POLYGON WAR

di Davide Sito 0124000050

L’applicazione sviluppata per l’esame di programmazione 3 consiste di un videogioco interattivo, in due dimensioni, le cui meccaniche sono quelle classiche degli sparatutto a scorrimento 2D.

Lo scopo e’ accumulare punteggi, eliminando navicelle dello stesso colore dell’astronave del giocatore, estendendo il timer, ed evitando di scontrarsi con i nemici.

Il giocatore ha a disposizione n vite, e n secondi di timer. Quando il timer scade o si esauriscono le vite, la partita termina. Se si colpiscono navicelle del colore corretto, il punteggio totale accumulato aumenta, altrimenti diminuisce. Nel primo caso inoltre viene esteso il timer di gioco. Piu’ nemici del colore corretto vengono abbattuti quindi, piu’ si puo’ giocare e piu’ si accumulano punti.

L’applicazione e’ sviluppata in Java.

CONTROLLI

W= sopra

A= sinistra

D= destra

S= sotto

F= spara

CLASSI COINVOLTE

Le classi implementate sono:

Il metodo main e' contenuto nella classe che ha lo stesso nome del gioco: PolygonWar.class

Abbiamo poi la classe Astronave che definisce l'oggetto dell'astronave guidata dal giocatore.

Abbiamo poi una serie di estensioni di alcune classi Swing, utilizzate per rappresentare graficamente l'ambiente di gioco (MiaFinestra, MioPannello,MiaLabel,LabelStat).

C'e' poi un'interfaccia, Spawner che viene implementata, e che serve a rappresentare gli oggetti che si occupano di dare vita e muovere le entita' coinvolte nel gioco.

Terminano la lista, una classe che si occupa di gestire e controllare le collisioni tra le entita' diverse in gioco (player e nemici, o nemici e colpi sparati) e un oggetto che controlla il timer di gioco.

Segue una breve panoramica delle classi introdotte

MIA FINESTRA (CLASS)

Estende la class JFrame dello Swing package.

E' utilizzata per rappresentare la finestra all'interno del quale sono inseriti gli altri elementi grafici che rappresentano il campo di gioco.

Come variabili di classe contiene vari riferimenti ad oggetti della classe Timer, che vengono utilizzati per rappresentare il "clock" di movimento/aggiornamento/azione

delle entita' diverse coinvolte, e un riferimento all'oggetto di classe MioPannello che altro non e' che un JPanel esteso, inserito nella finestra.

All'interno della classe MiaFinestra e' dichiarata tutta una serie di Inner Class che altro non sono che implementazioni di varie interfacce per il "grab " dei segnali, lanciati dai timer o per la pressione di tasti quando la finestra e' selezionata.

MIOPANNELLO(CLASS)

Estende la class JPanel, e altro non e' che un pannello, di dimensione stabilita, che e' inserito nell'oggetto di classe MiaFinestra, e contiene un riferimento all'oggetto di classe MiaLabel al suo interno inserito, e un riferimento all'oggetto di classe LabelStat, sempre inserito all'interno dell'oggetto pannello.

MIALABEL(CLASS)

Estende la class JLabel. E' uno degli elementi fondamentali dell'applicazione, in quanto all'interno del metodo (in overriding) paintComponent, vengono disegnate le entita' in gioco. All'interno della MiaLabel quindi c'e' tutta una serie di riferimenti ai vari oggetti di gioco (spawner vari, astronave e controllo collisioni)

LABELSTAT(CLASS)

Anch'essa estende la classe JLabel, ed e' la seconda label inserita all'interno del pannello principale. Contiene tutta una serie di informazioni relative alle statistiche di gioco (vita e punteggio)

ASTRONAVE (CLASS)

E’ una classe che si occupa di contenere le informazioni sulle coordinate dell’astronave del giocatore, sul colore attuale, un riferimento al timer che fa scattare ogni n secondi il colore, e i metodi che descrivono lo spostamento nelle varie direzioni dell’astronave. Le coordinate dell’astronave verranno poi disegnate dal metodo paintComponent della label.

CLASSI SPAWNER

Gli oggetti delle varie classi xSpawner sono implementazioni dell’interfaccia Spawner. Si occupano di tenere traccia in strutture dati apposite, delle posizioni dei nemici/stelle/etc generati, e contiene i metodi necessari per aggiornarne le posizioni/generarne di nuovi. Questi metodi sono lanciati dai listener dei timer nella finestra principale.

COLLISION CONTROLLER (CLASS)

L’oggetto di questa classe contiene i riferimenti agli spawner, e ogni volta che vengono lanciati i suoi metodi di controllo, questo scorre tutti gli oggetti gia’ generati, e controlla che non siano avvenute collisioni tra nemici/astronave nemici/colpi, ed in tal caso si occupa di avvisare le entita’ coinvolte, in modo tale che la situazione venga subito gestita.

GAMING TIMER (CLASS)

Questa classe contiene un contatore che rappresenta i secondi rimasti prima della fine della partita, e un oggetto di classe Timer sul quale e’ registrato un listener che ogni tot tempo decrementa il contatore. La classe ha inoltre i metodi necessari per

aggiornare il timer incrementandolo (quando vengono colpiti nemici di colore esatto).

FUNZIONAMENTO GENERALE

- CREAZIONE E CARATTERISTICHE DEGLI OGGETTI

Il programma parte dal main,che e' statico.
Viene allocato un oggetto della classe MiaFinestra. Parte quindi il costruttore di quest'ultima, che definisce dimensioni e proprieta' grafiche. Viene istanziato il pannello che verra' inserito nella finestra, e vengono istanziati i vari oggetti della classe Timer.

Sul timer1 vengono registrati 2 listener, uno per la simulazione dell'attrito (l'astronave che si sposta lentamente all'indietro) e uno per il movimento delle "stelle" (il pulviscolo sullo sfondo). Si fa partire il timer1. Sul timer2 viene aggiunto il listener per la generazione di nemici. E viene fatto partire. Sempre sul timer1 si aggiunge il listener per l'aggiornamento posizione nemici, e quello per l'aggiornamento dei colpi sparati (quindi gia' generati).
Viene allocato un oggetto di classe GamingTimer, e viene attivato.
Viene inoltre creato un altro oggetto di classe Timer (che pero' verra' attivato solo quando la partita sara' terminata, e su questo viene registrato un listener che fa chiudere la finestra quando dopo n secondi dall'attivazione del timer,questo scatta).
Viene poi registrato sulla finestra un listener che scatta quando vengono premuti tasti e la finestra e' selezionata.

