CAPITOLO 1: ANALISI

1.1 REQUISITI

L' obiettivo di questo progetto è stato quello di realizzare un' applicazione desktop ispirato al famoso gioco Asteroids, realizzato da Atari nel novembre del 1979.

L' utente deve pilotare una navicella spaziale all' interno della schermata, in un ambiente 2d, e come obiettivo deve distruggere tutti gli asteroidi del livello con le proprie armi (cannone, abilità specifiche).
É stato necessario gestire:
- I vari aspetti del gameplay.
- Le diverse schermate del menu, tra cui la scelta di alcune impostazioni.
- Una classifica che permette di salvare il proprio punteggio.

Per gestire la classifica è necessaria la lettura / scrittura su file, mentre per il gameplay è stato necessario anche l'utilizzo di alcuni file multimediali (immagini e suoni) esterni.

1.2 PROBLEMI

I problemi da risolvere riguardavano:
- la gestione delle entita (navicella, asteroidi, abilita e sparo) e la loro interazione.
- Le interazioni dei vari menu con l' utente.

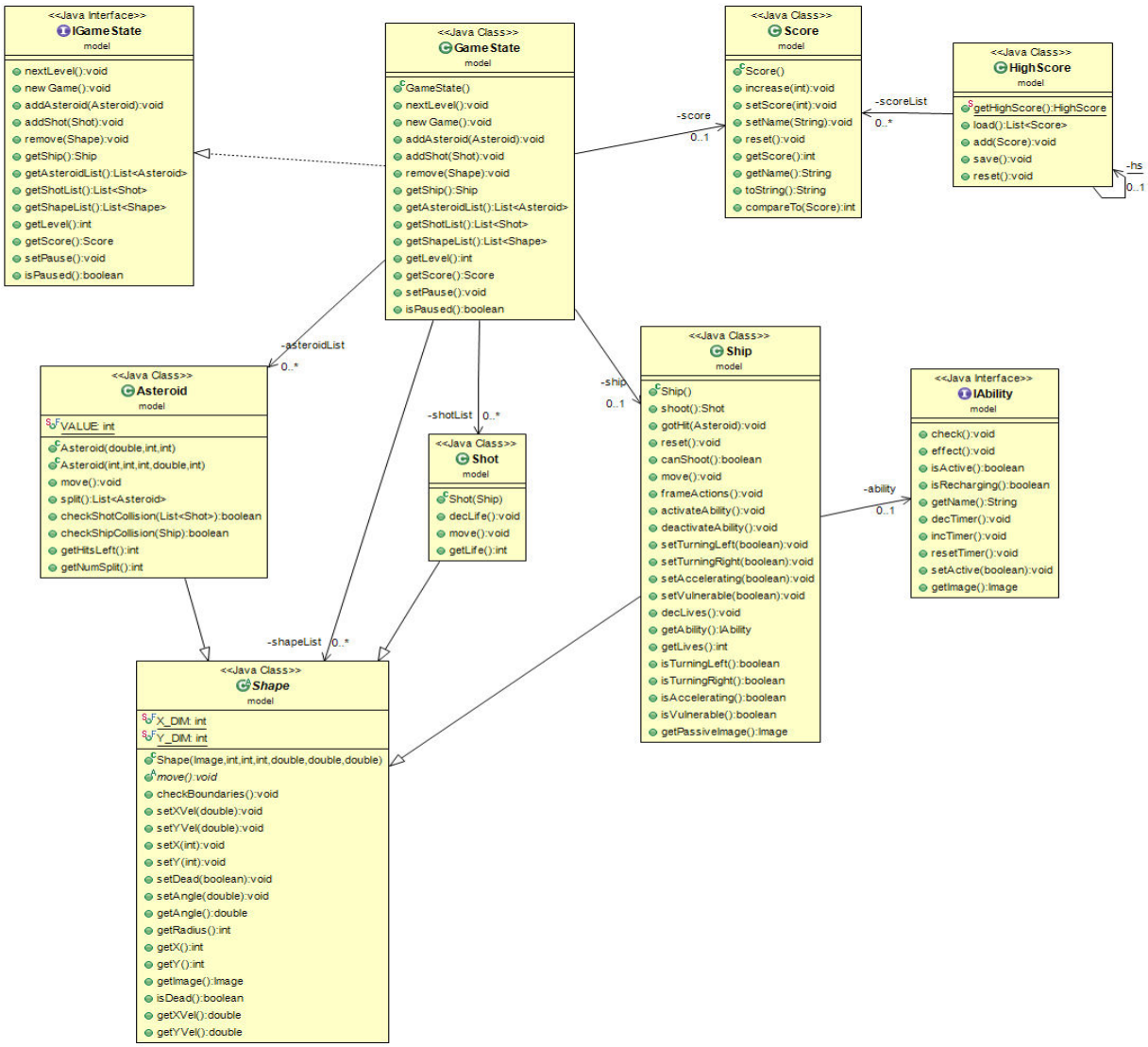
CAPITOLO 2: DESIGN

Per realizzare questo progetto è stato utilizzato il pattern architetturale MVC.

2.1 ARCHITETTURA

MODEL
Classi principali:

GameState: implementa l'interfaccia **IGameState** e mantiene e gestisce tutte le informazioni e i dati riguardo lo stato corrente del gioco.
Shape: modella una generica entità del gioco, da essa estendono le classi **Ship, Asteroid, Shot.**
Ship: modella la navicella controllata dal giocatore.
Asteroid: modella un asteroide.
Shot: modella lo sparo della navicella, utilizzato per distruggere gli asteroidi.
IAbility: interfaccia che modella l'abilità utilizzabile dalla navicella.
HighScore: classe realizzata tramite pattern **singleton,** si occupa di salvare e caricare la classifica dei punteggi.



VIEW

La view rappresenta tutte le interfacce grafiche presenti nell'applicazioni. Per rendere più veloce la visualizzazione all'interno del gioco si è deciso di mantenere un unico frame (MainFrame) che consenta lo scambio di vari pannelli al suo interno; in questo modo viene creato un frame solo all'inizio del gioco e ad ogni necessità viene richiamato il pannello necessario. Le classi principali sono quindi:

- **MainFrame** - frame principale come spiegato qui sopra.
- **MainMenu** che incorpora TitlePanel e vengono caricati all'avvio del gioco; il menù permette tramite appositi bottoni di accedere al primo livello di gioco, alle opzioni, ai credits, agli punteggi salvati o di uscire dal gioco.
- **Credits**, **Scores** e **Options** sono quindi i pannelli richiamati direttamente dal menù iniziale.
- **GamePanel** invece è il pannello in cui si svolgono tutti i livelli del gioco.
- Esistono inoltre le classi **Winner** e **GameOver** che defiscono pannelli legati al superamento o meno di un livello.

Nel package view si trovano tutte le interfacce e classi relative alla parte grafica del gioco e quindi le GUI. Troviamo numerose classi al suo interno, molte seguono una struttura comune poichè rappresentano pannelli simili. In ordine alfabetico sono presenti le seguenti classi:

- **Credits** - Mostra la schermata dei credits relativi all'applicazione.
- **CreditsInterface** - Interfaccia della classe Credits.
- **DrawPanel** - Disegna tutti gli elementi sul pannello di gioco.
- **GameOver** - Mostra la schermata nel momento in cui la partita si conclude.
- **GameOverInterface** - Interfaccia della classe GameOver.
- **GamePanel** - Classe che mostra il pannello di un livello di gioco.
- **GamePanelInterface** - Interfaccia della classe GamePanel.
- **ImageLoader** - Classe Singleton adibita al caricamento delle immagini.
- **MainFrame** - Classe che mostra l'unico frame in cui si alternano i vari pannelli.
- **MainFrameInterface** - Interfaccia della classe MainFrame.
- **MainMenu** - Pannello che mostra il menù principale.
- **MainMenuInterface** - Interfaccia della classe MainMenu.
- **Options** - Classe che mostra il pannello delle opzioni.
- **OptionsInterface** - Interfaccia della classe Options.
- **SaveScore** - Classe che mostra un frame per salvare il punteggio realizzato.
- **SaveScoreInterface** - Interfaccia della classe SaveScore.
- **Scores** - Classe che mostra il pannello con tutti i punteggi salvati.
- **ScoresInterface** - Interfaccia della classe Scores.
- **TitlePanel** - Classe che mostra un pannello con l'immagine del titolo del gioco.
- **TopWords** - Classe che contiene tutte le stringhe principali.
- **Winner** - Classe che mostra un pannello quando si supera un livello.
- **WinnerInterface** - Interfaccia della classe Winner.

