Università degli Studi di Napoli "Parthenope"
Facoltà di Scienze e Tecnologie
Corso di Laurea in *Informatica Generale*

Progetto ideato e realizzato per l'esame di Programmazione 3 e Laboratorio di Programmazione 3.

***Prof.*:** Raffaele Montella

***Studente*:** Gabriele Punzo
***Matricola*:** LI/890

***Studente*:** Gaetano Rosario Cirella
***Matricola*:** LI/913

La piattaforma Java

Java è un linguaggio di programmazione orientato agli oggetti, creato da **James Gosling** e altri ingegneri di **Sun Microsystems**. **Java** è un marchio registrato di **Oracle**.
Java soddisfa quattro caratteristiche principali:

1. È orientato agli oggetti;
2. È indipendente dalla piattaforma;
3. Contiene strumenti e librerie per il **networking**;
4. È progettato per eseguire codice da sorgenti remote in modo sicuro.

I programmi scritti in linguaggio Java sono destinati all'esecuzione sulla piattaforma Java, ovvero saranno lanciati su una **Java Virtual Machine** e, a tempo di esecuzione, avranno accesso alle **API** della libreria standard. Ciò fornisce un livello di astrazione che permette alle applicazioni di essere interamente indipendenti dal sistema su cui esse saranno eseguite.
Le specifiche di linguaggio richiedono un ambiente di esecuzione che vigila sull'esecuzione del programma e che proibisce determinate operazioni che altrimenti sarebbero insicure. Esse fanno riferimento esplicito alla **Java Virtual Machine**, indicandola come il destinatario tipico del **bytecode** prodotto dalla compilazione di un programma Java, e infatti il compilatore **`javac`** incluso nel **JDK** compila proprio in bytecode. Tuttavia, è possibile la compilazione verso architetture diverse, e infatti è possibile produrre codice oggetto specifico di un certo sistema operativo, servendosi di un compilatore apposito, ad esempio il **GNU Compiler Collection**.
Il linguaggio in sé definisce solo una minima parte delle librerie utilizzabili in combinazione con il linguaggio stesso. La parte restante è definita dalla piattaforma sulla quale il programma sarà eseguito. In aggiunta, il programmatore può utilizzare un numero arbitrario di librerie di terze parti.

Ambiente di sviluppo

La **Sun** (ora Oracle) mette a disposizione un software **development kit** specifico, chiamato ***Java Development Kit*** (o ***JDK***). Esso include un certo numero di **tool** di uso comune, fra cui **`javac`**, **`javadoc`**, **`jar`**, e altri, atti ad elaborare file sorgenti e/o compilati. Essi lavorano sul codice già scritto e salvato sul sistema: nessuno di essi fornisce un ambiente visivo di scrittura con quelle caratteristiche che tornano utili nella realizzazione di programmi complessi come l'evidenziazione della sintassi tramite colori diversi, l'autocompletamento, o la possibilità di navigare tra i sorgenti tramite il click del mouse. E' per questo motivo che si è scelto di utilizzare un ambiente di lavoro più adeguato che ci facilitasse i compiti di debugging.
L'ambiente di sviluppo utilizzato è **NetBeans IDE 7.1.1**.
NetBeans è un ambiente di sviluppo **multi-linguaggio** scritto interamente in **Java** nato nel giugno 2000. È l'ambiente scelto dalla **Oracle Corporation** come **IDE** ufficiale, da contrapporre al più diffuso **Eclipse**. Possiede numerosi **plug-in** che lo rendono appetibile al pubblico, e richiede **512 Megabyte** di **Ram** a causa dell'uso delle librerie grafiche standard di **Java (Swing)**.

Cos'è THE LAND OF GAMES?

The Land of Games è un **"contenitore di giochi",** con grafica **RPG (**dall'inglese R*ole-Playing Game*). Nel gioco sono implementati dei **Mini-Game**, che servono alla risoluzione della trama principale.