Tornando al pannello (oggetto di classe MioPannello) che abbiamo istanziato e inserito nella finestra, parte quindi il costruttore MioPannello(), viene creata la label di classe MiaLabel e viene inserita nel pannello, e viene creato l'oggetto di classe LabelStat, seconda label che viene sempre inserita nel pannello. In fase di creazione delle 2 label, sono passati i riferimenti incrociati affinche' si possa meglio tenere traccia all'interno delle label, degli altri componenti grafici.

Guardando all'istanzazione dell'oggetto di classe MiaLabel, parte il costruttore, e vengono istanziati gli oggetti di classe MiaFinestra (per contenere il riferimento alla finestra che contiene la label) , StarSpawn,EnemySpawner,Astronave,BulletSpawner e CollisionController.
Inoltre all'interno dell'oggetto MiaLabel e' presente un flag (inizializzato a false) chiamato printGameOverStat che verra' utilizzato come indicato in seguito.
Centrale nel funzionamento dell'applicazione e' il metodo paintComponent contenuto nella classe MiaLabel che "sovrascrive" il metodo ereditato dalla classe madre JLabel.
Questo parte alla creazione della MiaLabel, e ogni volta che viene chiamato il metodo repaint() dell'oggetto MiaLabel. Quando il metodo paintComponent() parte, innanzitutto controlla il valore del flag printGameOverStat: se e' false non fa nulla, altrimenti stampa una schermata che indica il punteggio raggiunto, con la scritta GAMEOVER, e attiva il timer4 della MiaFinestra (timer che quando scattera', fara' partire il listener che vi e' registrato sopra, che fa il dispose() della finestra, ovvero la chiude, e quindi chiudendosi la finestra termina il processo).
Se quindi il flag e' false, allora si scorre innanzitutto nell'array che contiene le coordinate x (e su quello delle y) dei punti che rappresentano il pulviscolo/stelle sullo sfondo, e si stampano quelle presenti nell'array contenuto nell'oggetto di classe StarSpawner. Per come la StarSpawner gestisce l'array contenente coordinate, appena, scorrendolo, si incontra un elemento non inizializzato, vuol dire che dopo non c'e' piu' nulla e quindi si puo' smettere di scorrere. Quindi in questo modo abbiamo stampato i puntini che rappresentano stelle/pulviscolo dello sfondo in movimento, nella loro posizione attuale.
Ora e' il momento di disegnare le navicelle nemiche. Si scorre sulle strutture dati che, all'interno dell'oggetto di classe EnemySpawner, contengono le coordinate delle navicelle nemiche. Ogni navicella ha un flag che indica se e' stato colpito o no. Se risultava precedentemente colpito, non scompare dalla struttura dati, ma il metodo paintComponent non lo disegna. Il procedimento e' il seguente: il paintComponent scorre sulla struttura dati contenente coordinate nemici. Se incontra elemento non inizializzato, allora per quanto detto sopra, non ci sono piu' nemici da disegnare, e quindi fa break ed esce dal ciclo. Altrimenti se trova un elemento inizializzato, nel caso in cui il valore del flag sia ==0, allora il nemico non e' stato colpito, quindi lo disegna normalmente saltando avanti nel ciclo. Altrimenti il valore del flag indichera' la fase attuale in cui il nemico COLPITO si trova nel ciclo dell'esplosione, quindi disegna (in base al valore del flag che essendo >0 indica il

cammino nell'esplosione) una forma gialla/rossa/arancione che rappresenta una vampata di fuoco. Nel caso in cui il flag sia uscito fuori range massimo, vuol dire che non solo e' stato colpito ma ha anche terminato la fase di esplosione, quindi semplicemente non viene disegnato nulla.
Ora il metodo paintComponent() stampa i proiettili generati (sempre secondo lo stesso procedimento utilizzato per il pulviscolo)
Ora per il disegno dell'astronave controllata dal giocatore: l'astronave viene disegnata sia se e' iniziata la fase dell'esplosione (perche' il giocatore ha esaurito tutte le vite disponibili) sia se questa non e' iniziata.
Nel caso in cui non sia iniziata, viene disegnata solo l'astronave (utilizzando le coordinate contenute nell'oggetto di classe Astronave chiamato pg e controllando il valore dello stato dei propulsori che quindi andranno disegnati/non disegnati). Se l'esplosione e' iniziata, si fa come sopra, tuttavia seguendo un procedimento simile a quello utilizzato per il "cammino dell'esplosione" per le navicelle nemiche, viene disegnata una diversa forma (rappresentante il fuoco) al di sopra dell'astronave (per dare l'effetto dell'astronave coperta di fiamme), e si incrementa il flag che rappresenta la fase dell'esplosione. Se incrementando la fase dell'esplosione, il frame esce fuori un determinato range, vuol dire che anche l'esplosione e' terminata, quindi facciamo sparire l'astronave e incrementiamo ancora il flag. Quando il flag superera' anche il secondo limite, viene attivato il flag printGameOverStat, e quindi si arrivera' nel caso descritto precedentemente.

Tornando a guardare alla seconda label inserita nel pannello (cioe' l'oggetto di classe LabelStat), quando questo e' inizializzata, parte il costruttore che inizializza le vite a 3 e il punteggio a 0.Anche questa fa override del metodo paintComponent della classe madre JLabel, per disegnare nella parte bassa del pannello (in cui e' inserita) il tempo rimanente (ottenuto lanciando il metodo getTime() dell'oggetto GamingTimer) e le vite rimaste.

Gli spawner che abbiamo inizializzato sopra, conterranno strutture dati apposite per contenere i riferimenti agli oggetti istanziati, con relativi flag e coordinate. In genere, tranne nel caso della classe StarSpawner, gli altri spawner hanno una inner class che definisce gli oggetti associati (es classe Bullet che e' inner class della classe BulletSpawner e classe Enemy che e' inner class della classe EnemySpawner) e che viene utilizzata per istanziare oggetti, ognuno con le proprie coordinate/flag, di cui si terra' traccia in strutture dati generali nello Spawner.