CONTROLLER

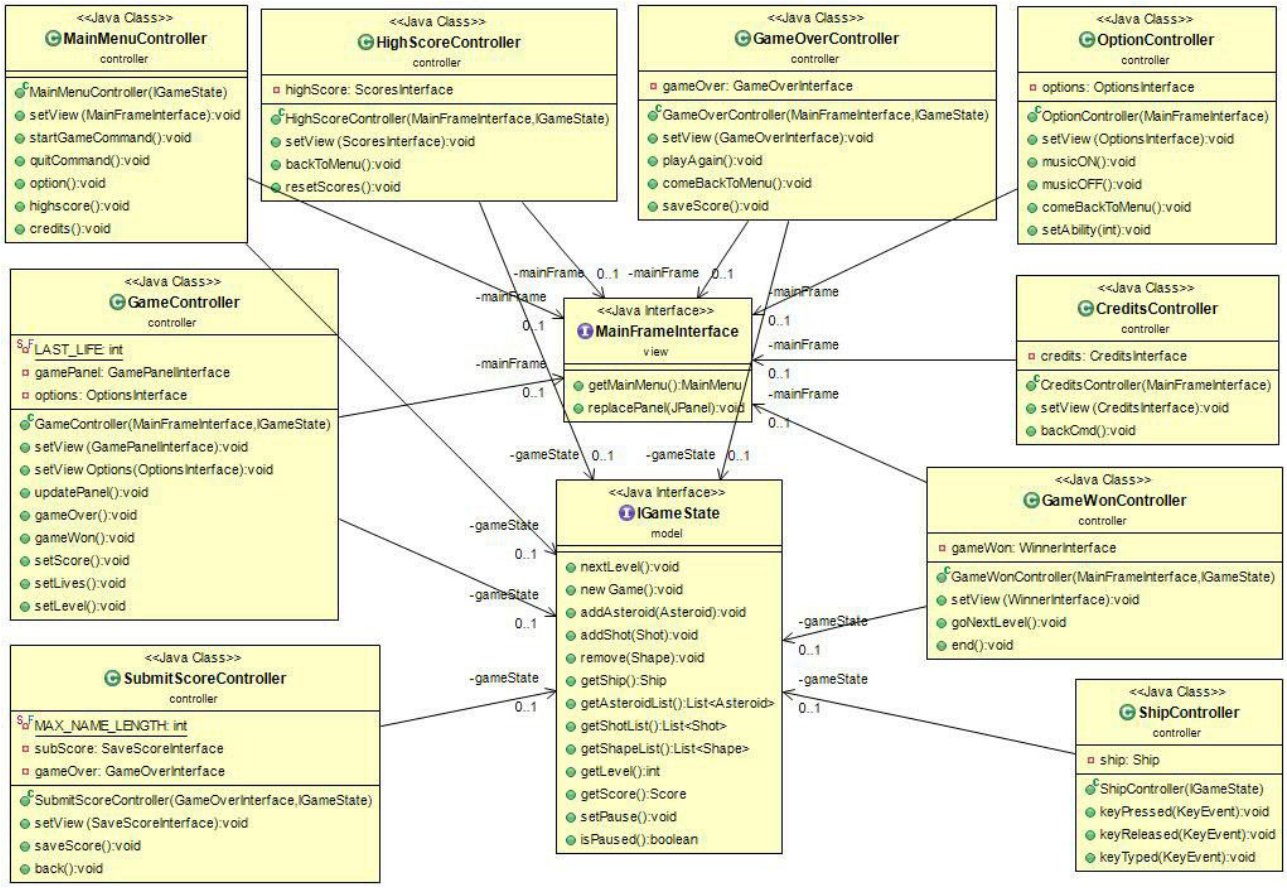


Diagramma semplificato relativo al menu principale.
In particolare mostro le classi controller implementate senza observer e interfacce piu due interfacce che sono oggetto di tutti i controller.

-**IGameState** è l' interfaccia relativa a GameState che fa parte del model: viene usato come oggetto in quasi tutte le classi del controller per accedere ai dati di gioco.

-**MainFrameInterface** è un oggetto presente in tutte le classi controller e permette di cambiare pannello e tornare al menu principale.

Spieghero la struttura dei controller in dettaglio e con il relativo diagramma uml nel paragrafo "Design dettagliato".

Da questi diagrammi risulta evidente che l' applicazione è gestita tramite pattern MVC.
I controller sono l' unica parte dell' applicazione che permette di raggiungere i dati contenuti nel GameState e ad ogni vista è associato un differente controller. In questo modo è il controller stesso a gestire le interazioni dell' utente con l' applicazione e con quella particolare vista.

Risulta facile quindi notare che per ogni classe di view e per la classe GameState è stata definita la relativa interfaccia e come i controller interagiscano con queste interfacce piuttosto che con la classe implementata.

Questa scelta facilita quindi un' eventuale rifattorizzazione o aggiunta di una nuova versione di una classe precedente, in quanto basterà semplicemente implementare l interfaccia relativa, invece che riscrivere parte del controller o doverne aggiungere dei nuovi.

2.2 DESIGN DETTAGLIATO

MODEL

GameState: gestisce tutto ciò che riguarda lo stato di gioco corrente. Mantiene una lista per ogni tipo di entità (tranne la navicella, di cui è presente solo un'istanza), utilizzate per svolgere operazione specifiche. Possiede metodi per cancellare e aggiungere entità. Si occupa inoltre di inizializzare i livelli, ad inizio partita e man mano che si avanza nel gioco. Infine tiene aggiornato il punteggio della partita corrente.

Shape: modella le entità del gioco. Presenta i campi comuni a tutte le entità, tra cui la posizione sul piano, la velocità si spostamento e l'immagine da cui è rappresentata graficamente. Il metodo **move()** (usato per gli spostamenti) è astratto in modo che ogni entità possa esprimere la propria logica di movimento. Contiente il metodo **checkBoundaries()** il quale fa sì che l'entità non abbandoni mai la schermata di gioco: se esce da un lato, rientrerà da quello opposto.

Ship: modella la navicella. Essa è caratterizzata un numero di vite (se scendono a zero il gioco finisce) e presenta un booleano che dice se è attualmente vulnerabile oppure no.. È in grado di sparare, creando nuove istanze della classe **Shot**. Dopo aver sparato deve passare un certo intervallo di tempo prima di poterlo rifare. La navicella possiete un campo **IAbility**, che rappresenta l'abilità speciale utilizzabile dal giocatore. La navicella si può muovere in qualsiasi direzione: attraverso l'input da tastiera può essere fatta ruotare su sé stessa e fatta accelerare. Se avviene una collisione con un asteroide mentre la navicella è vulnerabile, quest'ultima perderà una vita e sarà riposizionata alla posizione iniziale sullo schermo.

Asteroid: modella un asteroide. Una volta che tutti gli asteroidi presenti nel livelli sono distrutti, si passa al livello successivo. Poichè in una collisione è sempre coinvolto un asteroide, i metodi per rilevarle si trovano in questa classe. Quando avviene una collisione, l'asteroide si suddivide in un determinato numero di nuovi asteroidi più piccoli: ciò è reso possibile dal metodo **split()**, che restituisce una lista contenente i nuovi asteroidi. Raggiunto un certo stadio non si suddividono più e sono definitivamente distrutti. Gli asteroidi si muovono in linea retta a velocità costante. Questa classe presenta due costruttori, uno invocato al momento della creazione del livello, il secondo utilizzato dal metodo **split()**.

Shot: rappresenta lo sparo della navicella. Il costruttore ha come paramentro la navicella stessa, che fornisce le informazione necessarie: la posizione in cui creare lo sparo, e la sua velocità. Gli spari si muovono in linea retta. Possiedono un tempo di vita limitato, che viene decrementato ogni volta avviene uno spostamento, e che una volta esaurito comporta l'eliminazione di quello sparo.

