UNIVERSITÀ DEGLI STUDI
DI PERUGIA

Tesina finale di

Programmazione di interfacce grafiche e dispositivi mobili

Corso di Laurea in ingegneria informatica ed elettronica – A.A. 2020/2021

DIPARTIMENTO DI INGEGNERIA

Docente

Prof. Luca Grilli

SPACE APOCALYPSE

Applicazione desktop Java Swing

Studenti

306738	Gabriele	Boscarini	gabriele.boscarini@studenti.unipg.it
302326	Alessandro	Federici	alessandro.federici@studenti.unipg.it

1.Descrizione del problema

L’obiettivo di questo lavoro è lo sviluppo di un’applicazione desktop, denominata “Space Apocalypse”, che realizza una versione rivisitata del videogioco arcade Asteroids.

L’applicazione sarà sviluppata utilizzando la tecnologia Java Swing, il codice prodotto sarà testato e ottimizzato per la piattaforma Windows 10.

1.1 Il videogioco Arcade Asteroids

In *Asteroids* il giocatore deve guidare una navicella spaziale attraverso un campo di asteroidi occasionalmente attraversato da dischi volanti: lo scopo del gioco è quello di colpire e distruggere gli asteroidi e i dischi volanti senza entrare in collisione con essi e senza essere colpiti dal fuoco dei dischi volanti. A differenza della maggior parte dei videogiochi arcade, il sistema di controlli di *Asteroids* è composto esclusivamente da pulsanti. I due pulsanti collocati a sinistra permettono di ruotare l'astronave, rispettivamente in senso orario e antiorario. I due pulsanti a destra servono, rispettivamente, a sparare e ad attivare il propulsore per muovere l'astronave. Il movimento dell'astronave è quindi gestito dall'uso combinato di tre pulsanti che richiedono una buona dose di coordinazione tra le due mani, ma anche una grande abilità da parte del giocatore che, a seconda dei casi, deve capire se spostare l'astronave sia la soluzione migliore per sfuggire agli asteroidi o se invece possa essere pericoloso.

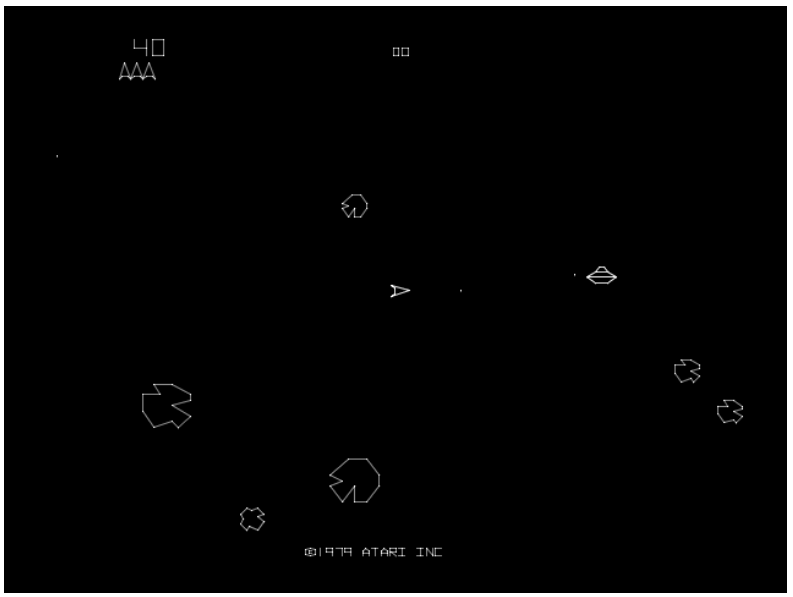


Figura 1: schermata del videogioco Asteroids originale.

1.2 L’applicazione Space Apocalypse

Space Apocalypse è un’applicazione desktop sviluppata usando la tecnologia Java Swing, che realizza una versione rivisitata del videogioco Asteroids mostrato in precedenza. L’applicazione è composta da cinque schermate: il menù principale, il tutorial, la schermata di gioco, il menù di pausa e la schermata di Game Over.

Quando l’applicazione viene lanciata si apre un menu dove è possibile iniziare una nuova partita. La partita inizia con una schermata di Tutorial dove vengono riassunti i comandi principali per giocare. Dopo l’avvio del gioco è possibile muovere la navicella per la mappa di gioco tramite comandi da tastiera (W,A,S,D) e si può far ruotare la sua torretta su se stessa spostando il mouse. Gli asteroidi in questo caso non sono singoli, ma sono blocchi costituiti da più asteroidi di colori diversi ammassati tra loro. Le forme dei blocchi e il numero di asteroidi contenuti in essi sono casuali. I blocchi si spostano nella mappa seguendo una traiettoria lineare. La navicella ha in dotazione più tipi di proiettile di diversi colori, tanti quanti sono quelli degli asteroidi. Il giocatore può decidere il tipo di proiettile utilizzando i numeri presenti sulla tastiera o scorrendo la rotella del mouse e sparare puntando il bersaglio desiderato con il mouse e premendo il tasto sinistro. Per distruggere un asteroide è necessario colpirlo con un proiettile del suo stesso colore, quando ciò accade tutti gli asteroidi dello stesso colore all’interno del blocco che li contiene vengono distrutti. Esiste una probabilità che all’interno di un asteroide siano contenuti dei nemici, che al momento della distruzione dell’asteroide convergeranno verso la navicella. Nel corso del gioco spunteranno anche degli alieni dai lati dello schermo in grado di sparare dei proiettili verso la navicella. Per uccidere un nemico basta colpirlo con uno qualsiasi dei proiettili. Lo scopo del gioco è quello di impedire ai blocchi o ai nemici di raggiungere la navicella. La difficoltà del gioco è data dal