- COMPORTAMENTO DINAMICO

Il comportamento dinamico (movimento, reazione, azione) dell'applicazione e' implementato attraverso un uso massiccio di oggetti di classi listener da noi definiti, che implementano relative interfacce a seconda del segnale che vogliono "intercettare".

Timer timer1: su questo sono registrati oggetti di classi ListenerTimerStelle, ListenerPerAttrito ,ListenerTimerAggiornaEnemy, ListenerTimerAggiornaBullet.

Scatta ogni 100 ms, lanciando un segnale, che e' intercettato dagli oggetti delle varie classi Listener registrati su di esso. Quindi ogni 100 ms parte il metodo actionPerformed() del ListenerTimerStelle, che si occupa di lanciare i metodi update() e spawn() dell'oggetto di classe StarSpawn. Questo fara' muovere le stelle e ne fara' generare una nuova. Inoltre lancia immediatamente il metodo per il controllo delle eventuali collisioni tra colpi sparati e nemici, dell'oggetto di classe CollisionController
Sempre per il timer1, parte anche il metodo actionPerformed() dell'oggetto di classe ListenerPerAttrito: questo controlla che l'astronave non si trovi troppo vicina al bordo inferiore, e se cosi' non e' richiama il metodo attrito() dell'oggetto di classe Astronave. Questo metodo sposta verso il basso le coordinate y dell'oggetto pg di classe Astronave, e aggiorna anche la posizione dei propulsori.
Sempre per il timer1 parte l'actionPerformed() dell'oggetto di classe ListenerTimerAggiornaEnemy, che si occupa di richiamare il controllo dell'oggetto classe CollisionController per valuta che non ci siano collisioni tra nemici e colpi, poi chiama il metodo update() dell'oggetto EnemySpawner, richiama il controllo collisione tra player e astronavi nemiche (sempre dall'oggetto di classe CollisionController) e lancia infine il repaint() della Label.
Secondo lo stesso principio funziona anche per il ListenerTimerAggiornaBullet.

Timer timer2: vi e' registrato oggetto di classe ListenerTimerSpawnEnemy

Il funzionamento di questo timer e' simile a quello del timer1, con la differenza che questo timer ha un clock molto piu' ampio, perche' i nemici vanno generati piu' lentamente del pulviscolo di sfondo.

Timer timer4: vi e'registrato un Listener che fa il dispose() della finestra. Questo timer dura molto piu' degli altri, e non e' attivato se non quando nella Label e' stata stampata la schermata di GAME OVER.

L'oggetto di classe ListenerKeyUp, che implementa l'interfaccia KeyListener, definisce i metodi che partiranno quando verra' premuto un tasto (o due assieme, visto che sono permessi anche gli spostamenti in diagonale, e gli spari menrte ci si sposta) o quando verra' rilasciato.

Quando il CollisionController si accorge che abbiamo colpito un nemico di un colore corretto, dice alla LabelStat di incrementare il punteggio, altrimenti di decrementarlo, e analogamente gestisce il tempo rimanente.
Ogni volta che l'oggetto di classe CollisionController si accorge che un nemico viene colpito, gestisce il flag dell'oggetto Enemy che avvia l'animazione dell'esplosione ed il flag che indica che il proiettile e' esploso e non va disegnato. Allo stesso modo quando il giocatore si scontra con un nemico, fa quanto sopra ed inoltre avvisa l'oggetto di classe LabelStat chiamandone il metodo colpito(). Questa, se le vite non sono terminate, si limita a decrementarle. Altrimenti se le vite sono terminate, avvia l'animazione relativa all'esplosione dell'astronave del giocatore, che quando sara' terminata portera' alla rappresentazione della schermata di GAME OVER, con relativa attivazione del timer4 per il dispose della finestra.

CONDIZIONE TERMINAZIONE PARTITA:

- SE PERDIAMO TUTTE LE VITE:

CollisionController (se giocatore colpito)->lancia colpito() della LabelStat..->La LabelStat se si accorge che le vite sono terminate->imposta il flag animazione esplosione sull'oggetto Astronave->ad ogni nuovo paintComponent si prosegue con l'esplosione->terminata la quale e' impostato il flag printGameOverStat->questo disegna screen di game over e attiva timer4->dopo n secondi parte il segnale dal timer4, che fa partire il metodo dell'oggetto classe Listener registratovi, che fa il dispose() della finestra ,che si chiude, e quindi il processo termina.

- SE SCADE IL TEMPO

  Il tempo di gioco rimanente e 'gestito dall'oggetto di classe GamingTimer. Ogni volta che e' aggiornato, questo lancia il metodo repaint() dell'oggetto di classe LabelStat. Questa quindi dovra' all'interno del metodo paintComponent() ridisegnare la label col nuovo tempo, ma prima di fare questo controlla che non sia scaduto: in tal caso imposta il flag printGameOverStat nell'altra label a true, quindi al prossimo paintComponent dell'oggetto MiaLabel, verra' disegnata la schermata di GameOver, verra' attivato quindi il timer4 e dopo n secondi parita' segnale, che fara' partire il metodo actionPerformed() del listener registrato sul timer4, che lancera' il dispose() della finestra.

FRAMMENTI DI CODICE

CLASSE PolygonWar contenente il metodo main()

```
PUBLIC CLASS POLYGONWAR{

   PUBLIC STATIC VOID MAIN(STRING[] ARGS){

       MIAFINESTRA FIN1=NEW MIAFINESTRA();


   }

}
```

CLASSE MiaFinestra

```
PUBLIC CLASS MIAFINESTRA EXTENDS JFRAME{

  GAMINGTIMER TIMERGAME1;
  TIMER TIMER1;

   TIMER TIMER2; //OGGETTO TIMER PER SPAWN NUOVI NEMICI

   TIMER TIMER4;



   PUBLIC MIAFINESTRA(){ //COSTRUTTORE

        …//INIZIALIZZAZIONE TRA LE VARIE COSE, DEI TIMER E REGISTRAZIONE OGGETTI
      CLASSI CHE IMPLEMENTANO INTERFACCE  LISTENER

   }
   //SERIE DI INNER CLASS CHE IMPLEMENTANO LE INTERFACCE DI LISTENER PER I SEGNALI
   MANDATI DAI TIMER O PER LA PRESSIONE TASTI
   …//AD ESEMPIO:


   PUBLIC CLASS LISTENERTIMERSPAWNENEMY IMPLEMENTS ACTIONLISTENER{

      @OVERRIDE //OVERRIDE DEL MEDOTO ASTRATTO DELL'INTERFACCIA ACTIONLISTENER

      PUBLIC VOID ACTIONPERFORMED(ACTIONEVENT AE) {

         PANE1.LABEL1.SPAWNERNEMICI.SPAWN();



      }
```