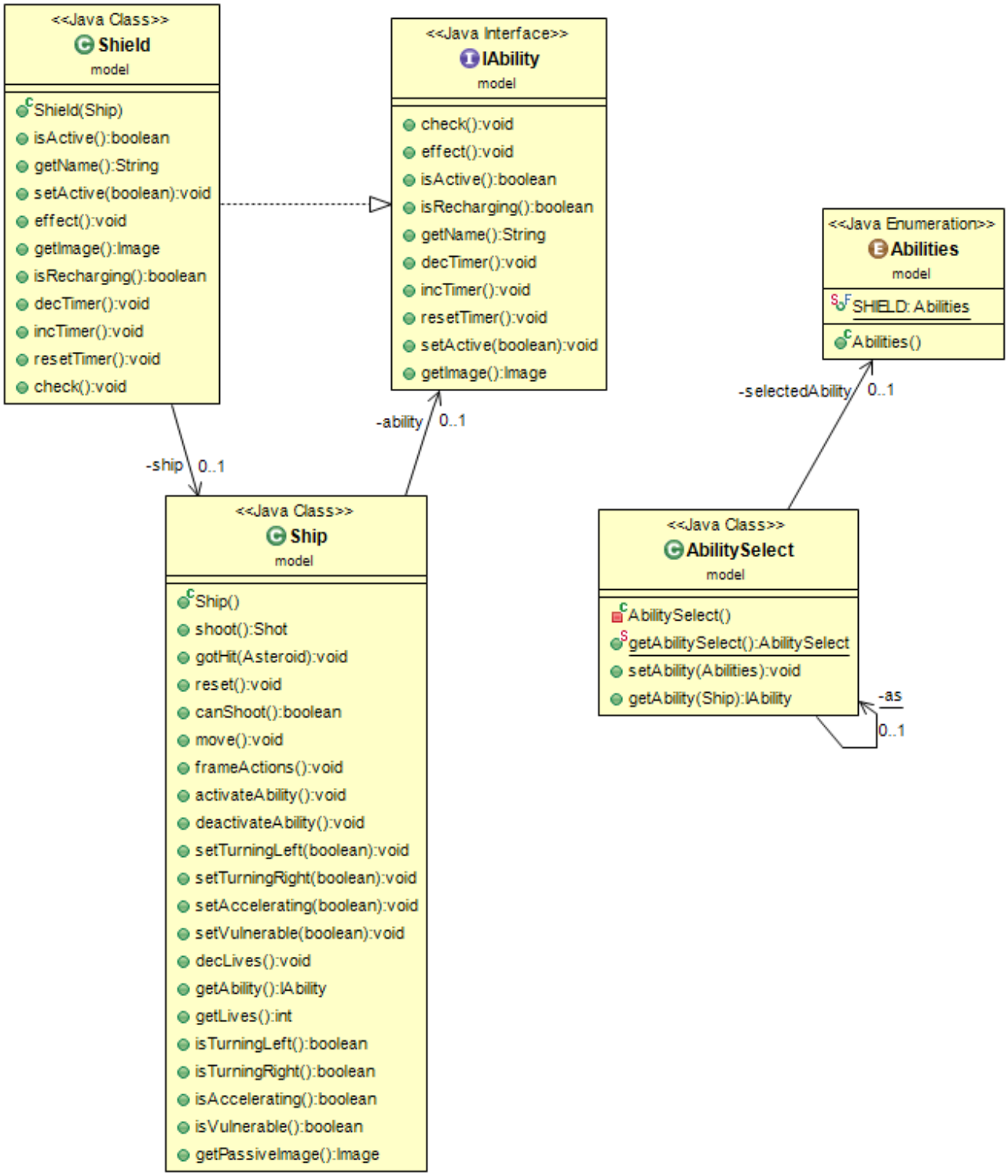
IAbility: interfaccia che modella una generica abilità speciale. In particolare è dotata di metodi per gestire un timer (per limitarne l'uso da parte del giocatore) e un metodo che definisce cosa accade una volta attivata.

Shield: modella l'abilità scudo: esso può essere mantenuta attivo per un certo periodo di tempo prima che abbia bisogno di ricaricarsi. Mentre è attivo rende la navicella invulnerabile. Ciò è reso possibile dedicando alla navicella stessa un campo all'interna della classe **Shield**.

AbilitySelect: classe realizzata tramite il pattern **singleton**, essa presenta un campo in cui è memorizzata l'abilità scelta nelle opzioni di gioco. Quando viene iniziata una nuova partita, **abilitySelect** viene richiamato nel costruttore della navicella, alla quale fornisce l'abilità.

Abilities: enumerazione che contiene i nomi delle abilità presenti nel gioco.

HighScore: classe realizzata tramite pattern **singleton**, è in grado di salvare punteggi su file, caricare e resettare la classifica corrente.
Score: classe che rappresenta il punteggio accumulato durante la partita corrente. Presenta due campi: uno per il punteggio in sé, l'altro per il nome del giocatore, che sarà immesso a partita terminata.



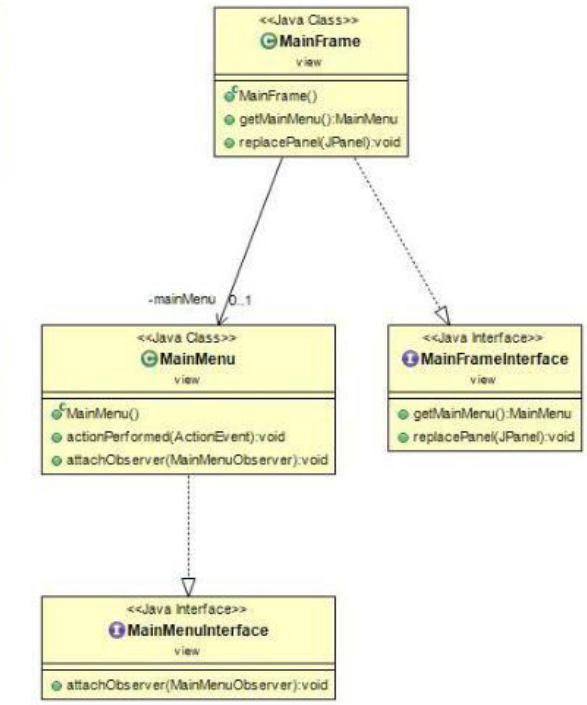
In figura: Schema UML della parte del model riguardante la abilità.

VIEW

MainFrame rappresenta il frame principale durante qualunque operazione nel gioco. Estende quindi la classe JFrame e imposta le dimensioni (width e height) e la posizione (x e y) del frame in base alla risoluzione dello schermo. Una volta chiamato aggiunge il JPanel MainMenu all'interno del frame. Implementa l'interfaccia MainFramelInterface con i metodi getMainMenu() e replacePanel(): il primo restituisce il menù iniziale, il secondo invece permettere di cambiare il pannello attuale con un altro dato in input.

MainMenu che si presenta all'avvio del gioco estende JPanel con un BorderLayout() e presenta al suo interno altri due pannelli: uno contiene un immagine con l'immagine principale del gioco (questo pannello è costruito in una classe a parte, TitlePanel), l'altro invece contiene cinque JButton che consentono di iniziare il gioco o passare ad altri pannelli (Scores, Options, Credits) o uscire dall'applicazione. Il pannello con i bottoni è stato realizzato tramite un GridBagLayout() per permettere di mettere i bottoni in colonna.
Infine MainMenu implementa la sua interfaccia MainMenuInterface che, come in quasi tutte le classi del package view, permette di aggiungere un osservatore tramite il metodo attachObserver() sfruttando il pattern Observer per gestire gli eventi tramite il controller. I metodi descritti dall'interfaccia MainMenuObserver sono: startGameCommand(); quitCommand(); option(); highscore(); credits() implementati nel MainMenuController.

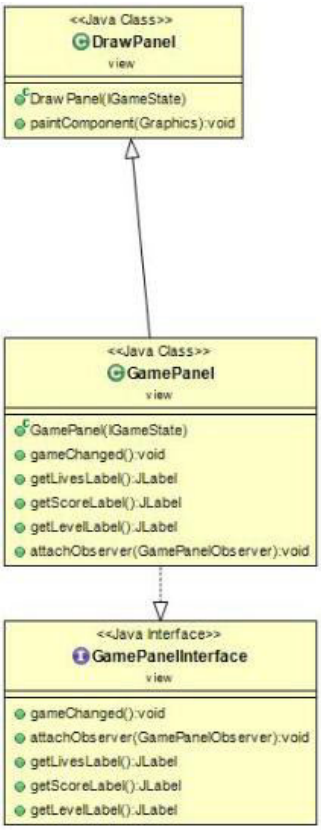
TitlePanel estende Jpanel e permette semplicemente di disegnare l'immagine principale del gioco tramite il metodo paintComponent().



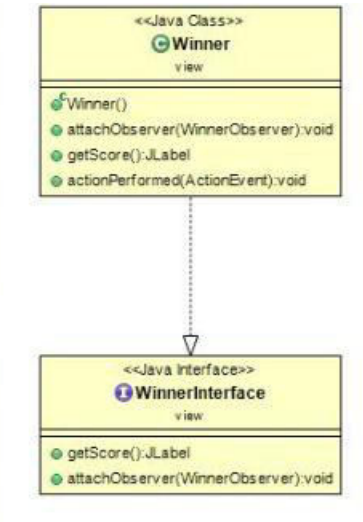
DrawPanel estende JPanel e permette di disegnare tutti i componenti che fanno parte del gioco

(asteroidi, navicella, spari….) richiamando il model che li crea. Tramite l'Override del metodo paintComponent() disegna ogni singolo oggetto. Inoltre permette tramite un AffineTransfrom di ruotare la navicella in base all'angolazione in cui si trova.

