasti *Data Analysis Sales* o *Best Mix for Sales*.



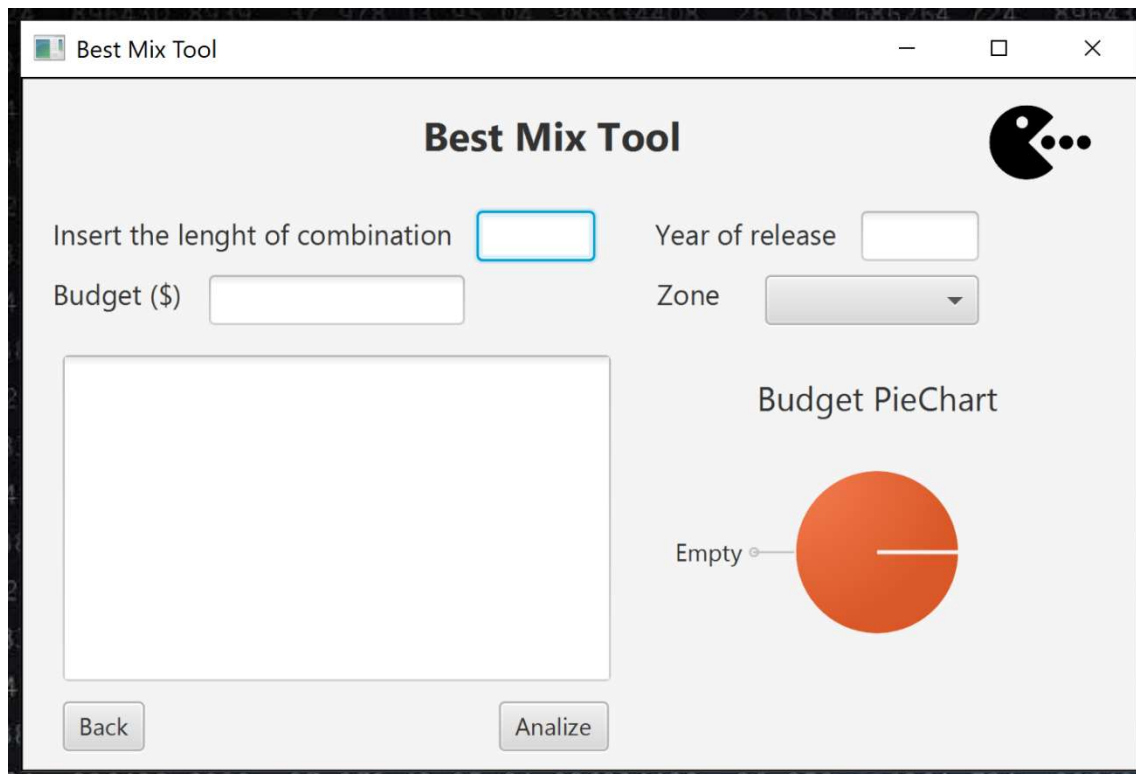
Selezionando il bottone *Data Analysis Sales*, si apre la seconda interfaccia grafica che si presenta in questo modo:



Zone	Avg Sales	Avg Ratings	Index
Europe	0.5456842105263...	6.80421052631579	1.6483768855121206
North America	0.3561052631578...	6.80421052631579	1.4921999414343587
Rest Of World	0.1348421052631...	6.80421052631579	1.3099212426828564
Japan	0.0636842105263...	6.80421052631579	1.2513006906581108

Dopo aver immesso i dati in input, viene restituita la classifica delle zone del mondo in cui è più conveniente pubblicare il videogames. L'esempio sopra è stato ottenuto inserendo i seguenti dati: *Racing*, *2010*, *Sales* (circa 0.8 come peso).