numero di nemici e asteroidi che spuntano in un determinato intervallo di tempo ed è incrementale, se nei primi minuti di gioco compariranno pochi asteroidi, andando avanti nel tempo la mappa ne diventerà piena. Il punteggio assegnato al giocatore aumenta ogni volta che un asteroide o nemico viene distrutto. Quando la navicella viene colpita il gioco finisce e si apre la schermata di GameOver, nella quale il giocatore può inserire il suo nickname(tre caratteri tra lettere o numeri) con un meccanismo ispirato ai classici giochi arcade: delle frecce disegnate a schermo che permettono di scorrere i caratteri e di scegliere una determinata lettera/numero premendo il tasto invio. Nella schermata di GameOver è possibile anche visualizzare il proprio punteggio e quello del campione corrente. Se il proprio punteggio è superiore a quello del campione in carica il vecchio campione viene cancellato e si prende il suo posto. É possibile mettere in pausa una partita cliccando il tasto esc della tastiera, così facendo si apre la schermata di pausa dove si può decidere di continuare la partita, ritornare al menu iniziale(perdendo il punteggio fino ad ora accumulato) oppure terminarla (senza salvare).

2.Specifica dei requisiti

Vengono ora elencati i requisiti richiesti all'applicazione nel suo complesso:

- il pattern architetturale del software è il Model-view-controller.
- Le schermate dell’applicazione sono a schermo intero e devono adattarsi alla risoluzione del dispositivo dove vengono lanciate.
- Le interfacce grafiche del menù principale, del menù di pausa e di GameOver devono avere pulsanti disegnati dagli sviluppatori e resi interattivi tramite l’uso del mouse evitando di usare la classi javax.swing.JButton/javax.swing.JLabel.
- Nella schermata di gioco l’utente può far ruotare la navicella su se stessa spostando il mouse, la navicella avrà una torretta che seguirà il movimento del cursore. Con il tasto sinistro del mouse il giocatore può sparare proiettili che andranno nella direzione in cui punta il cursore. Con i tasti W,A,S,D della tastiera è possibile muovere la navicella che sarà rivolta sempre verso la direzione di percorrenza.
- Le animazioni del gioco devono essere fluide.
- Devono essere presenti effetti sonori, sia nei Menù che durante le animazioni.

- La partita corrente deve poter essere messa in pausa, riavviata oppure terminata.

- La difficoltà deve essere incrementale, allo scorrere del tempo corrisponde un aumento della presenza di nemici nella mappa di gioco.

- Un blocco è costituito da un numero casuale di asteroidi di colori diversi, quando un asteroide viene colpito, tutti gli asteroidi del suo stesso colore presenti nel blocco a cui appartiene vengono distrutti. I blocchi seguono traiettorie lineari.

- I proiettili in dotazione sono di più tipi, uno per ogni colore degli asteroidi, la torretta della navicella assume il colore del proiettile attualmente selezionato.

- Per distruggere un blocco serve distruggere tutti gli asteroidi che lo compongono.

- Quando un asteroide viene distrutto con un proiettile sbagliato al suo posto spunta un nemico diretto verso la navicella.

- Nel corso del gioco spuntano randomicamente degli alieni dai lati dello schermo, i quali sono in grado di sparare dei proiettili contro la navicella.

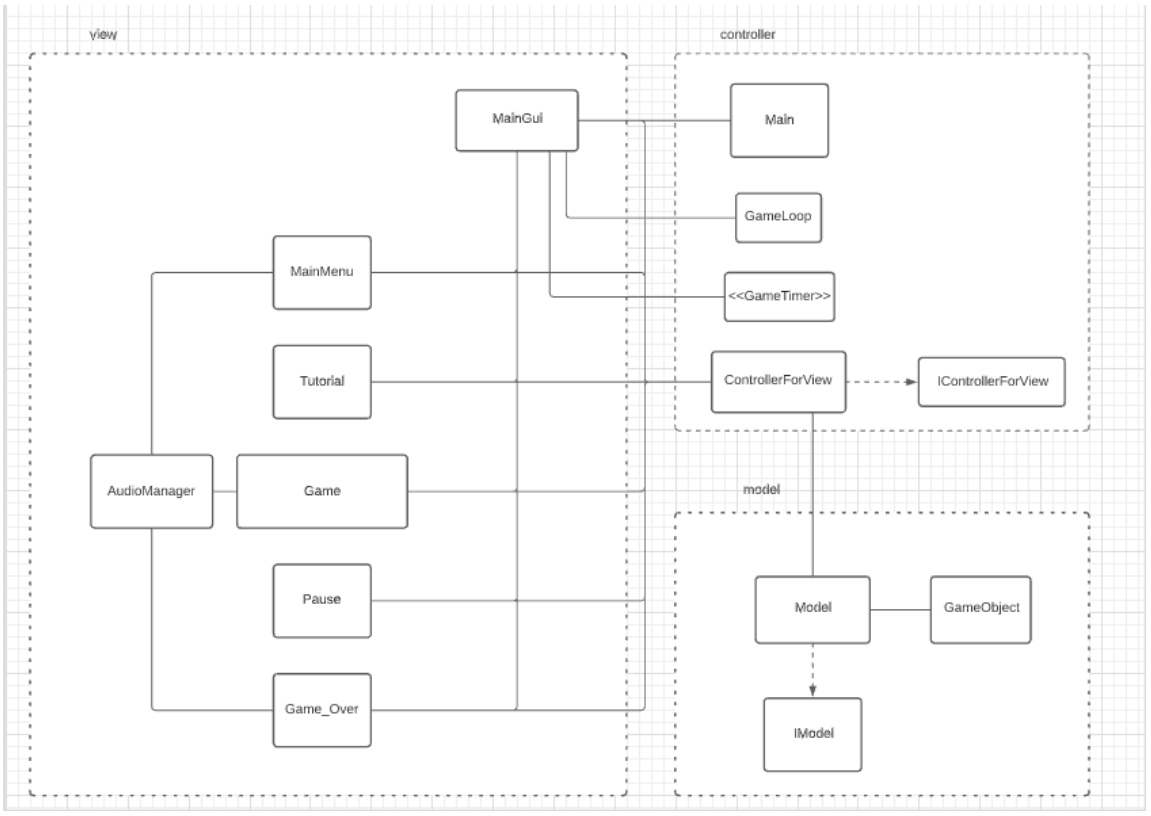
- Il punteggio aumenta ogni volta che viene colpito un asteroide o un nemico e viene salvato a fine partita.