GamePanel estende DrawPanel e aggiunge alla schermata del gioco tre label che mostrano le informazioni del gioco: score mostra il punteggio attuale, lives mostra le vite rimanenti, level mostra il livello attuale. Implementa GamePanelInterface con i metodi gameChanged() che tramite il metodo repaint() aggiorna gli elementi grafici del gioco; getLivesLabel() che restituisce il label delle vite rimanenti; getScoreLabel() che restituisce il label del punteggio; getLevelLabel() che restituisce il label del livello; attachObserver() che aggiunge un osservatore. Troviamo anche qui il pattern Observer quindi con i seguenti metodi implementati in GamePanelController: updatePanel(); setScore(); setLives(); setLevel(); gameOver(); gameWon().



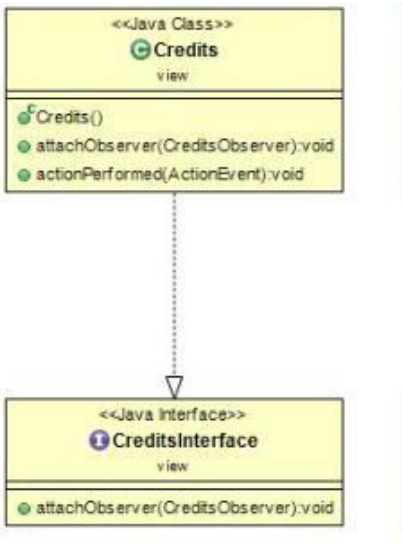
Winner estende JPanel, è realizzato con un BorderLayout() ed è composto a sua volta da altri due panel: il primo contiene l'immagine con il titolo ed è realizzato con un FlowLayout(); il secondo è realizzato con un GridBagLayout() e contiene un label per il punteggio attuale e due bottoni: uno per andare al livello successivo, l'altro per terminare la partita. Implementa WinnerInterface con i metodi getScore() che restituisce il label score e attachObserver() che aggiunge un Observer (WinnerOberserver prevede i metodi goNextLevel() e end().



GameOver estende JPanel, è realizzato con un BorderLayout() ed è composto a sua volta da altri due panel: titlePanel contiene l'immagine con il titolo della schermata ed è realizzato tramite un FlowLayout(), centerPanel è il pannello principale realizzato con un GridBagLayout() e contiene: un label score con il punteggio ottenuto nella partita appena conclusa; tre JButton per iniziare una nuova partita, per salvare il proprio punteggio e per tornare al menù iniziale. Implementa la classe GameOverInterface con i seguenti metodi: getSaveScoreButton() che restituisce il bottone saveScore; getScoreLabel() che restituire il label score; attachObserver() che aggiunge un osservatore. Il GameOverObserver presenta questi metodi (implementati nel controller): playAgain(); saveScore(); comeBackToMenu().

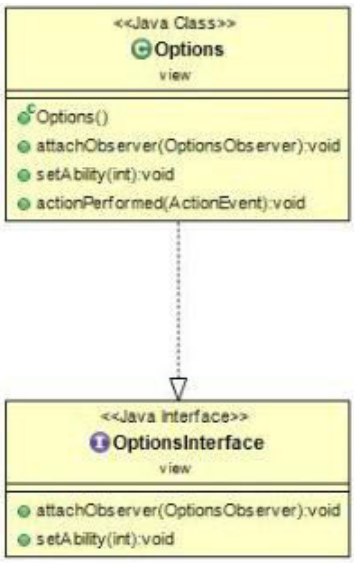
SaveScore è l'unica latra classe che estende JFrame; con questa classe si apre un nuovo frame piccolo frame (400x200) che permette di salvare il proprio punteggio. Contiene un JPanel con all'interno un label per il titolo del frame, una casella di testo per scrivere il proprio nome da salvare e due bottoni che consentono di salvare il punteggio o di chiudere il frame. Implementa SaveScoreInterface che presenta i seguenti metodi: getFrame() che restituisce il frame; getTextField() che restituisce la casella di testo; attachObserver() che aggiunge un osservatore che prevede i metodi SaveScore() e back() implementati in SaveScoreController.

Credits estende JPanel con un BorderLayout() e presenta tre panel: uno per l'immagine di titolo della schermata; uno che contiene i label per i crediti del gioco; uno per il tasto per tornare al menu. Il pannello centrale è realizzato tramite un GridBagLayout() all'interno del quale sono inizializzati i vari campi con le scritte. Implementa CreditsObserver con il solo metodo attachObserver() per aggiungere un osservatore. CreditsObserver contiene solo il metodo backCommand().



Scores estende JPanel con un BorderLayout(); possiede tre panel: quello in alto e in basso con un FlowLayout() contengono rispettivamente l'immagine con il titolo della schermata e i bottoni per tornare al menù e per resettare i punteggi. Questa classe interagisce con la Highscore del model per creare la lista dei punteggi salvati che poi vengono scritti in vari label con un iterazione. I label con i punteggi si trovano nel panel centrale costruito con un GridBagLayout(). Implementa ScoresInterface con il solo metodo attachObserver che aggiunge un osservatore. ScoresObserver contiene i metodi bakcToMenu() e resetScores().

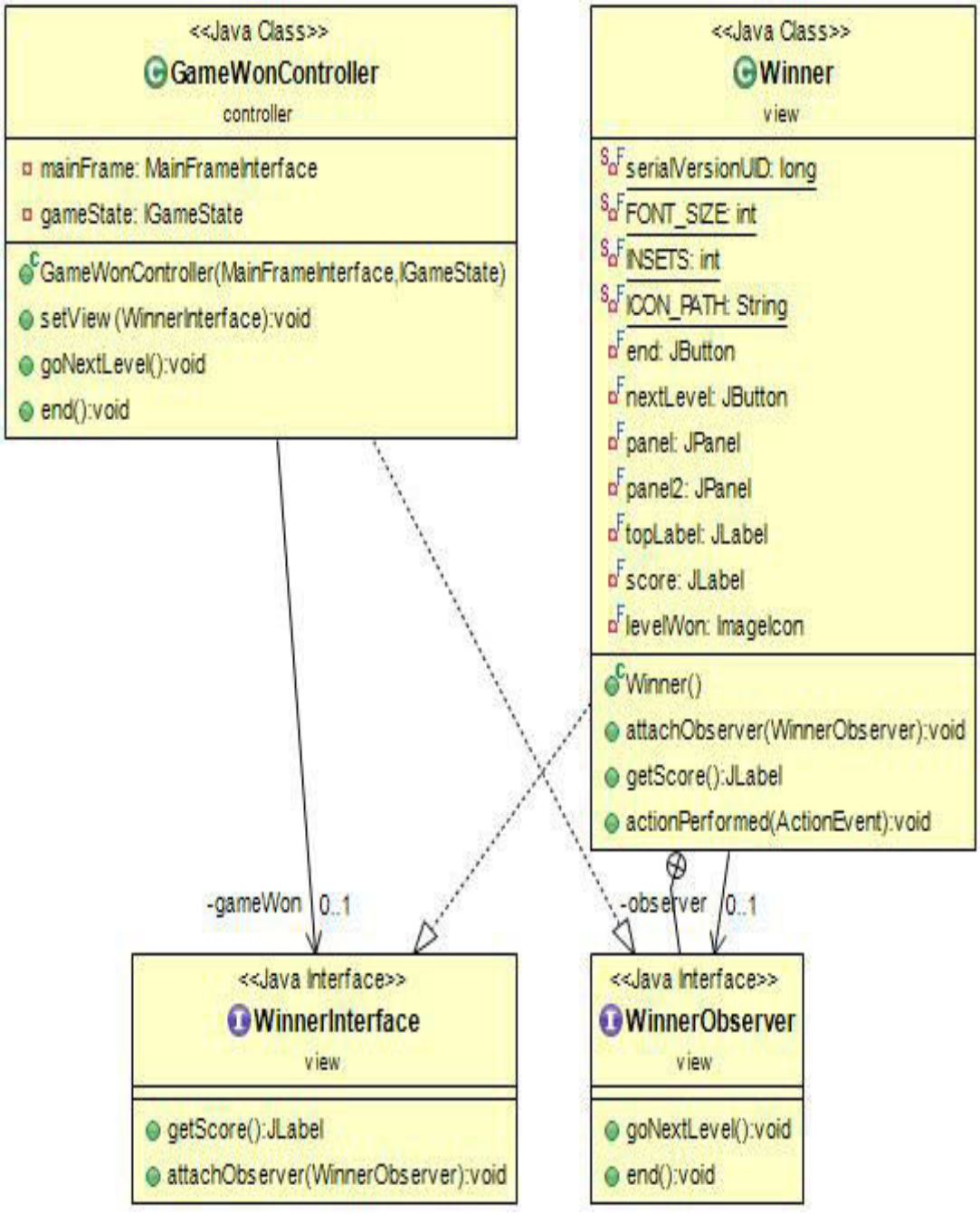
Options estende Jpanel costruito con un BorderLayout(). Troviamo tre panel: il panel in alto contiene solo l'immagine col il titolo della schermata e quello in basso solo il tasto per tornare al menù e sono costruiti con un FlowLayout(). Il pannello centrale realizzato con un GridBagLayout() contiene vari componenti. La prima opzione riguarda la musica e quindi grazie a due JRadiobutton è possibile selezionare il bottone ON oppure OFF. Poi con un JComboBox è possibile selezionare tra le abilità disponibili recuperate nella classe enum corrispondente nel model. Infine sono presenti i label con scritti i comandi per giocare. Implementa OptionsInterface con i metodi setAbility() che richiama l'observer per selezionare l'abilità scelta e attachObserver() che, come nelle altre classi, aggiunge un observer. In questo caso OptionsObserver include i seguenti metodi: musicON(); musicOFF(); setAbility(); comeBackToMenu().