Trama

Il protagonista è un accanito giocatore d'azzardo che resta al verde e pieno di debiti dopo aver dilaniato la sua fortuna al gioco. Rimane bloccato in una ridente cittadina sul mare senza potersi permettere nemmeno una corsa in taxi per tornare a casa e senza la possibilità di lasciare il paese a piedi poiché la polizia gli sta alle calcagna. A questo punto le uniche persone con cui potrà interagire sono chi lavora in tutte le attività del paese. Questi ultimi gli offriranno la possibilità di riscattarsi proprio tramite il gioco d'azzardo che in precedenza l'aveva portato alla rovina. Il nome **The Land Of Games** proviene dal fatto che, appunto, ogni negozio è in realtà una bisca clandestina con tavoli da gioco. Lo scopo del gioco è quello di far guadagnare al protagonista quanti più soldi è possibile, in modo da renderlo libero. Riuscirà a questo punto il nostro giocatore a tornare a casa?!

Dettagli implementativi

Il progetto è stato creato senza l'uso di **API** specifiche ma, soltanto tramite i package **AWT** e **SWING**. Tutti i **pulsanti**, compresa la **JCheckBox** per l'audio, sono selezionabili da **tastiera**. E' stata implementata l'interfaccia **KeyListener** per collegare i pulsanti ai tasti che per **default** saranno selezionati con la **barra spaziatrice**. Per muoversi tra i **componenti** bisogna utilizzare il **TAB**. Dato che le **JDialog** sono impostate in modo da non interagire con lo sfondo sottostante, essendo **"ladre"** di **focus**, prima di digitare i pulsanti bisogna prendere il **focus** con il tasto **TAB**.
I **JFame** sono resi non ridimensionabili e sempre centrati sullo schermo in modo da dare una maggiore robustezza al gioco. I **JFrame** principali sono sviluppati in modo che quando compare uno il precedente si dispone e viceversa, permettendo di utilizzare sempre e solo 1 finestra.
Il **Font** dei pulsanti è stato importato dall'esterno dato che si è scoperto che facendo girare il progetto su macchine diverse il **Font** prestabilito non era visualizzato correttamente. Di seguito è riportata la classe riguardante il **Font:**

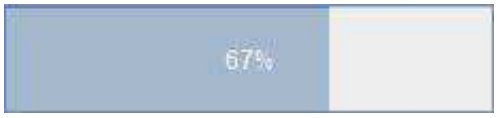
```
//Classe per Font esterni, utilizzati nell'intero progetto.
public class FontClass {

  public Font font1 ;
  public Font font2;

   public FontClass(){
                    "Manca Codice"
    }
}
```

C'è da ricordare che nell'intero progetto non sono stati usati **Layout** per avere più libertà nella disposizione dei componenti, quindi tutto quello che vedete è stato settato **A MANO**, tramite il codice seguente si annulla il **Layout**:

```
panel.setLayout(null);
```

All’inizio del gioco compare un **JFrame** non decorato che utilizza una **JProgressBar** che tramite il metodo **ActionPerformed,** dell’**ActionLiseter,** implementato all’interno di un **Timer** ne permette la progressione. Come si può osservare prima compare una scritta:

e poi la percentuale di caricamento:



Il progetto è diviso in 3 **JFrame** principali:

1. Menu;
2. Schermata Animazioni & Collisioni;
3. Schermata del sette e mezzo.

La **prima parte** è formata da un **JFrame** dove sono stati montati: **4 pulsanti**, il **suono** e la **data/ora,** di seguito è riportata la classe relativa al **Menù**:

```
public class Menu extends JFrame{

    public static StartFrame FS;

    class Evento implements ActionListener, KeyListener{

        @Override
         public void actionPerformed(ActionEvent ae) {

             if (ae.getSource()== musica){
                                        "MANCA CODICE"
                                        }

             else  if(ae.getSource() == bottoni.Start ){
                                                        "MANCA CODICE"
                                                        }

             else if(ae.getSource() == bottoni.Guide){
                                                      "MANCA CODICE"
                                                      }

             else if(ae.getSource() == bottoni.Credits){
                                                        "MANCA CODICE"
                                                        }

             else if (ae.getSource() == bottoni.Quit){
                                                      "MANCA CODICE"
                                                     }
              }
```

```
        @Override
        public void keyTyped(KeyEvent ke) {
        }

        @Override
        public void keyPressed(KeyEvent ke) {
        }

        @Override
        public void keyReleased(KeyEvent ke) {
        }

    }

    Evento E= new Evento();

    JPanel panel = new JPanel();

    MyButton bottoni= new MyButton();

    public static DataEOra deo= new DataEOra();

    public static Audio musica;

    public Menu() throws Throwable{
                                    "MANCA CODICE"
                                    }


   public void bye(){
                     "MANCA CODICE"
                     }
}
```

Si è deciso di inserire la **Data e l'Ora** per dare un tocco di originalità e utilità. Per far visualizzare lo scorrimento dei **minuti** e di conseguenza delle **ore** abbiamo utilizzato un Timer tramite: **import javax.swing.Timer**; inserendolo nel metodo dell'**Action Listener** cioè **Action Performed**. Di seguito è riportata la classe riguardante la **data/ora**:

```
public class DataEOra extends JTextArea {

        private DateFormat data;
        private DateFormat ora;

        FontClass fc= new FontClass();

        public DataEOra() {

                new Timer(1000, new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                        "MANCA CODICE"
                        }
                }).start();
        }
}
```

Il suono è un **MIDI**, settato per essere riprodotto di continuo tramite i metodi di: **import javax.sound.midi.MidiSystem**; Può essere **abilitato/disabilitato** tramite una **JCheckBox**. Sono stati implementati dei metodi per **avviare** e **fermare** l'audio. Di seguito è riportata la classe riguardante l'audio:

```
//Estendiamo JCheckBox.
public class Audio extends JCheckBox {

   //Creiamo un oggetto della nostra classe FontClass.
   FontClass fc= new FontClass();

    //prendiamo il file audio.
   File file = new File("src/Music/DB.mid");

   //Otteniamo il flusso dati tramite Sequence.
   Sequence sequence;

   //Otteniamo un player Sequencer.
   Sequencer player;


   //throws, aiuta un metodo a sollevare un'eccezione in caso di fallimento.
   Audio() throws Throwable {
                              "MANCA CODICE"
                              }

    //Metodo per far partire l'audio.
    public void startAudio(){
                              "MANCA CODICE"
                              }

    //Metodo per fermare l'audio.
    public void stopAudio(){
                             "MANCA CODICE"
                             }
}
```

Il primo pulsante **Start** serve per spostarsi nella seconda parte del gioco cioè nella **Schermata delle Animazioni & Collisioni**.
Il secondo, **Guide**, fornisce le indicazioni e regole del gioco.
Il terzo, **Credits**, dove è possibile visualizzare i crediti del gioco.
Il quarto, **Quit**, ovviamente serve per uscire dal gioco, anche se è possibile uscire anche tramite la crocetta in alto a destra.

Nella **seconda parte** abbiamo il nostro **RPG** cioè il protagonista che tramite **Sprite**, **Collisioni** e **Animazioni** interagisce con le **persone** e **l'ambiente circostante**. Deliberatamente si è pensato di non far gestire **l'audio** e la **data/ora** in questa schermata per non intaccare gli oggetti sullo sfondo. Di seguito abbiamo la classe dove sarà montato l' **RPG** vero e proprio cioè **StartPanel**. I **personaggi,** e le **palme** sono delle **Sprite** montate sul nostro sfondo. Esse sono importate tramite la classe **ImageIcon** e disegnate tramite il metodo **Paint Component** della classe **Graphics**, che è presente in ogni componente **Java**.