INTERFACCIA Spawner

```
PUBLIC ABSTRACT INTERFACE SPAWNER{
   ABSTRACT VOID SPAWN();
   ABSTRACT VOID UPDATE();



}
```

ESEMPIO DI IMPLEMENTAZIONE PER LA CLASSE

```
PUBLIC CLASS ENEMYSPAWNER IMPLEMENTS SPAWNER{
…//DICHIARAZIONE STRUTTURE DATI PER  TENERE TRACCIA OGGETTI NEMICI

PUBLIC VOID UPDATE(){
      FOR(INT I=0;I<ARRAYENEMY.LENGTH;I++)
         IF(ARRAYENEMY[I]!=NULL){
            ARRAYENEMY[I].YCORD+=7;



            IF(ARRAYENEMY[I].OSCILLAZIONE==0)
               ARRAYENEMY[I].XCORD-=5; //SX
            ELSE IF(ARRAYENEMY[I].OSCILLAZIONE==2)
               ARRAYENEMY[I].XCORD+=5; //DX
            //ALTRIMENTI RIMANE AL CENTRO


         }


   }
```

```
PUBLIC VOID SPAWN(){
    INT ILAST;
    FOR(ILAST=0;ILAST<ARRAYENEMY.LENGTH;ILAST++)
        IF(ARRAYENEMY[ILAST]==NULL)
            BREAK;
    IF(ILAST>=ARRAYENEMY.LENGTH-1)
        ILAST=ARRAYENEMY.LENGTH-2;

    FOR(INT I=ILAST;I>=0;I--)
        ARRAYENEMY[I+1]=ARRAYENEMY[I];

    ARRAYENEMY[0]=NEW ENEMY();

}
…DICHIARAZIONE INNER CLASS NEMICI…
```

METODO DI CONTROLLO COLLISIONE ASTRONAVE PLAYER E ASTRONAVE NEMICA NELL'OGGETTO DI CLASSE COLLISION CONTROLLER

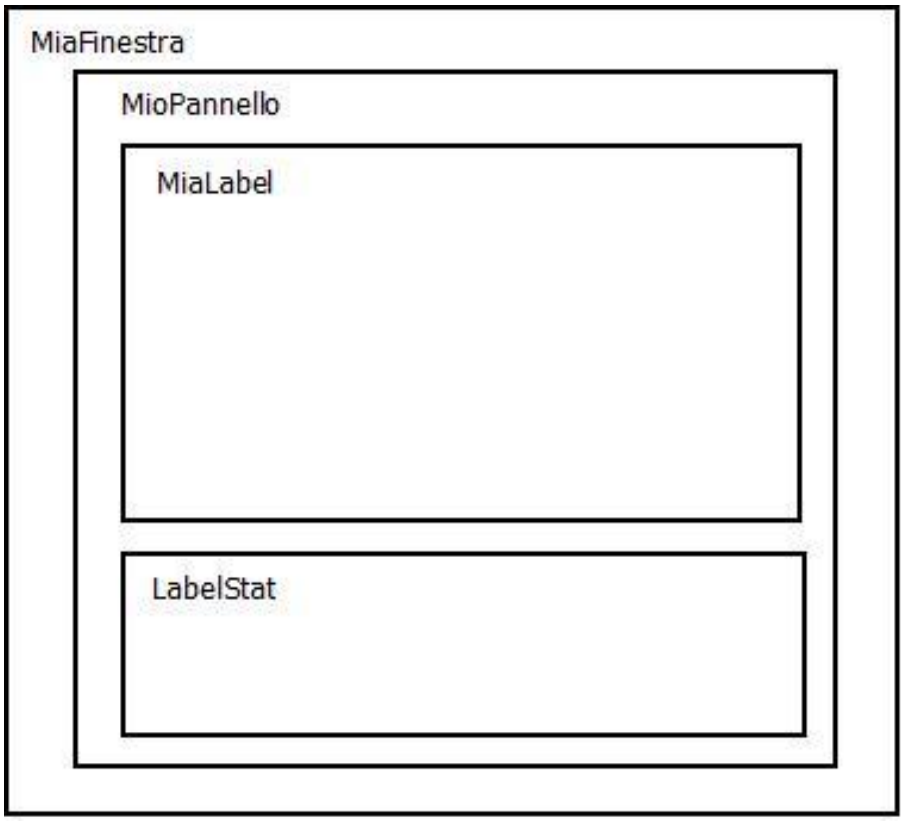
```
PUBLIC VOID ENEMYVSPG(){

    FOR(INT I=0;I<SPAWNERENEMY.ARRAYENEMY.LENGTH;I++){

      IF(SPAWNERENEMY.ARRAYENEMY[I]==NULL)

        BREAK;

      IF(SPAWNERENEMY.ARRAYENEMY[I].COLPITO==TRUE) //CIOE' IL NEMICO RISULTA GIA' COLPITO ED E' INVISIBILE NELLA LABEL

        CONTINUE;

      IF((PG.XCORD[1]>=SPAWNERENEMY.ARRAYENEMY[I].XCORD &&
PG.XCORD[1]<=SPAWNERENEMY.ARRAYENEMY[I].XCORD+55

          && PG.YCORD[1]>=SPAWNERENEMY.ARRAYENEMY[I].YCORD &&
PG.YCORD[1]<=SPAWNERENEMY.ARRAYENEMY[I].YCORD+10)

        || (PG.XCORD[2]>=SPAWNERENEMY.ARRAYENEMY[I].XCORD &&
PG.XCORD[2]<=SPAWNERENEMY.ARRAYENEMY[I].XCORD+55

          && PG.YCORD[2]>=SPAWNERENEMY.ARRAYENEMY[I].YCORD &&
PG.YCORD[2]<=SPAWNERENEMY.ARRAYENEMY[I].YCORD+10)

          || (PG.XCORD[0]>=SPAWNERENEMY.ARRAYENEMY[I].XCORD &&
PG.XCORD[0]<=SPAWNERENEMY.ARRAYENEMY[I].XCORD+55

            && PG.YCORD[0]>=SPAWNERENEMY.ARRAYENEMY[I].YCORD &&
PG.YCORD[0]<=SPAWNERENEMY.ARRAYENEMY[I].YCORD+10)){ //ALLORA ABBIAMO COLPITO UN NEMICO CON LA PUNTA NOSTRA ASTRONAVE E QUINDI:


        SPAWNERENEMY.ARRAYENEMY[I].ANIMAZIONEESPLOSIONE=1;

        SPAWNERENEMY.ARRAYENEMY[I].COLPITO=TRUE;

        //VENIAMO COLPITI

        LABEL2.COLPITO(); //CI LIMITIAMO A DIRLO ALL'OGGETTO DI CLASSE LABELSTAT

      }

    }


  }
```