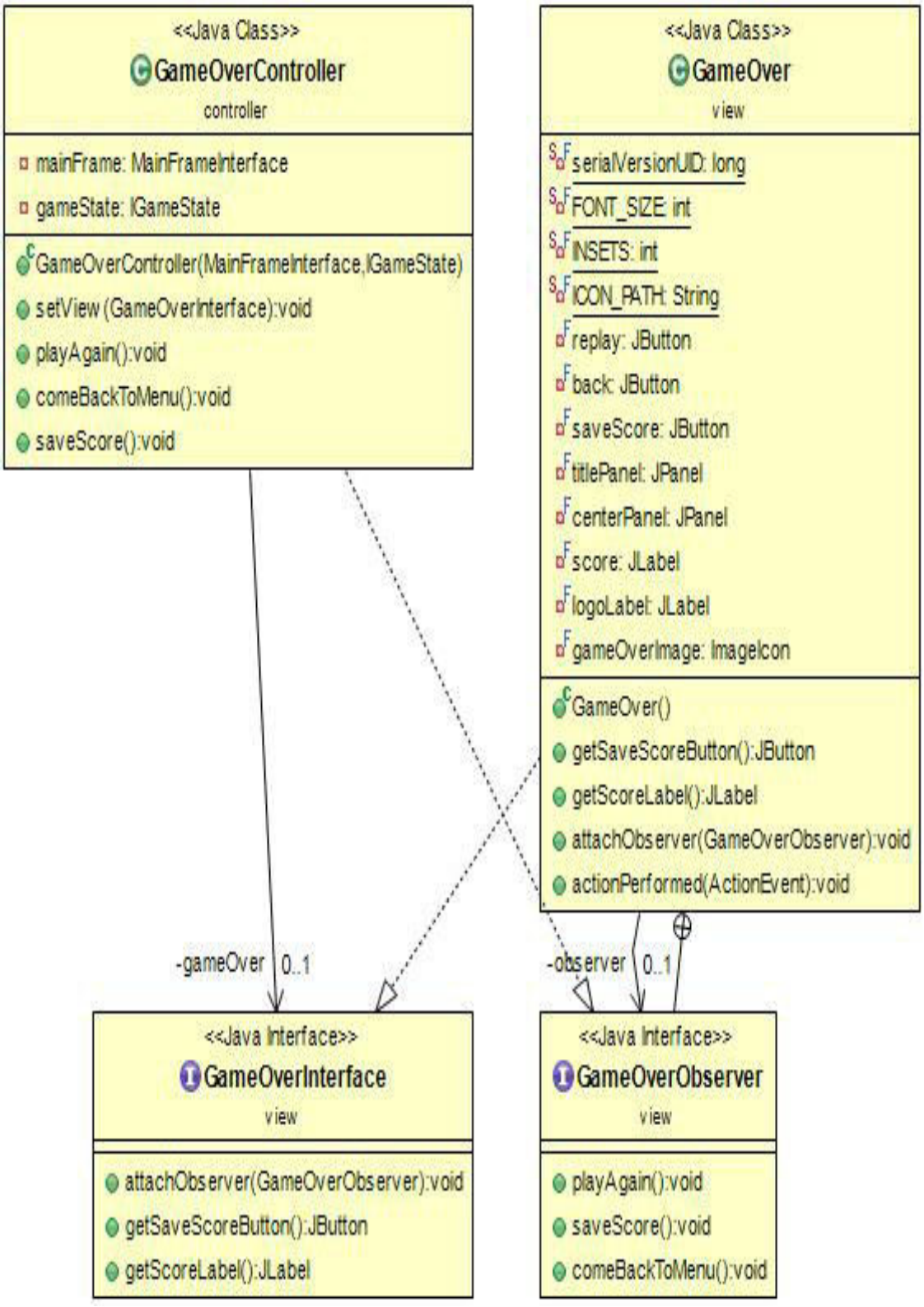
Infine ImageLoader è una classe costruita col pattern Singleton che permette di caricare più velocemente le immagini del gioco. Questa classe inizializza in una mappa tutte le immagini. Grazie al metodo getImageFromPath() che consente di prendere l'immagine da un percorso dato in input e inserirla nella mappa se non è ancora presente, alla fine restituisce l'immagine. Sono poi presenti tutti i metodi per ottenere le singole immagini: getPassiveShip(), getActiveShip(), getShipShield()....

CONTROLLER

-**GameWonController** implementa l interfaccia WinnerObserver e ha come oggetto WinnerInterface**.** Metodi:
-setView imposta la schermata del pannello Winner.
-goNextLevel permette di passare al livello successivo.
-end permentte di tornare al menu principale.



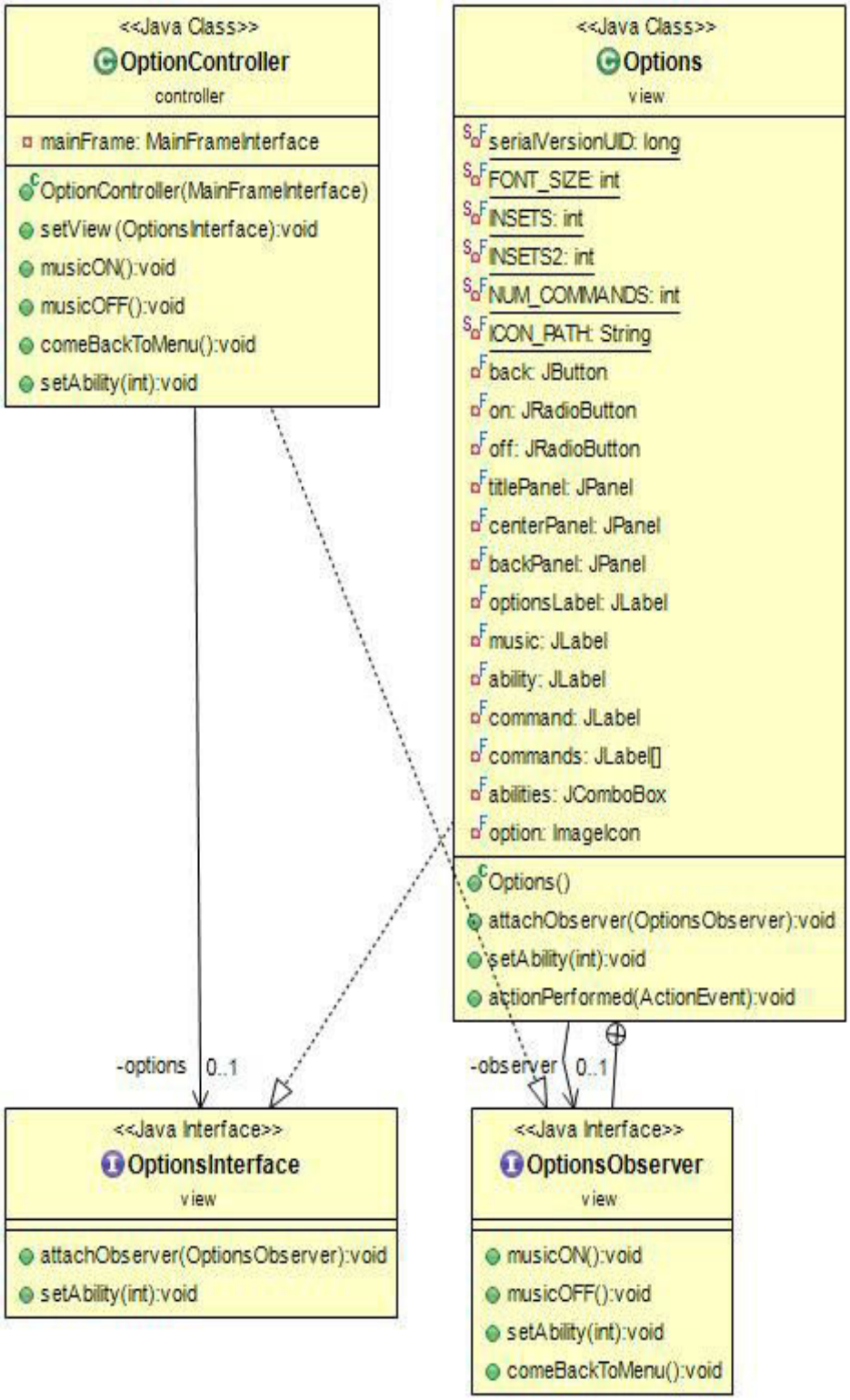
GameOverController implementa l' interfaccia GameOverObserver e ha come oggetto GameOverInterface. Metodi:
-setView imposta la visuale al pannello GameOver.
-playAgain permette di cominciare una nuova partita.
-comeBackToMenu permette di tornare al menu principale.
-saveScore permette di salvare il proprio punteggio.