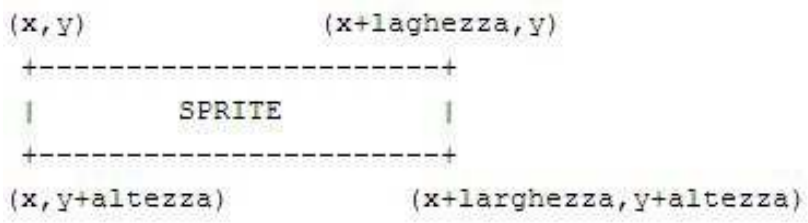
```
public class StartPanel extends JPanel implements ActionListener{

    JLabel L1 = new JLabel(new ImageIcon("src/Image/Sfondo1.jpg"));
    JLabel L2 = new JLabel("Copyright © G&G 2012");

    JTextField saldoCorrente = new JTextField();

    FontClass fc= new FontClass();


    Protagonista personaggio=new Protagonista();

    Usciere usciere= new Usciere();
    Cuoco cuoco= new Cuoco();
    Commessa commessa= new Commessa();
    Taxi taxi = new Taxi();
    Bello bello= new Bello();
    Guardia guardia= new Guardia();
    Vignetta vignetta= new Vignetta();

    Palme palme= new Palme(833,147);
    Palme2 palme2= new Palme2(626,147);
    Palme palme3= new Palme(281,147);
    Palme3 palme4= new Palme3(142,147);


    MyTimer t;
    Timer timer;

    public static boolean stampaCorriera;
    public static boolean stampaPersonaggio;

    public StartPanel() {
                          "MANCA CODICE"
                         }


    @Override
     public void actionPerformed(ActionEvent e) {
                                                   "MANCA CODICE"
                                                  }

     @Override
     public void paintComponent(Graphics g){
                                           "MANCA CODICE"
                                           };
}
```

Nella classe **protagonista** è stato implementato un algoritmo, **animation**, che permette l'animazione del personaggio, rappresentato tramite **sprite**.
L'algoritmo scorre una successione di **Sprite** diverse simulando il movimento. I **tasti** utilizzati sono le **frecce direzionali** della **keyboard**, gestiti attraverso i metodi **KeyPressed** e **KeyReleased** della **import java.awt.event.KeyEvent;**

```
//Classe principale del protagonista con collisioni e animazione.
public class Protagonista extends MySprite{

                //Dichiariamo gli oggetti dei vari personaggi.
                Usciere us = new Usciere();
                Cuoco cu= new Cuoco();
                Commessa co= new Commessa();
                Taxi ta = new Taxi();
                Bello be= new Bello();
                Guardia gu= new Guardia();

                //Variabile per il controllo della poszione del taxi.
                public static boolean stampaRetro;

                //Variabile che gestisce la velocità.
                protected int velocità;

                protected Image[] image;

                //Incrementi sulla direzione x e sulla direzione y.
                protected int dx,dy;

                //Flag per la gestione dei movimenti.
                protected boolean up, down, right, left;
                protected boolean up2, down2;
                //Creiamo un oggetto della nostra classe MyTimer, per dare il tempo all'animazione.
                //Cioè ogni quanto deve cambiare frame.
                MyTimer timer;

                //Variabili per la gestione dell'animzione.
                double tempoAnimazione;
public Protagonista(){
                    //Settiamo i parametri.
                    timer=new MyTimer();
                    //Coordinate iniziali del protagonista.
                    this.setX(945);
                    this.setY(150);

                    "MANCA CODICE"
                }
```

```
//Metodo per il movimento del protagonista.
public void movimento(){
                    "MANCA CODICE"
                }

//metodo per l'animazione dei frame del protagonista.
public void animation(int frame){
                                "MANCA CODICE"
          }

//Metodo  della classe KeyAdapter dell'interfaccia KeyListener
public void keyReleased(KeyEvent e){
                                "MANCA CODICE"
      }

 //Metodo  della classe KeyAdapter dell'interfaccia KeyListener
public void keyPressed(KeyEvent e){
                                "MANCA CODICE"
        }
public void collisionOb(MySprite ob){
                                "MANCA CODICE"
   }


public void collisionBoard(){

}


public MyTimer getMyTimer(){
        return timer;
    }

public void cancMove()
{
    dx=0;
    dy=0;
}

}
```