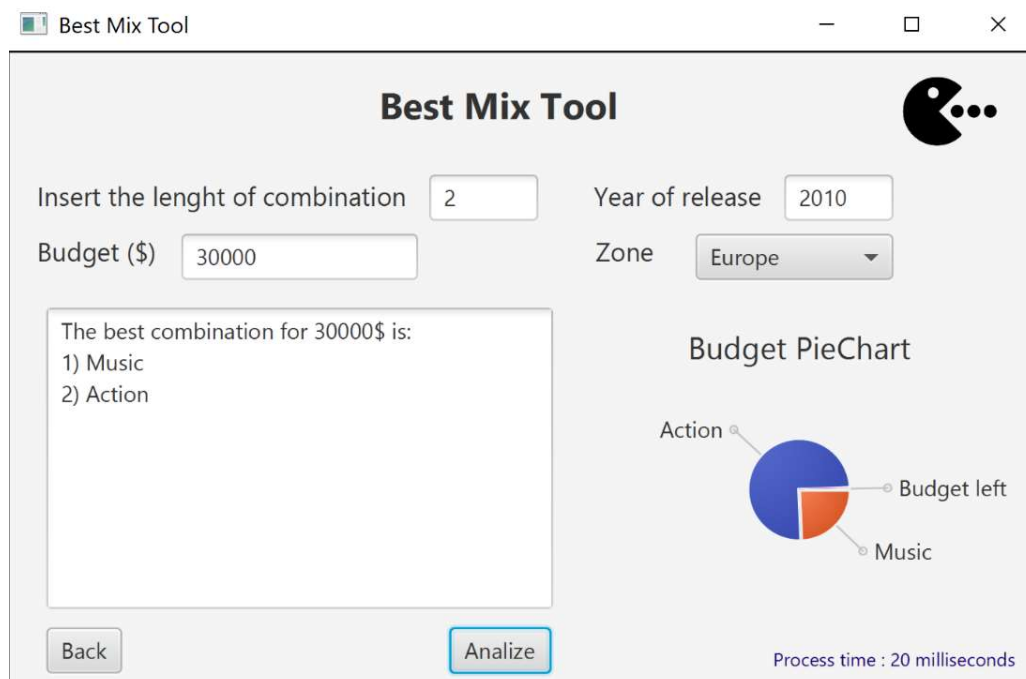
Selezionando invece il bottone *Best Mix For Sales*, si aprirà la seguente interfaccia grafica:



The screenshot shows the 'Best Mix Tool' window. It has a title bar with standard window controls. The main area contains the title 'Best Mix Tool' and a logo. Below the title, there are four input fields: 'Insert the lenght of combination' (with a blue border), 'Year of release', 'Budget (\$)', and 'Zone' (a dropdown menu). A large empty rectangular box is on the left. On the right, there is a 'Budget PieChart' which is a solid orange circle with a label 'Empty' pointing to it. At the bottom, there are 'Back' and 'Analyze' buttons.

Come è possibile notare, prima dell'elaborazione dell'algoritmo, l'area di testo dei risultati è vuota e il diagramma a torta è settato ad "Empty" di default.

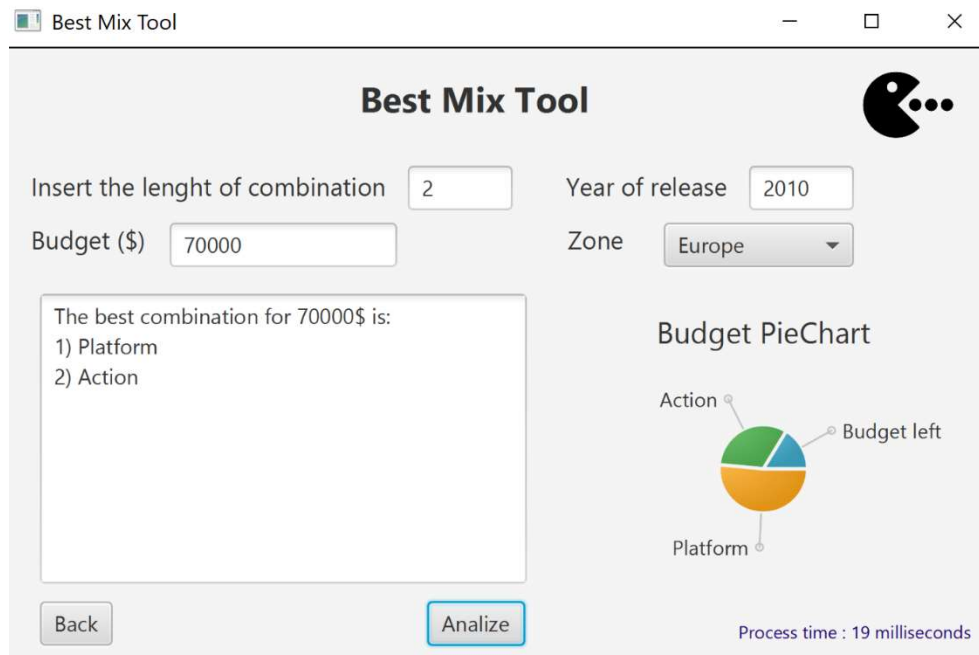
Inserendo diverse combinazioni di dati in input, otteniamo i seguenti risultati:



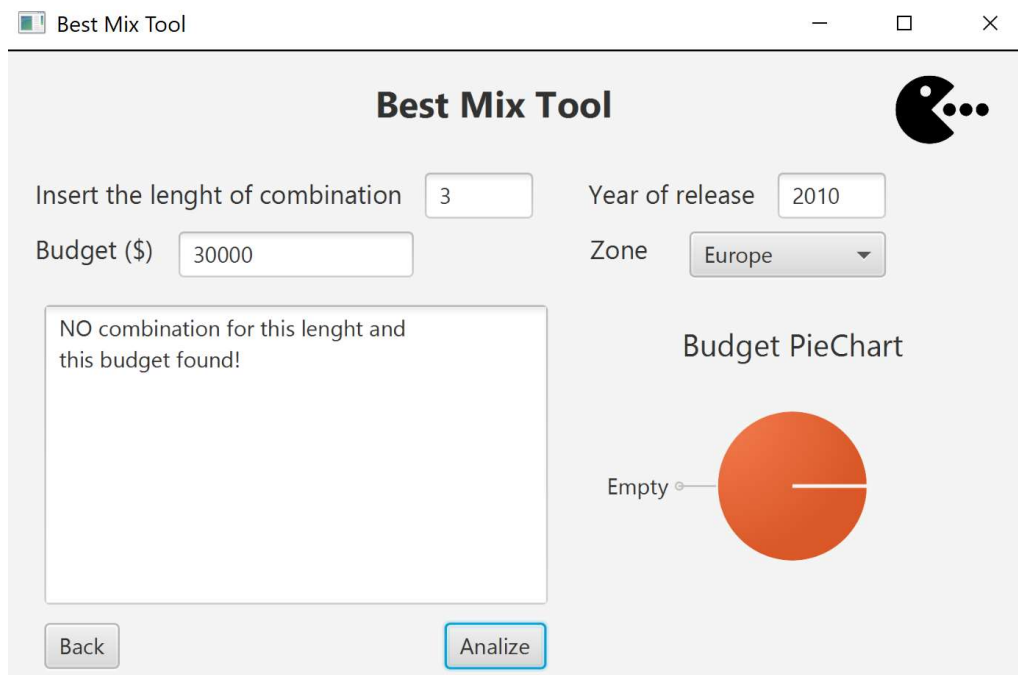
The screenshot shows the 'Best Mix Tool' window after processing. The input fields now contain: 'Insert the lenght of combination' (2), 'Year of release' (2010), 'Budget (\$)' (30000), and 'Zone' (Europe). The large rectangular box on the left now contains the text: 'The best combination for 30000\$ is: 1) Music 2) Action'. The 'Budget PieChart' is now a pie chart with three segments: a large blue segment labeled 'Action', a small orange segment labeled 'Music', and a small white segment labeled 'Budget left'. At the bottom right, it says 'Process time : 20 milliseconds'. The 'Back' and 'Analyze' buttons are still present.

Per una combinazione di **2 elementi** e con un budget di 30000\$ si ottiene un risultato con un tempo di elaborazione davvero piccolo, solo 20 millisecondi.