METODO CONTROLLO COLLISIONE PROIETTILE E ASTRONAVE NEMICA SEMPRE NELL'OGGETTO DI CLASSE COLLISIONCONTROLLER

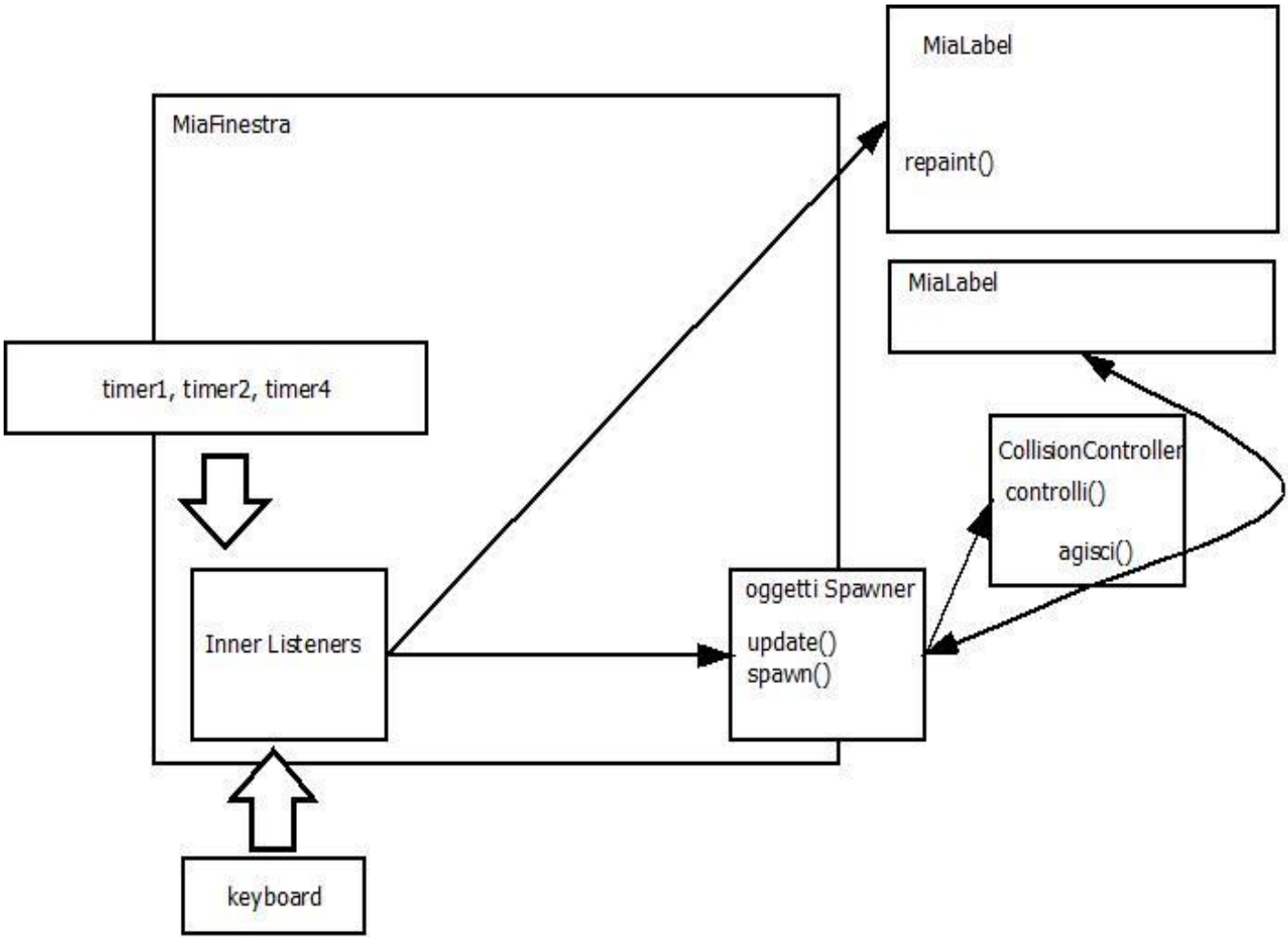
```
PUBLIC VOID ENEMYVSBULLET(){

    FOR(INT I=0;I<SPAWNERENEMY.ARRAYENEMY.LENGTH;I++){

      IF(SPAWNERENEMY.ARRAYENEMY[I]==NULL)

        BREAK;

      IF(SPAWNERENEMY.ARRAYENEMY[I].COLPITO==TRUE)

        CONTINUE; //IL NEMICO RISULTA GIA' COLPITO QUINDI E' COME SE NON ESISTESSE

      FOR(INT J=0;J<SPAWNERBULLET.ARRAYBULLET.LENGTH;J++){

        IF(SPAWNERBULLET.ARRAYBULLET[J]==NULL) //NON CI SONO PIU' COLPI SPAWNATI

          BREAK;

        IF(SPAWNERBULLET.ARRAYBULLET[J].COLPITO==TRUE) //ALLORA IL PROIETTILE E' GIA' ESPLOSO CONTRO QUALCUN ALTRO, QUINDI E' COME SE NON ESISTESSE

          CONTINUE;

        IF(SPAWNERBULLET.ARRAYBULLET[J].X>=SPAWNERENEMY.ARRAYENEMY[I].XCORD-5 && SPAWNERBULLET.ARRAYBULLET[J].X<=SPAWNERENEMY.ARRAYENEMY[I].XCORD+60

            &&
SPAWNERBULLET.ARRAYBULLET[J].Y>=SPAWNERENEMY.ARRAYENEMY[I].YCORD-5 &&
SPAWNERBULLET.ARRAYBULLET[J].Y<=SPAWNERENEMY.ARRAYENEMY[I].YCORD+15){

          //ALLORA NEMICO COLPITO E ANCHE COLPO ESPLOSO (QUINDI COLPITO)

          SPAWNERENEMY.ARRAYENEMY[I].COLPITO=TRUE;

          SPAWNERENEMY.ARRAYENEMY[I].ANIMAZIONEESPLOSIONE=1; //VA ANIMATA L'ESPLOSIONE

          SPAWNERBULLET.ARRAYBULLET[J].COLPITO=TRUE;

          //SE IL COLORE DEL PROIETTILE COMBACIA, ALLORA INCREMENTA PUNTEGGIO E INCREMENTA TIMERGAME


IF(SPAWNERBULLET.ARRAYBULLET[J].COLORE==SPAWNERENEMY.ARRAYENEMY[I].COLORE){

            LABEL2.SCORING(+100);

            LABEL2.PANE1.FRAMECONTENITORE.TIMERGAME1.INCREMENTA();
```