Ovviamente è importante gestire il comportamento tra le **sprite**. Ad esempio se il protagonista incontra un altro personaggio del gioco, oppure deve interagire con un qualsiasi oggetto del gioco. Per gestire questi casi, si utilizza il metodo del "**rilevamento delle collisioni**".

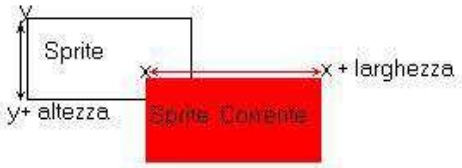
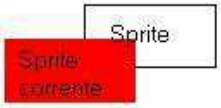
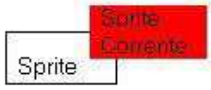
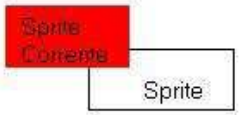


E’ pratica comune per tutti i giochi che utilizzano **sprite** di gestire queste **collisioni** tramite dei metodi all’ interno delle **classi** e associare al rilevamento delle stesse le azioni che ne derivano (effetti grafici oppure comparsa di finestre, etc.). In **The Land of Games**, le collisioni sono state gestite all’ interno della classe **protagonista** che a sua volta deriva dalla classe **MySprite**, come si può vedere di seguito:

```
public class MySprite
{
    protected int x,y;
    protected Image image;
    protected int altezza,larghezza;

  public MySprite(){
                    "MANCA CODICE"
                  }

    public Image getImage(){
                            "MANCA CODICE"
                           }

    public void setImage(Image img){
                                    "MANCA CODICE"
                                    }

    public int getX(){
                      "MANCA CODICE"
                     }

    public void setX(int x){
                            "MANCA CODICE"
                            }

    public int getY(){
                      "MANCA CODICE"
                      }

    public void setY(int y){
                            "MANCA CODICE"
                            }

    public Rectangle getRectangle(){
                                    "MANCA CODICE"
                                    }

    public boolean collisione(MySprite s){
                                          "MANCA CODICE"
                                          }

        }
```

E’ stata una scelta progettuale dovuta al fatto che solo in base alle **collisioni** che il protagonista provoca, nel camminare sullo schermo, si devono verificare determinate azioni all’ interno del riquadro di gioco (es.: è il protagonista, con la collisione con la sprite dell’ **Usciere**, ad avviare il gioco del **Sette e Mezzo**, etc.). Per fare quanto descritto si è deciso di utilizzare la classe **java.awt.Rectangle** che ci fornisce gli strumenti necessari. Essendo ogni **sprite** un’immagine,

quindi di forma **rettangolare**, è possibile porre su di essa un **rettangolo virtuale** così da poter tracciare gli spigoli e i bordi. Tutto ciò è utile poiché si può avvertire la **collisione** tra questi **rettangoli** tramite il metodo **intersect()** della classe.
C'è da dire che ogni sprite dovrà essere in grado di dire se c'è una collisione o meno e dovrà possedere a questo scopo il **rettangolo**. Ecco perché questi due metodi sono stati implementati all'interno della classe **MySprite**. Mentre nella classe **protagonista** sono definiti i metodi che descrivono la sua interazione con le altre sprite. Nell'immagine che segue, si può notare graficamente il lavoro svolto per implementare le collisioni:

	Lo sprite corrente (quello in esame) è entrato con l' angolo superiore sinistro nell' altro sprite di conseguenza c' è stata una collisione.
	Lo Sprite Corrente è entrato in collisione con l'angolo superiore destro: Collisione avvenuta
	Collisione con l' angolo inferiore Sinistro
	Collisione con l'angolo inferiore destro