-OptionController implementa l' interfaccia OptionsObserver e ha come oggetto OptionsInterface.
Metodi:
setView: permette di impostare la visuale al pannello Options.
MusicON: permette di attivare l' audio.
MusicOFF: permette di disattivare l audio.
ComeBackToMenu: permette di tornare al menu principale.

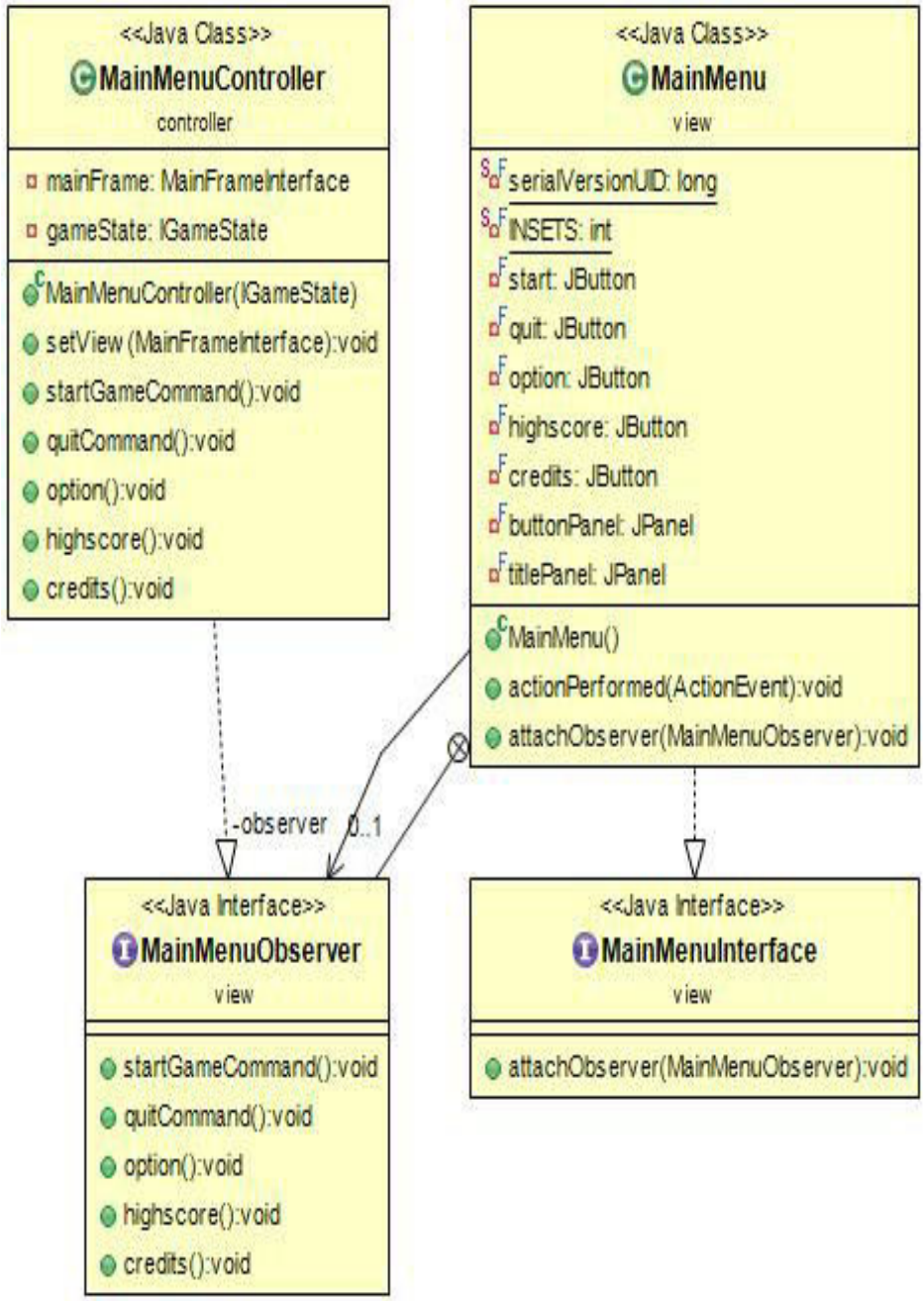
SetAbility permette di impostare un' abilita predefinita.

-

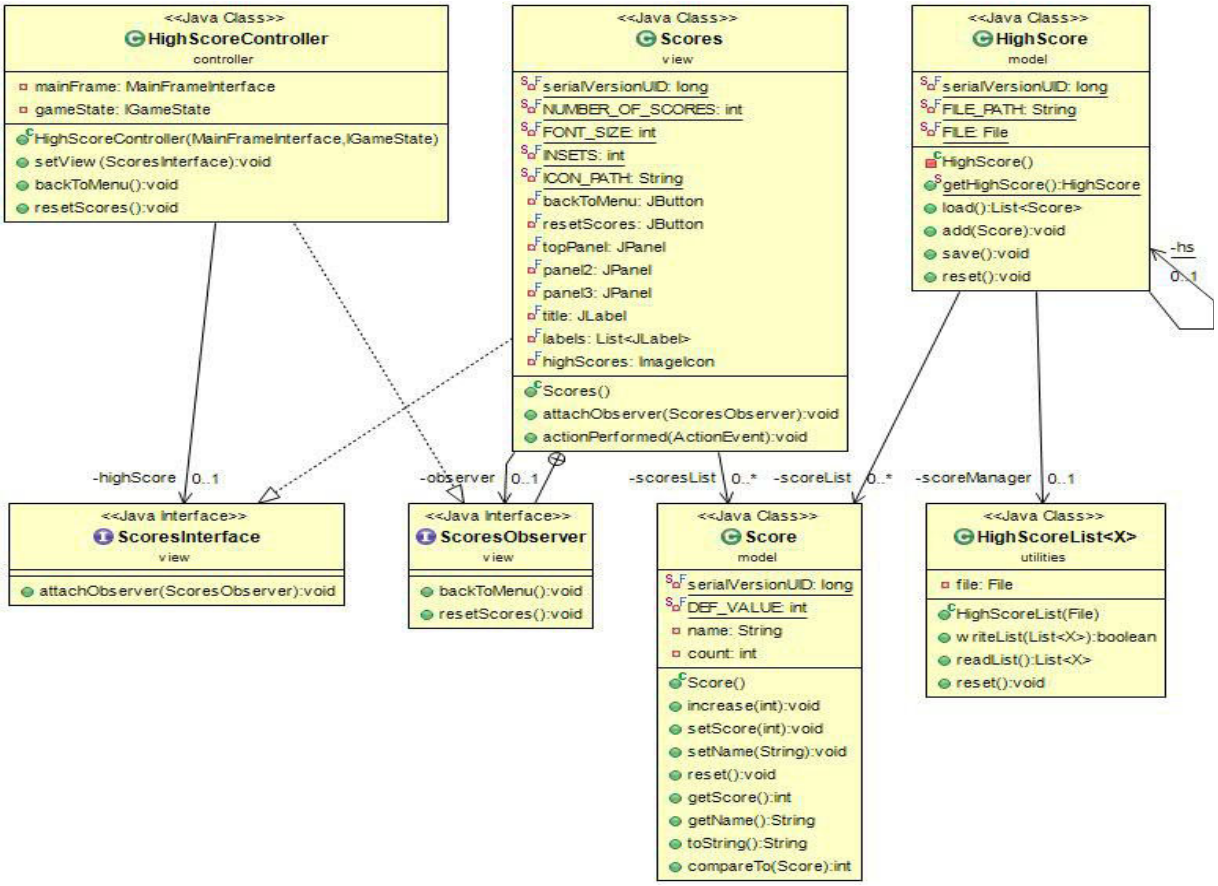


MainMenuController implementa l' interfaccia MainMenuObserver e ha come oggetto MainFrameInterface e IGameState
-setView imposta come pannello corrente il mainMenu.
-startGameCommand permette di inizare una nuova partita.
-quitCommand chiude l' applicazione.
-option accede al pannello opzioni
-highscore accede al pannello classifica.
-credits accede al pannello crediti.