- Nella schermata di GameOver è possibile inserire il proprio nickname e visualizzare il proprio punteggio accanto al punteggio del campione, che è l'utente che fino a quel punto ha raggiunto il punteggio più alto.

3.Progetto

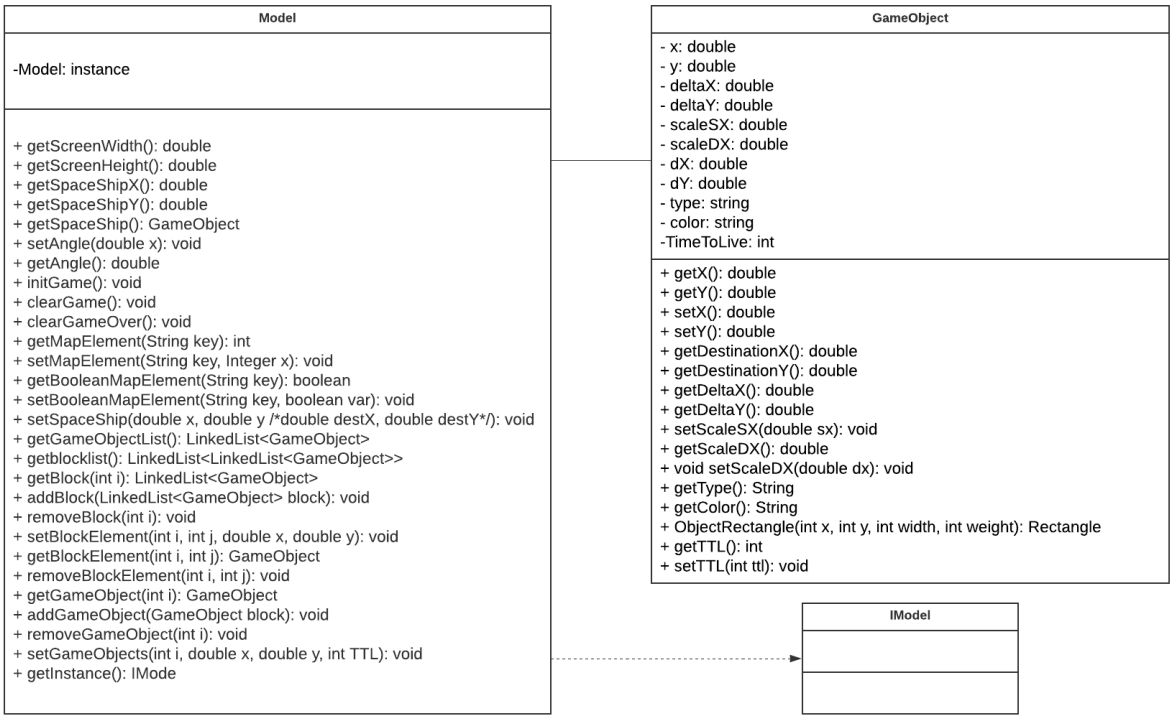
3.1 Architettura software

Il pattern architetturale del software di Space Apocalypse è basato sul MVC. Questa scelta consiste nel compartimentare il codice dell'applicazione in tre package: Model, view, controller. Il Model contiene le istanze di tutti gli oggetti del gioco e i parametri relativi ad essi, insieme alle variabili che regolano il funzionamento e lo stato dell'applicazione.
Il Model comunica direttamente con il Controller, i cui metodi si occupano di modificare lo stato e il comportamento degli oggetti di gioco rendendo possibile il movimento, la creazione e rimozione degli oggetti, la modifica delle variabili che pilotano la visualizzazione dei pannelli dell'applicazione, l'aggiornamento del punteggio.
La view si occupa della visualizzazione di tutti gli elementi dell'applicazione gestendo il caricamento delle schermate e dei pannelli. Nella View vengono caricati tutti i file delle immagini assegnate agli oggetti (nel nostro caso immagini di tipo BufferedImage) e si impostano i fattori di scala. La view si occupa anche di ricevere gli input dell'utente e inviarli al Controller in modo tale da essere gestiti.
La direzione della messaggistica tra i package è quindi: Model <-- Controller <--View.
Una descrizione più dettagliata è espressa dal seguente diagramma UML:



3.2 Model

Le classi appartenenti al blocco Model di Space Apocalypse si trovano nel package model. La loro struttura è rappresentata nel seguente diagramma UML:



3.2.1 La classe Game Object

Gli oggetti di gioco presenti in Space Apocalypse sono di più tipi e di diversi colori ed è possibile generarli tramite una sola classe.

La classe Game Object realizza gli oggetti di gioco. Ogni oggetto viene creato passando come parametri al costruttore della classe: le coordinate della sua posizione nella mappa di gioco, le coordinate della destinazione che l'oggetto deve raggiungere, il tipo e il colore. I tipi possibili sono: **asteroid** (rappresenta gli asteroidi), **bullet** (rappresenta i proiettili), **enemy** (rappresenta i nemici di tipo 1), **enemy2** (rappresenta i nemici di tipo 2), **broken** (Asteroidi colpiti da un proiettile e quindi andati in frantumi), **enemy_broken, enemy_2_broken** (Nemici colpiti da un proiettile e quindi andati in frantumi), **enemy_bullet** (I proiettili che sparano i nemici di tipo 2). I colori possibili sono: orange, blue, pink, green.

Il metodo **ObjectRectangle** restituisce un rettangolo che viene creato ricevendo come parametri le coordinate della posizione dell'oggetto, la larghezza e la

lunghezza dei lati. Questo rettangolo serve per dare una dimensione spaziale all'oggetto permettendo di determinare quando collide con altri oggetti.

3.2.2 La classe Model

La classe Model, che implementa l'interfaccia Imodel, è un contenitore delle istanze di tutti gli oggetti di gioco e delle variabili che descrivono lo stato dell'applicazione. Gli oggetti vengono memorizzati in due liste principali: gameObjects e blocklist. La prima è una LinkedList di tipo GameObject che memorizza tutti quegli oggetti che non hanno bisogno di essere raggruppati in blocchi, cioè quelli a cui NON viene assegnato il tipo "asteroid". La seconda è una LinkedList che contiene al suo interno altre LinkedList che svolgono la funzione di "blocchi di asteroidi", infatti il contenuto di queste liste interne è un insieme di oggetti GameObject a cui è assegnato il tipo "asteroid". Questo meccanismo serve a raggruppare gli asteroidi in blocchi, visto che nelle specifiche è spiegato che quando un singolo asteroide viene colpito ciò ha delle ripercussioni su tutto il blocco a cui appartiene. La maggior parte dei metodi della classe servono ad aggiungere ed estrarre gli elementi dalle liste, memorizzarli in variabili di appoggio e manipolarli in base a quanto viene deciso dal Controller.

In particolare il metodo:

- **LoadBooleanVar** Inserisce tutte le variabili di tipo Boolean all'interno di una mappa contenente chiavi di tipo String.

- **LoadIntVar** inserisce tutte le variabili di tipo Boolean all'interno di una mappa contenente chiavi di tipo String.

- **clearGame** svuota le liste in seguito ad un reset del gioco.

- **clearGameOver** azzera gli indici delle liste presenti nel pannello di GameOver nella View.

3.2.3 La classe GameLoop

I meccanismi di Timing di Space Apocalypse sono affidati ad un Thread indipendente dall'event dispatch thread a cui è assegnato il nome "gameloop".
La classe GameLoop implementa l'interfaccia Runnable sovrascrivendo il metodo astratto **run.** Nel corpo del metodo **run** viene eseguito all'interno di un ciclo while il metodo **timeWarp,** che gestisce il tempo di sleep (30 millisecondi) del Thread gameloop. Ad ogni ripetizione del ciclo viene eseguito anche il metodo

updateGame, un metodo definito nella classe MainGui (contenuta nel package View) di cui si avrà una descrizione più raffinata nel paragrafo 3.4.1.
Il metodo **start** crea il thread gameloop, lo avvia ed esegue il metodo **run.**