Nel caso dei **personaggi** la collisione provoca la visualizzazione di una **JDialog**, mentre le collisioni riguardanti i limiti del **campo d'azione**, fanno si che il personaggio non acceda a zone al di fuori della **strada principale**.
Lo sfondo del gioco non offre una visione puramente dall'alto, quindi le **collisioni** e le **sprite** sono state gestite in modo da offrire una prospettiva laterale più simile alla realtà. Il personaggio fuori all'albergo è **l'usciere,** tramite un'avvenuta collisione si potrà accedere alla **terza parte** del progetto, cioè il gioco d'azzardo del **Sette e Mezzo**.

La **terza parte** è in sostanza il gioco del **Sette e Mezzo** con **carte francesi**, variante del classico gioco di carte napoletane, per maggiori informazioni si consulti **Wikipedia**. Alle regole descritte, abbiamo introdotto una variazione, cioè non si possono pescare carte se non si esegue una puntata, in modo da rendere il gioco più avvincente e azzardoso.
In questo **terzo JFrame** abbiamo montato i pulsanti in base ai comandi classici del gioco cioè: **SOLDI**, **PUNTA**, **CARTA** e **STAI.**
I pulsanti sono **abilitati** o **disabilitati automaticamente** in base alle fasi del gioco. Sono state create delle **JLabel**, i quali bordi definiscono la posizione delle carte sul tavolo di gioco. La prima carta del **banco** è **coperta** in modo da nascondere al giocatore il suo valore. Si è cercato di dare un'intelligenza artificiale al **banco** tramite la creazione di un semplice algoritmo, che permettesse al giocatore di vivere un'esperienza di gioco più avvincente!!!
Sono state implementate 3 **JTextArea**, per la visualizzazione dell'andamento del **danaro**, cioè: il passaggio dei soldi dalla **cassa** alla **puntata** e da quest'ultima al **piatto**. In caso di **vittoria** i soldi

dal **piatto** ritornano nella **cassa**, opportunamente incrementata, mentre in caso di **sconfitta** i soldi saranno persi. Anche in questa fase del gioco, in alto a destra, si potrà attivare o disattivare l'audio.
Se **"STIAMO"** con un punteggio minore o uguale di 2 e vinciamo, apparirà una **JDialog** che ci avverte del **bluff** riuscito, dato che **"STARE"** con una mano inferiore o uguale ai 2 punti è **bluffare**!!!
I progressi nel gioco (Budget del giocatore) sono salvati nel caso si voglia prendere una pausa e giocare dopo un po' di tempo. Attenzione però a non chiudere del tutto l' applicazione, altrimenti si ripartirà dai soliti **50 euro** a disposizione del nostro protagonista i quali verranno dati solo all'ingresso dell'albergo e non prima!
E' stata anche implementata una **JTextField** che, nella schermata precedente al **Sette e Mezzo**, ci tiene informati costantemente sui progressi del gioco, cioè memorizza il **saldo corrente**. Poichè noi non sappiamo a priori a quanto ammonterà il nostro saldo quando smetteremo di giocare (saldo a 1, 2, 3, 10 o 100 cifre) quest' ultima è stata resa **dinamica**. Ovvero s'ingrandisce o rimpicciolisce in base al numero di cifre dell'ammontare. Tutto questo è stato possibile, collegandola ad un timer (presente nell' **Action Performed**) che controlla periodicamente il **Budget** del giocatore al quale il campo di testo è collegato. E' da notare la **classe Carta** che si occupa di definire una **carta** come un insieme di proprietà quali il **seme**, il **valore** (relativo al gioco implementato), il **numero** e se è **coperta o meno**. La **classe Carta** è coadiuvata dalla **classe Mazzo**, che forma un mazzo di carte. Poi c'è la classe **Shuffler** che ricopre un ruolo essenziale in tutto il gioco: il **Mazziere** ovvero il **PC**. **Shuffler** si occupa di mischiare il mazzo di carte prima e durante il gioco e di dare le carte (da non dimenticare che il gioco implementato è un **gioco da banco**). Allo stesso tempo deve avere un'intelligenza, perché oltre a compiere i compiti del mazziere deve anche svolgere i compiti di un normale giocatore.

```
public class Shuffler {
//Il mazzo di carte utilizzato dal Shuffler.
public static Mazzo mazzo;
//Le carte sul tavolo.
public static Carta[][] table;
//Il numero di carte prelevate dal Shuffler e dal giocatore.
public static int[] numCarta;

//Il costruttore di default, inizializza il mazzo e lo mischia.
public Shuffler() {
                    "MANCA CODICE"
        }

/*Serve le prime 4 carte.*/
public void start() {
                     "MANCA CODICE"
        }

public void carta(int chi) {
                            "MANCA CODICE"
        }

public void DEmblè() {
                    "MANCA CODICE"
          }


public static Carta[] getCarta(int chi) {
                                        "MANCA CODICE"
                                        return carta;
        }
public int getNumCarte(int chi) {
                                  "MANCA CODICE"
                                  return Shuffler.numCarta[chi];
        }

public float getPuntiMano(int chi) {
                                    "MANCA CODICE"
                                    return player;
        }

public float getPuntiScoperti(int chi){
                                       "MANCA CODICE"
                                       return player;
        }


public void Vincitore() {
                         "MANCA CODICE"
        }


public boolean isSballato() {
                            "MANCA CODICE"
                            return false;
        }


public void intelligence() {
                           "MANCA CODICE"
        }
}
```