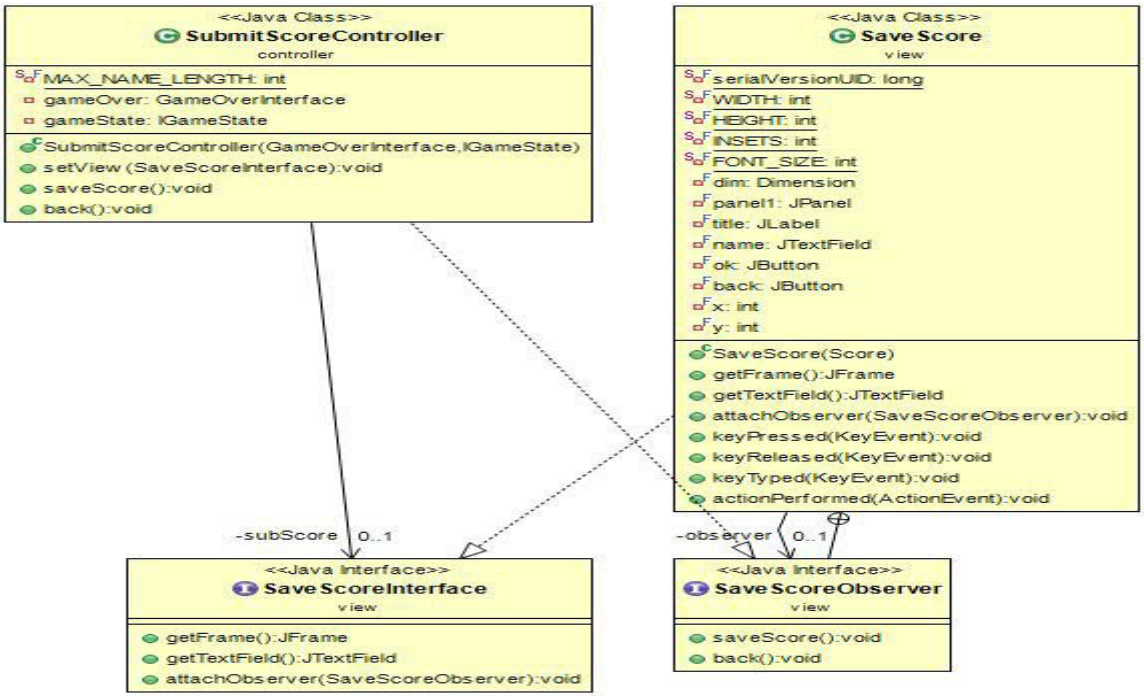
-



HighScoreController implementa l' interfaccia ScoreObserver e ha come oggetto ScoresInterface.
Metodi:
-setView: imposta la visuale al pannello Scores.
-backToMenu: permette di tornare al menu principale.
-resetScore: permette di resettare la classifica.



SubmitScoreController implementa l' interfaccia SaveScoreObserver e ha come oggetto SaveScoreInterface.

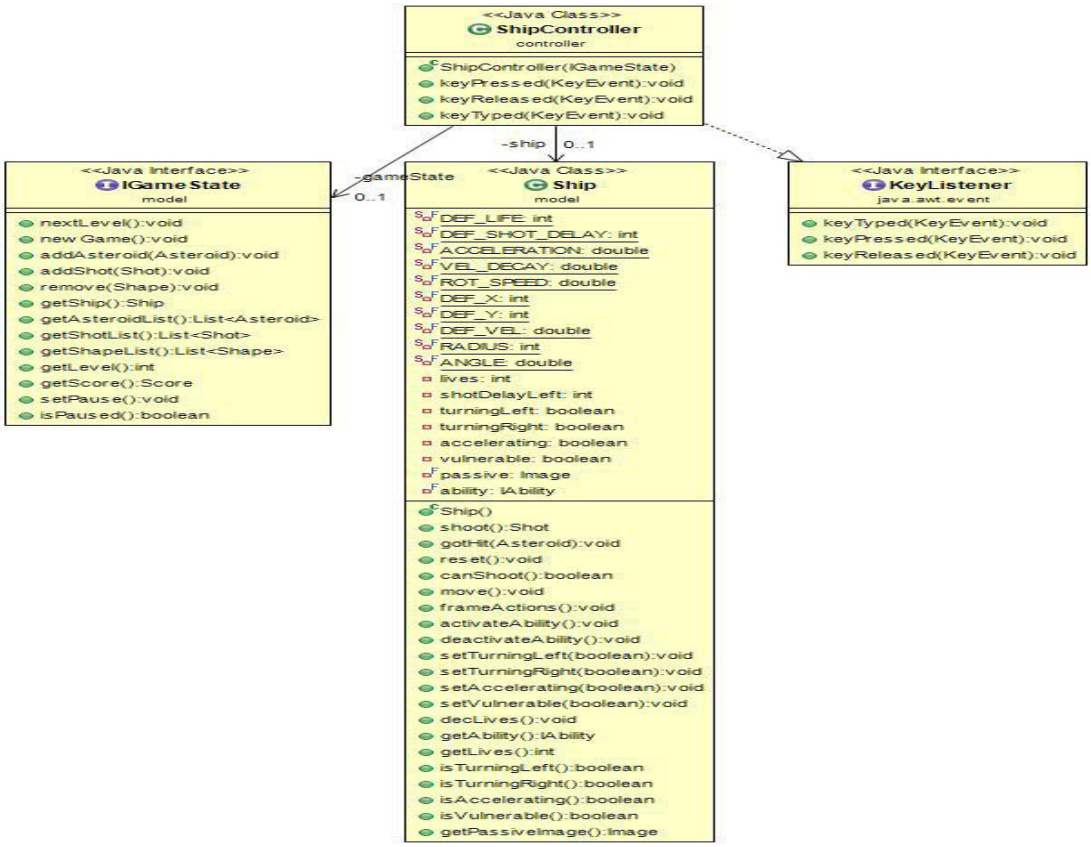


-

ShipController implementa l' interfaccia KeyListener e ha come oggetto l' interfaccia IgameState.

Metodi: keyPressed, keyReleased, keyTyped metodi dell interfaccia KeyboardController.

Il diagramma del controller nel paragrafo 2.1 in particolare spiegava in modo elementare il

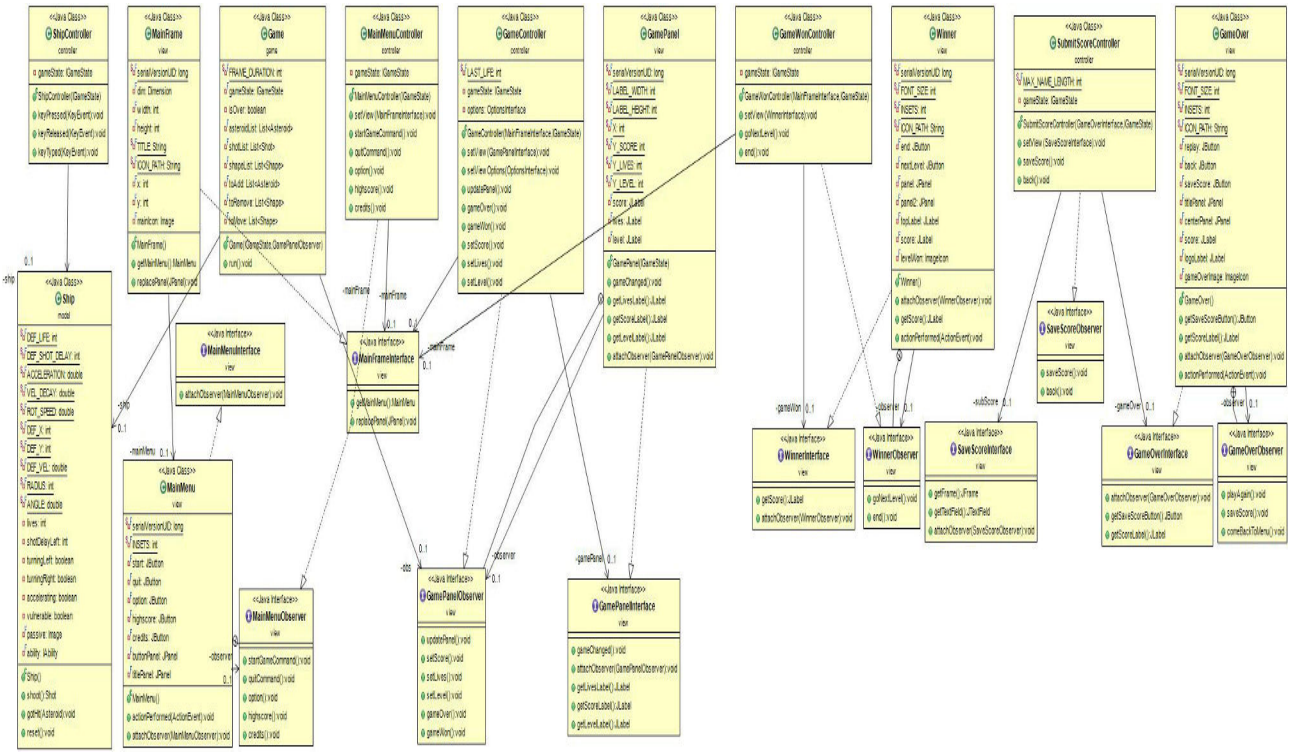


funzionamento della prima schermata di gioco, relativa al menu principale. In particolare:
Il GameController permette di cominciare una nuova partita inizializzando le risorse.
OptionController permette di accedere al pannello delle opzioni, in particolare è possibile:
-atttivare/disattivare il suono
-scegliere l' abilita di gioco (per ora solo lo scudo).
-visualizzare la lista dei comandi di gioco.
-tornare al menu principale