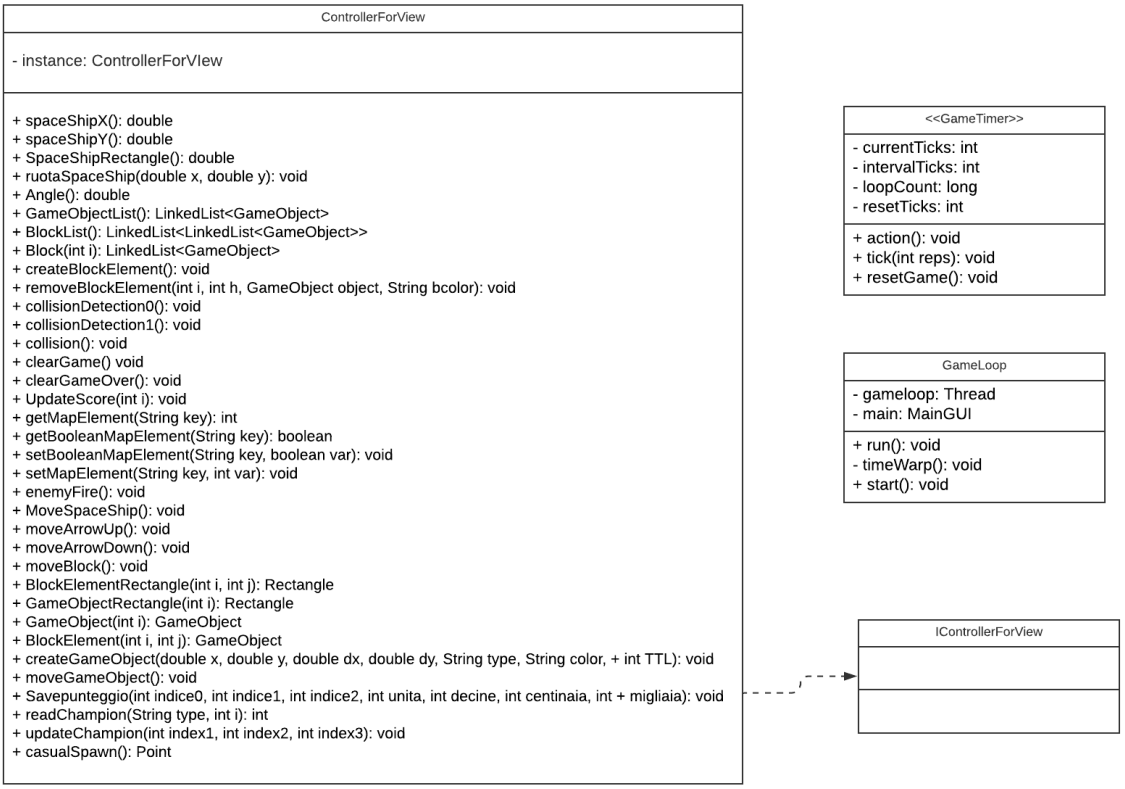
3.2.4 La classe GameTimer

La classe GameTimer crea un oggetto paragonabile al Timer che Java mette a disposizione: è infatti un "lanciatore" di eventi, che in questo caso non vengono gestiti nell'Event Dispatch Thread ma dal metodo **action,** un metodo astratto che può essere implementato ogni qualvolta occorra un Timer per pilotare una determinata serie di eventi (ad esempio lo spawn degli oggetti di gioco). Al costruttore della classe viene passato un intero come parametro (chiamato interval) che indica la "frequenza" del Timer, cioè ogni quanto tempo deve essere eseguito il metodo **action.**
Il metodo **tick** viene chiamato nel metodo **UpdateGame** introdotto in precedenza e contiene un meccanismo che fa eseguire il metodo **action** ogni n interval.
In Space Apocalypse la difficoltà del gioco consiste nella frequenza di spawn degli asteroidi all'interno della mappa di gioco, spawn che sono pilotati da un oggetto di tipo GameTimer. Per rendere progressivo l'aumento della difficoltà, viene passato al metodo **tick** un parametro che indica ogni quante ripetizioni di **action** calare il valore di interval cioè aumentare la frequenza del Timer e di conseguenza la frequenza degli spawn degli asteroidi.

Il metodo **resetGame** resetta la frequenza del Timer al valore di partenza.

3.3 Controller

Le classi appartenenti al blocco Controller di Space Apocalypse si trovano nel package controller. La loro struttura è rappresentata nel seguente diagramma UML:



3.3.1 La classe ControllerForView

La classe ControllerForView, che implementa l'interfaccia IcontrollerForView, implementa la logica dell'applicazione, i suoi metodi infatti si occupano di modificare le variabili che gli vengono passate dalla classe Model rendendo possibile la creazione/rimozione degli oggetti di gioco, il loro movimento e la gestione delle collisioni che avvengono nel pannello di gioco. Oltre ad agire sulle istanze degli oggetti di gioco, ControllerForView manipola anche le variabili che pilotano lo stato dell'applicazione. I primi metodi della classe si occupano di richiamare i metodi del Model che restituiscono le istanze degli oggetti.

In particolare il metodo:

- **createBlockElement** sceglie randomicamente un punto da uno dei 4 lati dello schermo, da dove creerà un blocco di asteroidi. Crea una LinkedList chiamata "block" che verrà passata come parametro al metodo del Model **addBlock**, che si occupa di aggiungere un elemento alla lista che contiene tutti i blocchi di asteroidi. Inizializza un array che contiene i possibili colori che un asteroide può assumere e crea una variabile di tipo Random che sceglierà randomicamente il colore che ogni asteroide assumerà.
  Viene avviato un ciclo for che crea una matrice di punti equamente distanziati, e per garantire che i blocchi abbiano una forma diversa decide in base ad una

variabile Booleana randomica se creare un asteroide o meno in ognuno dei punti.

- **moveBlock** Si occupa del movimento degli asteroidi. Due cicli for annidati scandiscono tramite indice due liste: La prima è quella che contiene tutti i blocchi, la seconda rappresenta i singoli asteroidi all'interno del blocco. Una volta avuto accesso al singolo asteroide viene chiamato il metodo del Model **setBlockElement,** a cui vengono passati come parametri le coordinate attuali dell'oggetto sommate al valore del delta del primo asteroide del blocco(La differenza tra le coordinate della destinazione e le coordinate attuali dell'oggetto) e moltiplicate per un fattore che determina la velocità del movimento. Se le coordinate di un asteroide escono dal confine dello schermo l'intero blocco viene eliminato per far spazio nella lista e non appesantire l'esecuzione del codice.