La **classe** più importante del gioco è quella relativa al **tavolo**. Essa implementa il **tavolo da gioco**, dove compariranno le carte date dal mazziere e dove l' utente può interagire. L' utente è inteso come utente che gioca, quindi la classe giocatore, relativamente al gioco del sette e mezzo, possiede

un solo attributo inerente: il Budget, cioè la quantità di soldi che è riuscito a guadagnare con il gioco.

```
public final class Tavolo extends JFrame implements ActionListener, KeyListener{

    @Override
    public void actionPerformed(ActionEvent ae) {
        {
         if(ae.getSource()==bottoni.cinque){
                                            "MANCA CODICE"
                                           }

         else if(ae.getSource()==bottoni.dieci){
                                             "MANCA CODICE"
                                            }

         else if(ae.getSource()==bottoni.venti){
                                             "MANCA CODICE"
                                             }

         else if(ae.getSource()==bottoni.cinquanta){
                                               "MANCA CODICE"
                                              }

         else if(ae.getSource()==bottoni.cento){
                                            "MANCA CODICE"
                                             }
         else if(ae.getSource()==bottoni.duecento){
                                              "MANCA CODICE"
                                               }

         else if(ae.getSource()==bottoni.cinquecento){
                                                  "MANCA CODICE"
                                                  }

         else if(ae.getSource()==bottoni.mille){
                                            "MANCA CODICE"
                                             }

         else if(ae.getSource()==bottoni.allIn){
                                             "MANCA CODICE"
                                             }

         else if((ae.getSource()==bottoni.punta)&&(somma!=0)){
                                                          "MANCA CODICE"
                                                          }

        else if(ae.getSource()==bottoni.carta ){
                                             "MANCA CODICE"
                                            }

        else if (ae.getSource()==bottoni.stai ){
                                              "MANCA CODICE"


                                         }
        }
                             "MANCA CODICE"


        }
```

```
    @Override
    public void keyTyped(KeyEvent ke) {
    }

    @Override
    public void keyPressed(KeyEvent ke) {
    }

    @Override
    public void keyReleased(KeyEvent ke) {
    }


         class FrameListener extends WindowAdapter{

            @Override
            public void windowClosing(WindowEvent e){

                try {
                     "MANCA CODICE"
                    } catch (Throwable ex) {
                      Logger.getLogger(StartFrame.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        }

    //Creiamo l'oggetto della inner class che verrà montato sul JFrame
    FrameListener F= new FrameListener();

    MyButton bottoni= new MyButton();

    Carta carte= new Carta();

    FontClass fc= new FontClass();

    public static Shuffler banco=new Shuffler();

    public static Giocatore g= new Giocatore();

    public static int somma=0;
    public static int piatto=0;
   Bordo bordo= new Bordo();

    JPanel panel= new JPanel();
    JLabel l= new JLabel(new ImageIcon("src/Image/Tavolo.jpg"));
    JLabel L2 = new JLabel("Copyright © G&G 2012");

    JTextArea testoCassa=new JTextArea("     CASSA  ");
    JTextArea testoPiattoBox=new JTextArea("   PIATTO ");
    JTextArea testoPuntata= new JTextArea("  PUNTATA ");

    public static JTextArea cassa=new JTextArea();
    public static JTextArea piattoBox=new JTextArea();
    JTextArea puntata= new JTextArea("\n"+"        0€");



    public Tavolo(int salva){
                             "MANCA CODICE"
                            }


   public void drawCard(){
                           "MANCA CODICE"
                           }
   }
}
```