Andando ad aumentare a 70000\$ il budget (quindi più che raddoppiarlo) otteniamo un nuovo risultato con un tempo di elaborazione praticamente uguale al precedente:

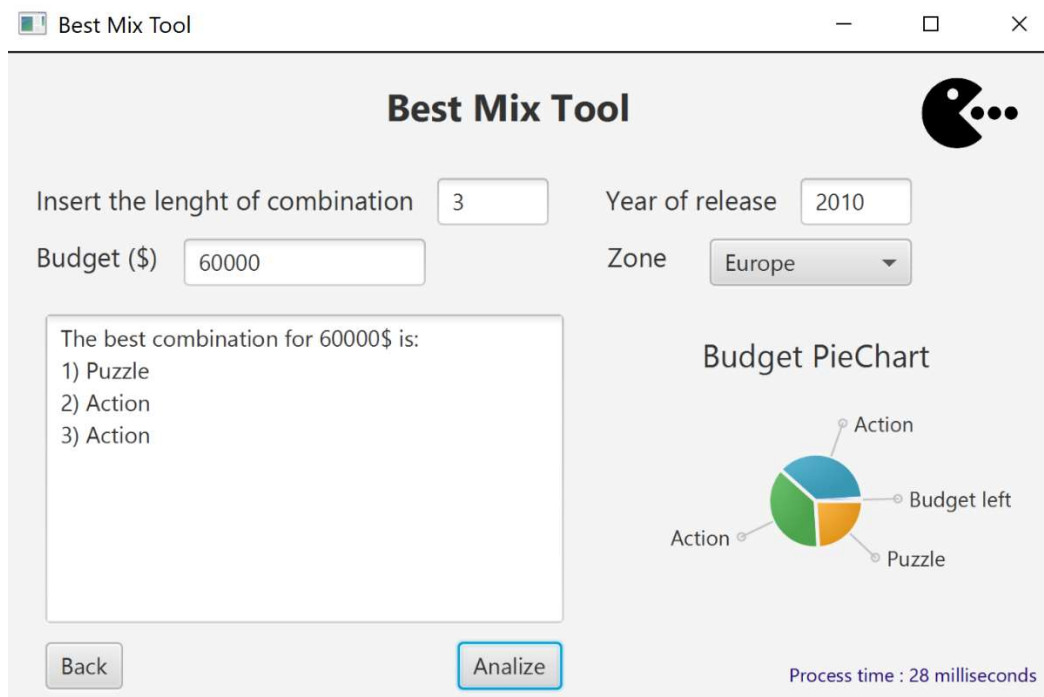


Passando invece ad una combinazione di **3 elementi** e inserendo 30000\$ come budget otteniamo il seguente risultato:

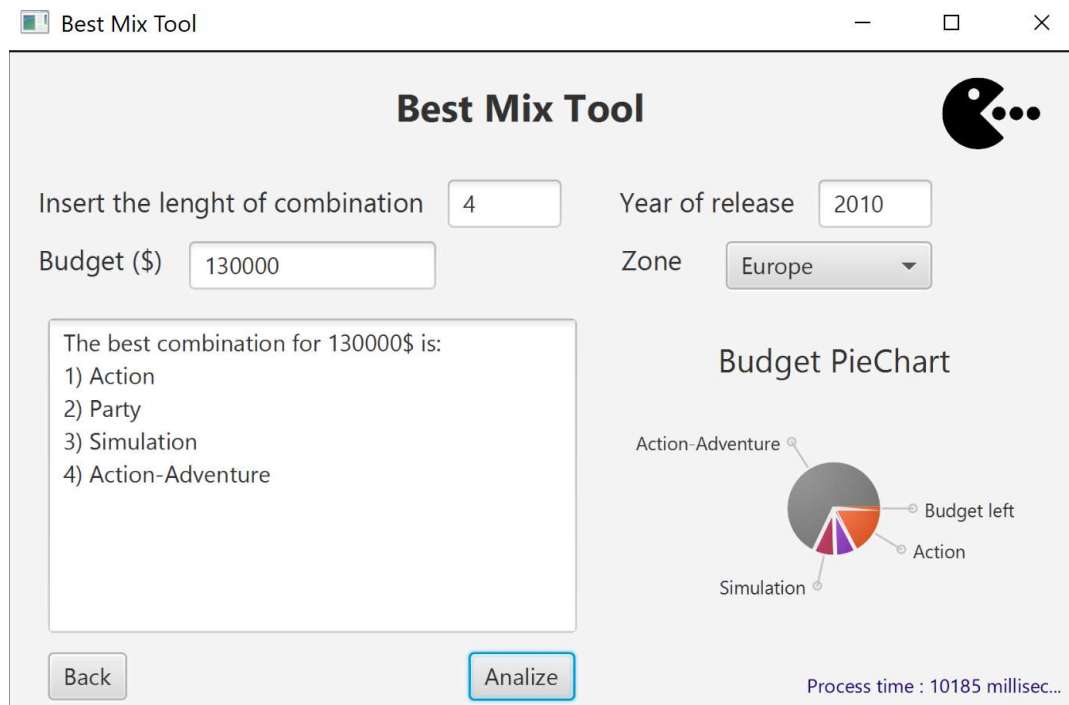


In questo caso l'algoritmo non riesce a trovare una combinazione di lunghezza 3 che riesce ad ottimizzare il budget inserito. Aumentiamo quindi il budget per ottenere un risultato accettabile.

Inserendo, invece, 60000\$ di budget otteniamo questa combinazione:



Infine, andiamo a testare l'applicazione con una combinazione di **4 elementi** e un budget di 130000\$. Otteniamo il seguente risultato:



Come era lecito aspettarsi, in questo caso i tempi di elaborazione salgono a 10185 millisecondi (*circa 10 secondi*) rimando comunque un tempo di elaborazione estremamente accettabile.

❖7 Link al video YouTube

Il video di dimostrazione del funzionamento dell'applicazione è disponibile al seguente link:

<https://youtu.be/miM3XS6t82Y>

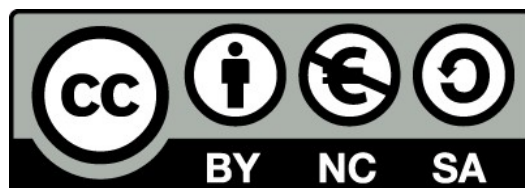
❖8 Valutazione dei risultati e conclusioni

L'obiettivo dell'applicazione è quello di aiutare le software-house videoludiche nella scelta dell'avvio di diversi progetti nelle diverse zone del mondo.

Il programma è rivolto sia per compagini che possiedono un budget alto sia a compagini che possiedono un budget basso. Nel primo caso viene incontro il primo algoritmo, il quale fornisce un semplice aiuto sulla zona del mondo in cui è più favorevole la pubblicazione. In questo caso la consultazione può essere effettuata anche in data successiva all'avvio del progetto.

Nel secondo caso, i piccoli tempi di elaborazione dell'applicazione permettono di ottenere risultati chiari e ottimizzati in breve tempo e utilizzando poche risorse (ovviamente limitate per piccole software-house). Infatti, anche per calcolare una combinazione di quattro elementi vengono spesi soltanto 10 secondi. Per ovvi motivi, questo numero aumenterà all'aumentare della lunghezza della combinazione richiesta, ma per lo scopo dell'applicazione è davvero un ottimo risultato raggiunto.

Infine, bisogna ricordare, che l'applicazione è stata sviluppata come **aiuto al processo decisionale** e non si intende sostituire completamente a quest'ultimo, visto che le variabili in gioco possono essere anche molte altre, oltre ai dati che si trovano sul database che utilizza questa applicazione.



Quest'opera è stata rilasciata con licenza Creative Commons Attribuzione - Non commerciale - Condividi allo stesso modo 4.0 Internazionale.

Copia della licenza consultabile al sito web:

<http://creativecommons.org/licenses/by-nc-sa/4.0/>