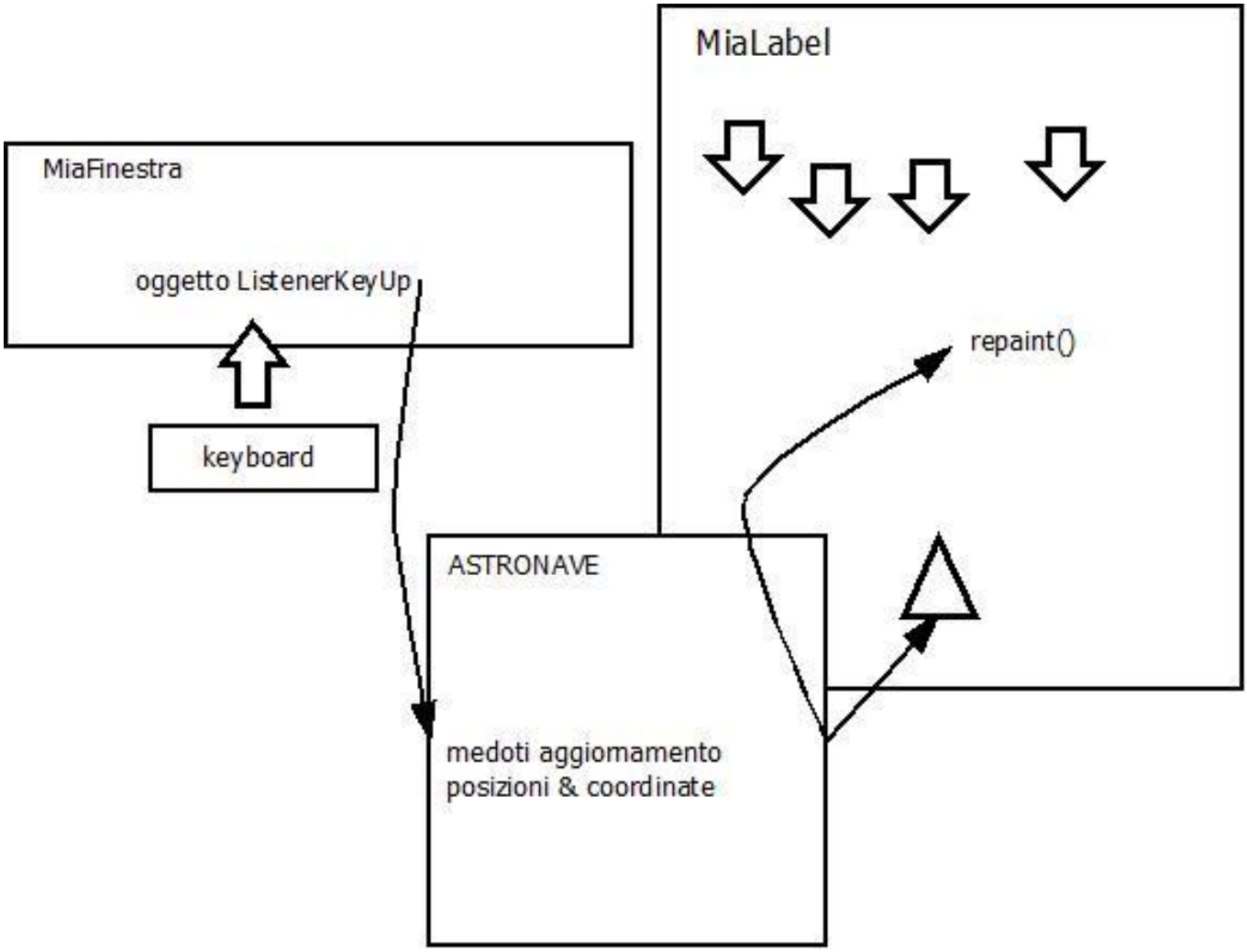
```
            }

            ELSE //ALTRIMENTI DECREMENTALO

               LABEL2.SCORING(-50);

            //E LANCIAMO IL REPAINT DELLA LABEL2 PER RIDISEGNARE LA STRINGA DEL
PUNTEGGIO AGGIORNATA

            LABEL2.REPAINT();

          }

        }

     }


  }
```