HighScoreController permette di accedere al pannello della classifica dove si possono visualizzare i 10 giocatori che hanno totalizzato i punteggi piu alti, e permette inoltre di resettare la classifica e di tornare al menu principale

CreditsController permette di accedere al pannello per la visualizzazione dei crediti di gioco e di tornare al menu precedente

Vi è infine il pulsante quit game usato per chiudere il gioco.



La seconda schermata è invece relativa a una nuova partita.

In particolare il gameController carica e inizializza le risorse del model e fa cominciare un nuovo thread gioco in cui si può
-visualizzare real time vite disponibili.
-visualizzare real time punteggio.
-visualizzare real time il livello corrente.
-comandare la navicella nello spazio e distruggere gli asteroidi.

Nel caso il giocatore completi il livello, il GameWonController indirizza il giocatore al pannello Winner dove è possibile visualizzare il punteggio, passare al livello successivo oppure tornare al menu prinipale.

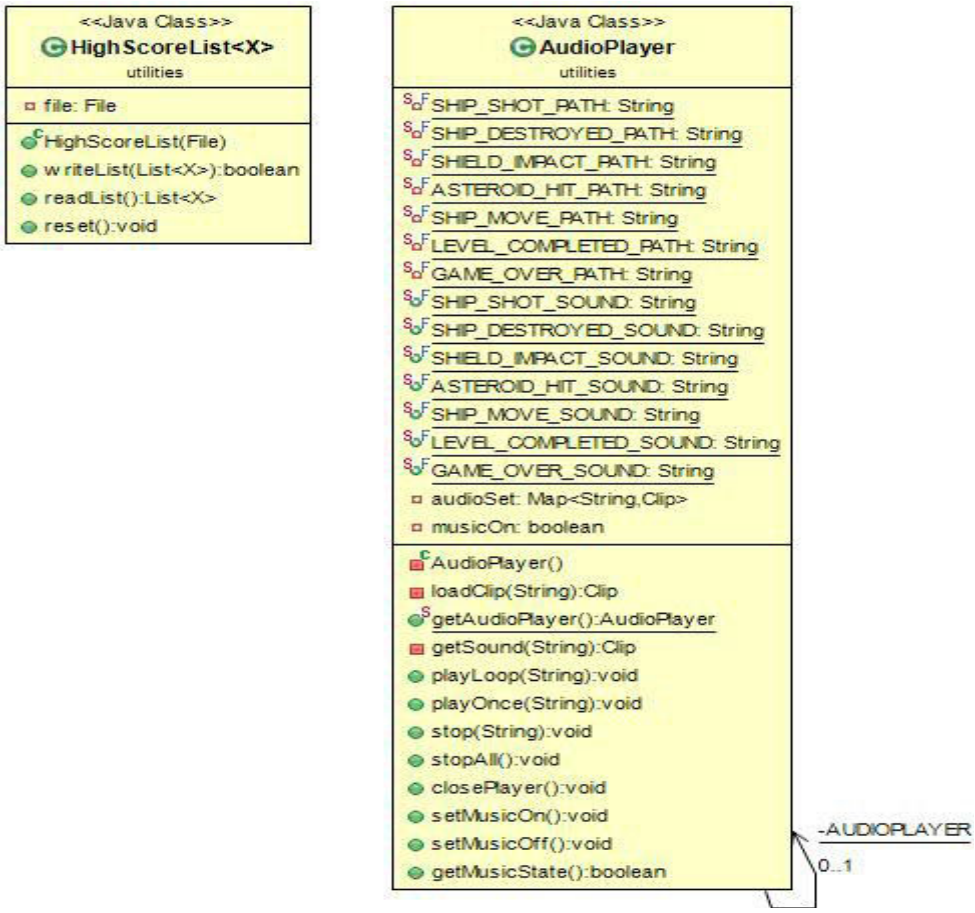
Se invece il giocatore dovesse perdere tutte le vite, il GameOverController indirizza il giocatore al pannello GameOver dove è possibile visualizzare il punteggio effetuato, riniziare subito una nuova partita, tornare al menu precedente o salvare il proprio punteggio, che verrà aggiunto alla highscore list in caso di lista vuota o di punteggio piu alto almeno dell' ultimo punteggio.

Infine espongo le ultime due classi realizzate, che si trovano nell package utilities.

HighScoreList è una classe che serve a scrivere su file la classifica. Metodi:
-writeList: scrive su file una lista e ritorna un booleano che notifica se l' operazione è stata eseguita correttamente oppure no.
-readList: legge su file una lista e ritorna un booleano che notifica se l' operazione è stata eseguita correttamente oppure no.
-Reset: permette di azzarare la lista.

AudioPlayer è una classe che utiliiza pattern singleton è permette di riprodurre effetti sonori duran te la partita, con piu modalita. Metodi:
-loadClip: permette di caricare una clip musicale.

-getSound: ritorna una clip musicale.
-playLoop: riproduce una clip musicale all' infinito.
-playOnce: riproduce una clip musicale una volta.
-stop: stoppa la clip musicale corrente.
-stopAll: stoppa tutte le clip musicali.
-closePlayer: chiude l' AudioPlayer.
-setMusicOn: attiva la musica.
-setMusicOff: disattiva la musica.
-getMusicState: ritorna lo stato corrente della musica (on/off).



Package "Game":

Questo package contiene la classe Game, la quale estende la classe Thread e gestisce il loop principale del gioco. Termina quando il giocatore supera un livello, oppure perde terminando le vite. La classe **Game** ha accesso al model, tramite classe **model.GameState,** che rappresenta e modifica lo stato del gioco, e alla view, attraverso il suo observer. Nel metodo **run()**, se il gioco non è in pausa, ad ogni ciclo vengono mosse le varie entità e controllati e gestitit i vari eventi (come collisioni e spari), vengono rimosse le entità morte e infine chiama la view, che si occupa di aggiornare la schermata.

CAPITOLO 3: TESTING

3.1 TESTING AUTOMATIZZATO

Non sono stati realizzati dei test automatici ma soltanto dei test manuali.

3.2 DIVISIONE DEI COMPITI E METODOLOGIA DI LAVORO

Francesco Dicara

-ha scritto tutte le classi del package asteroids.view.
Ore di Lavoro: 84

Martin Ruini

-ha scritto tutte le classi del package asteroids.model.
Ore di Lavoro: 83

Antonio Pitzus

-ha scritto tutte le classi del package asteroids.controller e del package asteroids.utilities.
Ore di Lavoro: 81.

Insieme

- La classe asteroids.main, le immagini e gli effetti sonori.

3.3 NOTE DI SVILUPPO

Prima di iniziare è stato dedicato del tempo per delle ricerche sul web per trovare qualche buono spunto o base.
In particolare è stato di aiuto il file "Asteroids Manual",[1] un applet che sviluppa il gioco in maniera piuttosto elementare.
Mentre il codice della classifica è stato tratto dal progetto oop13-space.[2], in particolare le classi model.HighScore, model.Score, utilities.HighScoreList.

CAPITOLO 4

COMMENTI FINALI

A causa dell'inesperienza abbiamo avuto difficoltà nella fase iniziale di progettazione e ciò ha causato lo slittamento dell'avvio effettivo del progetto. A causa di ciò non si è riusciti a completare il progetto nel migliore dei modi, completando tuttavia le implementazioni di tutte le funzionalità prefissate.
Funzionalità future: si potrà espandere il gioco aggiungendo nuove abilità, più entità ostili oltre agli asteroidi, gioco multiplayer.

BIBLIOGRAFIA

[1] http://rickbarrette.org/wp-content/uploads/2012/04/Asteroids-Manual.pdf

[2] https://bitbucket.org/Capu93/oop13-space/