- **moveGameObject:** Si occupa del movimento di tutti gli oggetti che non sono asteroidi. La lista gameobject viene scandita tramite indice con un ciclo for e una volta avuto accesso all'oggetto viene chiamato il metodo del Model **setGameObject,** a cui vengono passati gli stessi parametri usati per il movimento degli asteroidi, solamente che in questo caso il fattore che determina la velocità può variare in base al tipo di oggetto. Il parametro TTL (time to live) viene diminuito ad ogni chiamata del metodo, quando questo valore diventa negativo l'oggetto viene rimosso dalla lista.

- **MoveSpaceShip** Si occupa del movimento dell'astronave, la quale può spostarsi nelle quattro direzioni cardinali. L'utente può cambiare la direzione della navicella tramite i tasti W,A,S,D della tastiera. Quando uno di questi tasti viene premuto la variabile Booleana che sblocca il movimento in quella direzione viene settata a true e le coordinate della navicella vengono aggiornate tramite il metodo **setSpaceship.** Se l'astronave esce da un lato dello schermo le sue coordinate vengono settate per farla spuntare dal lato opposto.

- **checkCollision** Rileva una collisione tra due rettangoli.

- **collisionDetection1** Si occupa di gestire le collisioni tra un asteroide e un qualsiasi oggetto della lista gameobjects (che contiene anche i proiettili di tutti i colori). Un primo ciclo for scandisce la lista blocklist, un secondo ciclo scandisce, dato un blocco, tutti gli asteroidi del blocco. Una volta avuto accesso al singolo asteroide viene prima verificata e gestita un'eventuale collisione con la navicella, poi viene avviato un ciclo for per scandire la lista gameobject e gestire un eventuale collisione con un proiettile. Se la collisione è verificata, viene rimosso dalla lista gameobjects il proiettile tramite il metodo

removeGameObject e viene chiamato il metodo **removeBlockElement,** descritto in seguito.

- **collisionDetection0** Si occupa di gestire le collisioni tra oggetti appartenenti alla stessa lista: gameobjects. Un ciclo for scandisce la lista tramite indice, se un oggetto è di tipo “enemy” viene prima gestita un’eventuale collisione con la navicella, poi viene avviato un ciclo for per scandire di nuovo la lista gameobjects e verificare un’eventuale collisione con qualsiasi oggetto che non sia se stesso. Se la collisione accade, viene creato un oggetto di tipo “enemy_broken”, ovvero l’animazione di un nemico colpito.

- **removeBlockElement** Viene chiamato dal metodo **collisionDetection1** nel caso in cui un asteroide venga colpito. Se un asteroide viene colpito con il proiettile del colore giusto il metodo si occupa di rimuovere tutti gli asteroidi dello stesso colore all’interno del blocco, altrimenti rimuove il singolo asteroide, crea un nemico nel punto dove è avvenuta la collisione e crea un oggetto di tipo “broken” per animare la rottura dell’asteroide. Il metodo riceve come parametri l’indice del blocco in cui agire, l’indice dell’asteroide colpito, un riferimento all’oggetto dell’asteroide colpito e il colore del proiettile. Se il colore dell’asteroide e del proiettile combaciano viene scandita tramite indice la lista che contiene gli asteroidi appartenenti al blocco e vengono eliminati dalla lista gli asteroidi con lo stesso colore di quello colpito.

- **collision** viene chiamato ad ogni ciclo di aggiornamento dello stato dell’applicazione, si occupa di richiamare **collisiondetection0** e **collisiondetection1.**

- **spaceShipHit** gestisce la collisione di qualsiasi oggetto con la navicella.

- **moveArrowUp** incrementa l’indice della lista che contiene gli oggetti di tipo BufferedImage che servono per rappresentare il punteggio nel pannello “GameOver”. Il metodo **moveArrowDown** è simmetrico.

- **increaseScore** si occupa di aggiornare il punteggio corrente del giocatore. Riceve come parametro un intero che indica di quanto aumentare il punteggio. Aggiorna la variabile intera “score” e avvia un meccanismo che consente di aggiornare l’indice della lista stanziata nel pannello “GameOver”, di cui si ha una descrizione completa più avanti (paragrafo: **3.4.4**)