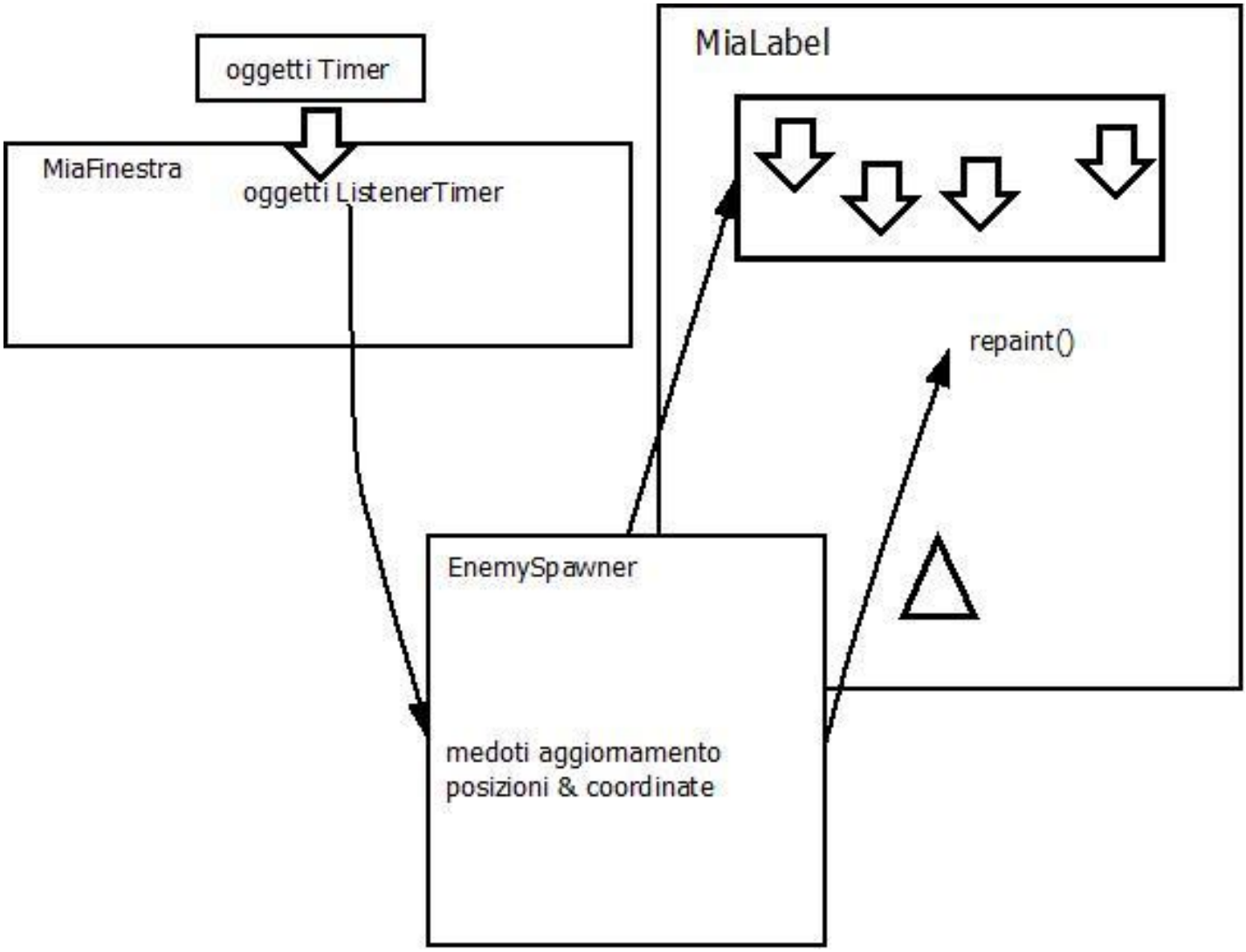
GERARCHIA COMPONENTI GRAFICI

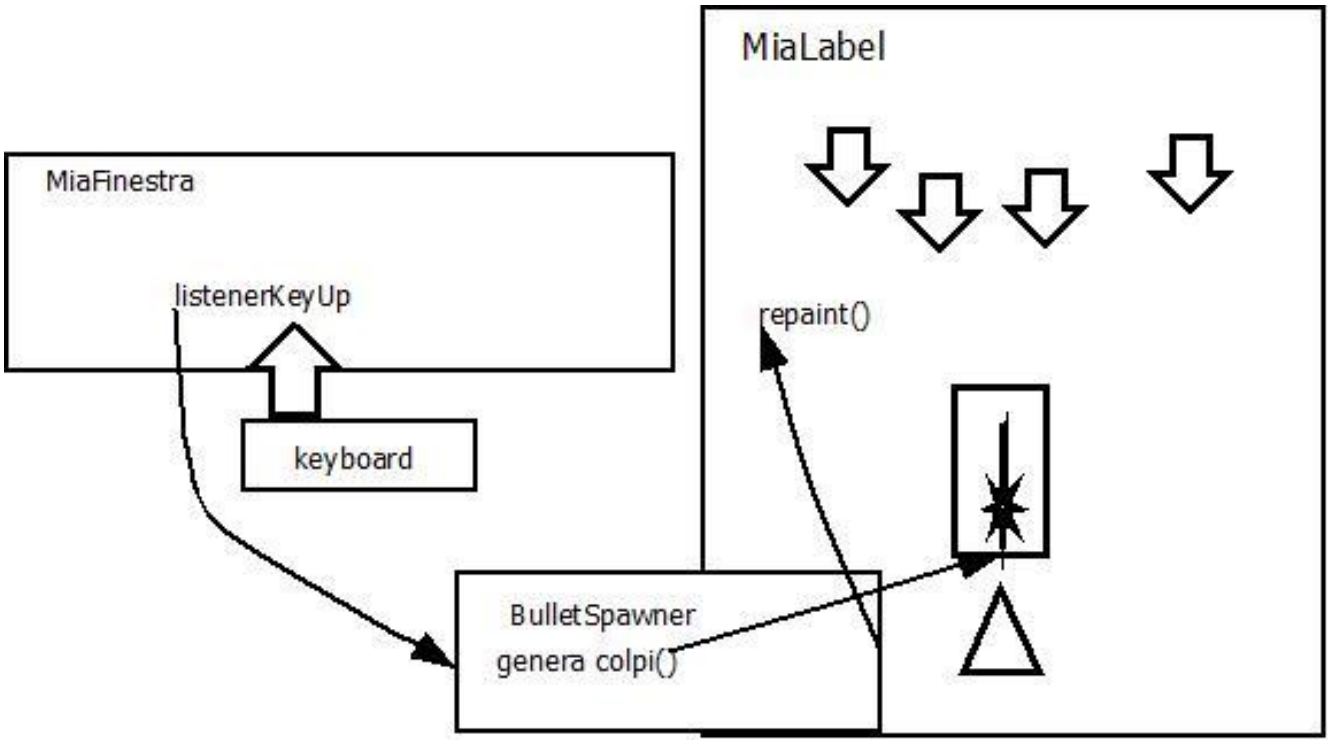


ESEMPIO DI INTERAZIONE DINAMICA TRA GLI OGGETTI

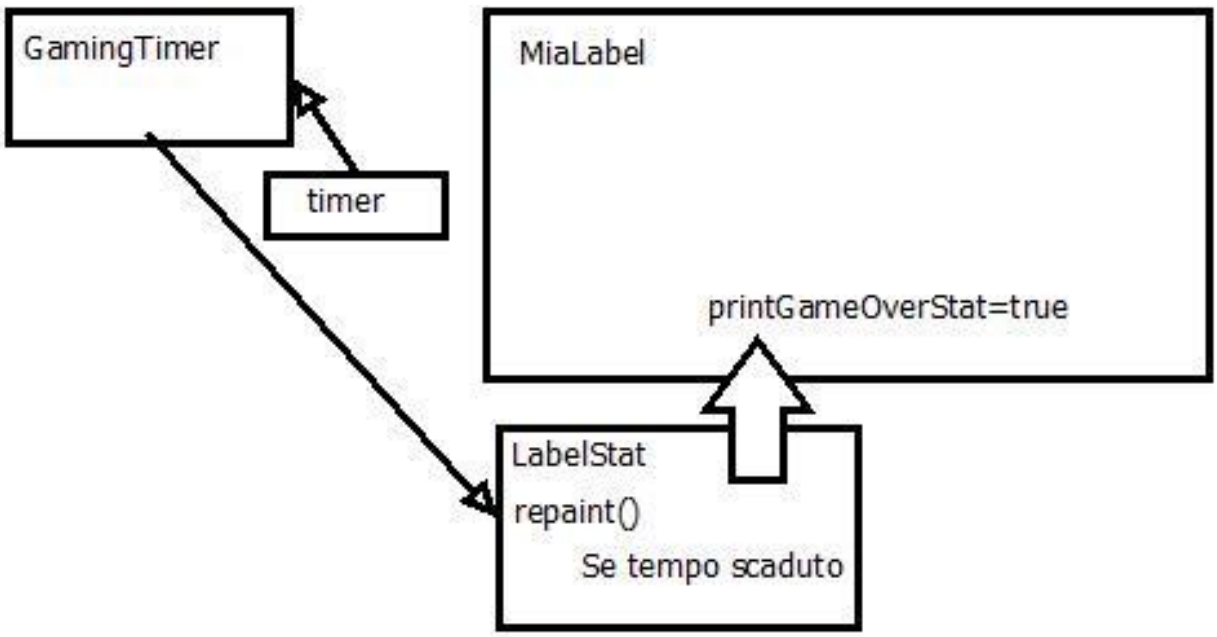


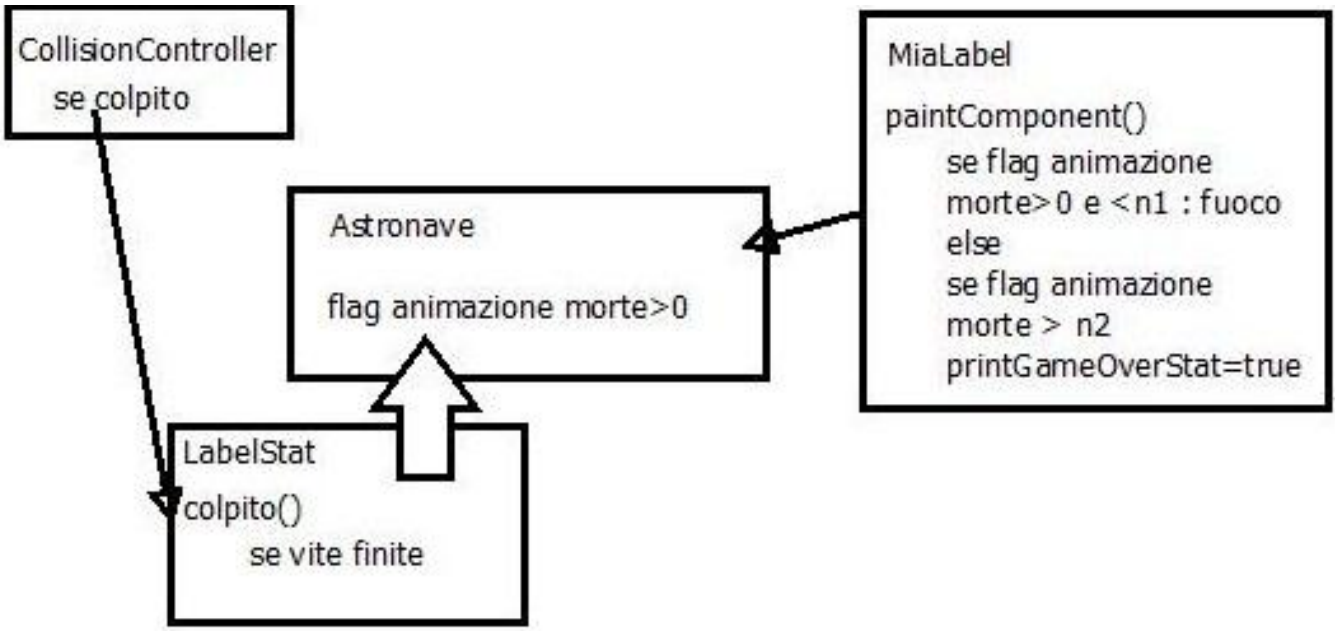

MiaLabel
MiaFinestra
oggetto ListenerKeyUp
keyboard
repaint()
ASTRONAVE
medoti aggiornamento
posizioni & coordinate


oggetti Timer
MiaFinestra
oggetti ListenerTimer
MiaLabel
repaint()
EnemySpawner
medoti aggiornamento
posizioni & coordinate


MiaLabel
MiaFinestra
listenerKeyUp
keyboard
repaint()
BulletSpawner
genera colpi()

CONDIZIONI TERMINAZIONE




CollisionController
se colpito
MiaLabel
paintComponent()
se flag animazione
morte>0 e <n1 : fuoco
else
se flag animazione
morte > n2
printGameOverStat=true
Astronave
flag animazione morte>0
LabelStat
colpito()
se vite finite