Una volta raggiunta la somma necessaria per pagare i creditori comparirà una **JDialog** che avvertirà il giocatore. Se si seleziona "**SI"** di colpo partirà la conclusione del gioco tramite un animazione gestita con un nuovo **thred**, raffigurante il nostro protagonista che esce dall'hotel e sale sulla **Corriera** che a tutta "birra" fugge dalla città. È buona pratica, per gestire le animazioni, utilizzare un altro **thread**. **In The Land of Game** è stata implementata una nuova classe che implementa l'**interfaccia Runnable**. Questa interfaccia implementa automaticamente un metodo **Run()**, tale metodo dirà al **thread** qual' è il suo compito e cosa dovrà eseguire, fino a che non terminerà. Nel gioco, la classe che implementa **runnable** è la **classe GameOver** in tale classe sono stati implementati anche dei metodi per gestire la pausa, lo start e lo stop del **thread.**

```
public class GameOver implements Runnable {

    public static Corriera corriera;
    public static Fumo fumo = new Fumo();
    public static Fumo fumo1 = new Fumo();
    public static Fumo fumo2 = new Fumo ();
    private Image image1[];
    private Image image2[];
    private Thread t=null;
    public static Protagonista personaggio;

    public GameOver(){
                      "MANCA CODICE"
                     }

    @Override
    public void run(){
                       "MANCA CODICE"
                     }
                       }

               }
    }

    public void pause(int time){
                                "MANCA CODICE"
                                }

    public void start(){
                         "MANCA CODICE"
                        }

    public void stop() {
                         "MANCA CODICE"
                         }
}
```

Finale

Il nostro protagonista dopo aver estinto i debiti maturati fuggendo scopre a sue spese che il paese e l'intero mondo sono invasi dagli **zombie**. Quindi nelle applicazioni future il nostro progetto si trasformerà da **The Land Of Games** a **The Land Of Zombie**. Quest'idea è liberamente ispirata al film **"dal tramonto all'alba"**, nel quale primo e secondo tempo, raccontano due storie consecutive, ma di genere completamente diverso. Solo chi finirà il gioco potrà vedere il tutto. Come anticipazione di quanto ha detto, alla fine del gioco la schermata di **GameOver** raffigura una fine imprevista per il nostro protagonista.

Future Implementazioni

La **versione demo** che può essere scaricata è fruibile a patto che sulla macchina sia installata la **JVM** e prevede solo la prima parte del gioco, quella relativa a **"The Land Of Games"**. La seconda parte è un idea per una futura implementazione che speriamo di poter realizzare, magari, aggiungendo anche la portabilità sui dispositivi mobili provvisti di **Android** o sotto forma di **Applet Java** da mettere in rete.