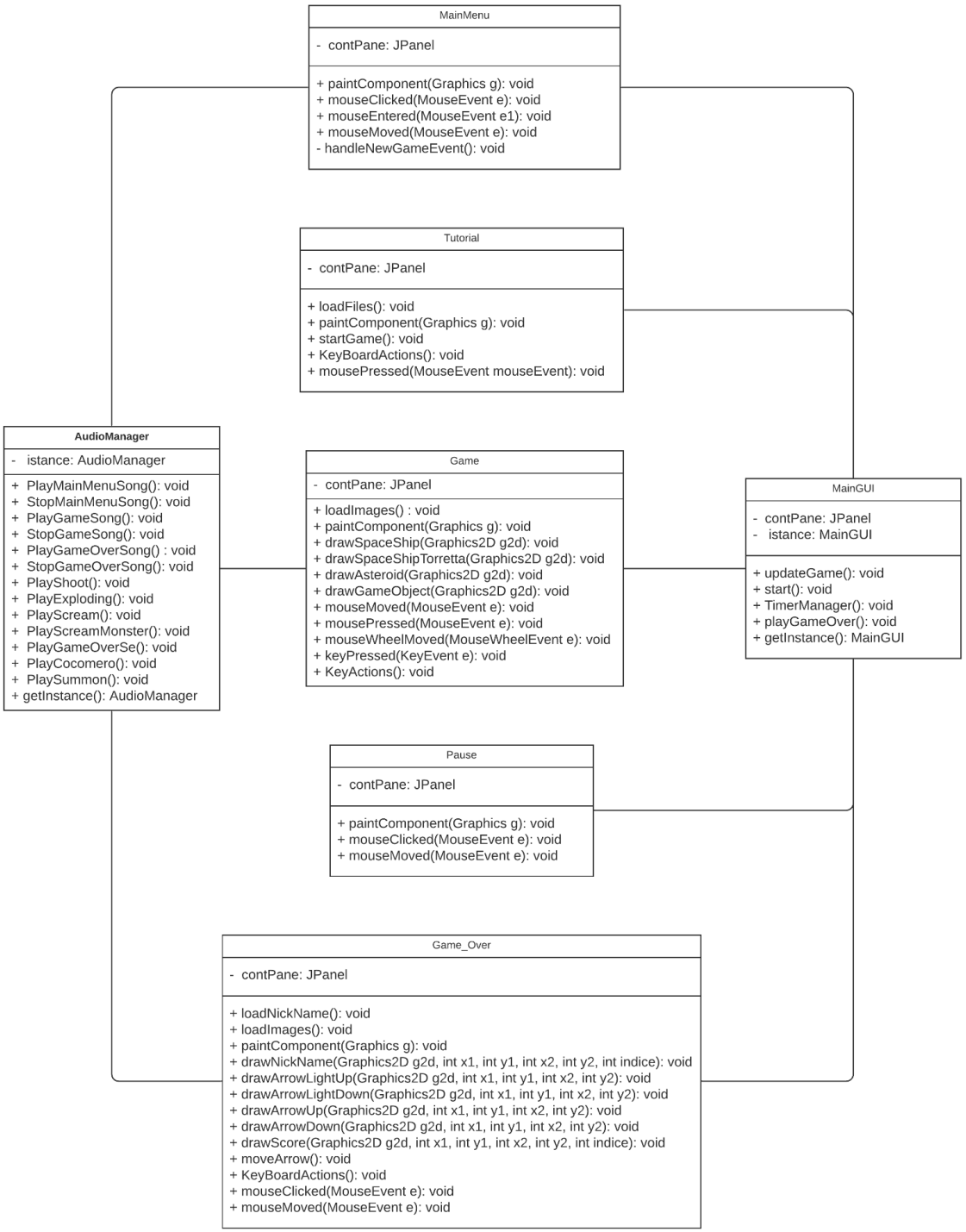
- **savePunteggio** Salva il punteggio corrente in un file di testo.

- **readChamp** Legge il punteggio del campione in carica, precedentemente salvato su un file di testo e aggiorna l’indice della lista stanziata nel pannello “GameOver” che contiene le immagini dei caratteri e dei numeri.

Gli ultimi due metodi presentati vengono richiamati nella classe “Game_Over” nel package “view”.

3.4 View

Le classi appartenenti al blocco View di Space Apocalypse si trovano nel package view. La loro struttura è rappresentata nel seguente diagramma UML:



3.4.1 La classe MainGui

La classe MainGui estende la classe JFrame, si occupa di creare la finestra dove viene visualizzata l'applicazione e di creare i pannelli al suo interno. Vengono settate due variabili "height" e "width" che rappresentano la lunghezza e la larghezza dello schermo dove verrà lanciata l'applicazione. Nel nostro caso abbiamo scelto una risoluzione fissa di 1280x720p. Viene creato un oggetto JPanel col nome di "contentPane" sul quale viene chiamato il metodo setLayout passando come parametro un oggetto di tipo CardLayout. A questo punto contentPane è un contenitore di oggetti JPanel e andrà a contenere tutti i pannelli dell'applicazione, ovvero: mainmenu, tutorial, game, pause e gameover. In seguito viene creato un oggetto GameLoop per avviare il thread dove viene eseguito il metodo **updateGame**. Viene creato anche un oggetto di tipo GameTimer, che è un timer che pilota lo spawn degli asteroidi nella mappa di gioco.
In particolare i metodi:

- **updateGame:** viene chiamato ad ogni ripetizione del ciclo while all'interno del metodo **run**, nella classe GameLoop del Model. Quando le condizioni descritte negli if sono verificate, esegue tutti i metodi della classe ControllerForView che hanno bisogno di essere richiamati ad ogni ciclo di aggiornamento dello stato dell'applicazione, ovvero il metodo che controlla le collisioni e quelli che regolano il movimento della navicella, degli asteroidi e degli oggetti della lista GameObject. Viene anche aggiornato il timer creato in precedenza chiamando sulla sua istanza il metodo **tick**.

- **TimerManager:** implementa il metodo **action** di GameTimer regolando i timer che gestiscono rispettivamente lo spawn degli asteroidi, lo spawn dei nemici e gli spari dei nemici.

3.4.2 Le classi MainMenu, Pause, Tutorial

Queste classi hanno la stessa struttura, estendono la classe JPanel e implementano l'interfaccia MouseInputListener. Sono presenti dei pulsanti la cui animazione è gestita all'interno del metodo **paintComponent** e regolata da una variabile booleana che viene modificata in base all'input dell'utente, infatti quando con il mouse l'utente entra nell'area della schermata dedicata al pulsante la variabile viene settata a True e viene visualizzata un'immagine che simula l'illuminazione del pulsante per dare un feedback visivo all'utente.

3.4.3 La classe Game

La classe Game estende JPanel e implementa i metodi MouseMotionListener, MouseListener, KeyListener, MouseWheelListener. Il compito della classe è raccogliere gli input dell'utente, inviarli al Controller e visualizzare gli oggetti che compongono il gioco. In particolare i metodi:

- **LoadImages** Inserisce tutti gli oggetti di tipo BufferedImage nella mappa imageMap.

- **drawSpaceShip** Disegna la navicella tramite la concatenazione di più trasformazioni affini: traslazione e rotazione. Le trasformazioni sono gestite tramite la creazione di un oggetto di tipo AffineTransformation. Le coordinate dove visualizzare la navicella vengono passate al metodo dal Controller. I fattori di scala dipendono dalla grandezza dello schermo dove viene lanciata l'applicazione. (Queste ultime considerazioni valgono anche per tutti i metodi con il prefisso "draw").

- **drawSpaceShipTorretta** Disegna la torretta della navicella, che cambia colore in base al proiettile che l'utente sceglie.

- **drawGameObject** Si occupa di disegnare tutti gli oggetti della lista gameobjects

- **mouseMoved** gestisce l'input derivante dal movimento del mouse. Vengono passate al metodo **ruotaSpaceShip** le coordinate del puntatore del mouse per permettere alla torretta della navicella di ruotare seguendo la punta del mouse.

3.4.4 La classe Game_Over

La classe estende JPanel e implementa il metodo MouseInputListener. Questa classe si occupa di visualizzare la schermata di game over e di raccogliere l'input dell'utente per permettergli di inserire il proprio nickname a fine partita. Vengono create due liste, "characterList" e "componentList" in cui in una vengono inserite le immagini delle lettere dell'alfabeto e le cifre, e nell'altra le immagini dei componenti del menu. L'inserimento del nickname avviene sequenzialmente in più fasi, in ognuna di queste è possibile, tramite mouse o tastiera, far scorrere una lista di caratteri e selezionare un numero o una lettera. Nella lista "indexmax" contenuta nel model sono memorizzati una variabile chiamata "avanzamento" che indica in che fase dell'inserimento ci si trova (le fasi sono tre, una per ogni carattere da inserire) e un indice numerico "index1" che viene usato per estrarre i caratteri dalla lista

“characterList”. All’interno del metodo **moveArrow** l’indice “index1” viene memorizzato in delle variabili d’appoggio, in base all’avanzamento corrente. Quando un carattere viene inserito l’avanzamento viene incrementato di uno e “index1” azzerato.

3.5 Problemi riscontrati

Space Apocalypse presenta una sola finestra con diversi pannelli al suo interno. Il primo problema riscontrato è stato come gestire il focus su un determinato pannello quando l’utente preme un tasto della tastiera. Per questo motivo abbiamo scelto di usare la tecnica del Key Bindings seguendo l’apposito tutorial java: https://docs.oracle.com/javase/tutorial/uiswing/misc/keybinding.html
Questa tecnica si affida alle classi InputMap, che associa ad un tasto della tastiera un nome, e ActionMap, che a quel nome fa corrispondere una determinata azione sovrascrivendo il metodo ActionPerformed. In questo modo è possibile associare ad un tasto della tastiera un’azione diversa in base al pannello che in quel momento ha il focus.

Un secondo problema si è presentato nel gestire le animazioni degli oggetti, specialmente nella rotazione della torretta della navicella. Per far fronte a ciò abbiamo usato la classe AffineTransform, che permette di realizzare sia traslazioni che rotazioni.

Per quanto riguarda la realizzazione dei blocchi il problema consisteva nel raggruppare un insieme di asteroidi mantenendo separate le istanze dei singoli asteroidi. Si è scelto quindi di creare una lista che contiene tutti i blocchi che a loro volta sono delle liste i cui oggetti sono le istanze degli asteroidi. In questo modo quando un proiettile colpisce un asteroide è possibile sapere in che blocco si trova e eliminare tutti gli altri asteroidi del suo stesso colore all’interno del blocco.

Ci eravamo inizialmente prefissati di riuscire a realizzare una schermata che si adattasse a qualsiasi risoluzione dove venisse lanciata l’applicazione. Siamo riusciti a far visualizzare la finestra principale a schermo intero ma questo causava dei problemi nella visualizzazione delle sprites, in quanto la loro risoluzione non può essere adattata dinamicamente alla risoluzione del dispositivo di output. Una possibile soluzione sarebbe stata quella di usare una tecnica di seam carving per ridimensionare le immagini al lancio dell’applicazione. Per motivi di tempistica di

consegna del progetto non abbiamo potuto tentare questa strada e abbiamo impostato una risoluzione fissa della schermata di gioco.