## RELAZIONE PROGETTO NON-GRAVITAR

## Il seguente progetto redatto da Luca Genova e Matteo Lorenzoni è una versione del gioco “Gravitar”, senza gravità in C++.

Il tutto è realizzato su di una matrice [101][21], che viene aggiornata continuamente.

Dopo l'avvio del gioco, è possibile procedere all'uso dopo aver premuto il tasto invio.

Il giocatore ha a disposizione una navicella con cui può muoversi *su*, *giù*, a *destra* e a *sinistra*, *sparare* e attivare il *raggio traente*.

Inizialmente la navicella si trova nel primo Sistema Solare(dopo aver distrutto tutti i pianeti, scorrendo verso il bordo destro, è possibile accedere ad un nuovo sistema solare) dove vi sono diversi pianeti generati random. Lo scopo è quello di distruggere tutti i pianeti, per fare ciò il giocatore deve dirigersi con la navicella all’interno di un pianeta. Il pianeta al suo interno ha una superficie generata anch’essa random, dei bunker (che sono di due tipi, uno che spara in due direzioni ed un altro che spara in tre direzioni) e le ricariche carburante (che sono anch’esse di due tipi, una con ricarica minora l’altra con una ricarica maggiore) che vengono prese con il raggio traente.

Per uscire da un pianeta e tornare nel sistema solare basta salire fino alla cornice superiore e si tornerà nel sistema, per distruggerlo bisogna abbattere tutti i Bunker presenti al suo interno, sparando con la navicella, ogni qualvolta venga distrutto un Bunker lo score (il punteggio) aumenta di 3, e ovviamente i Bunker cercheranno di distruggere la navicella con i loro spari, ogni qualvolta la navicella venga colpita da uno sparo i punti vita della navicella vengono decrementati di uno.

Dopo aver distrutto tutti i pianeti di un sistema solare è possibile passare ad uno nuovo.

Il gioco finisce quando si è esaurito il carburante o sono finiti i punti vita.

In un pianeta, se la nostra navicella dovesse toccare il suolo perderebbe una vita, anche venendo colpiti da un proiettile dei bunker. Il livello di vita e di carburante è possibile vederlo nella “tabella” a destra della schermata di gioco.

Classe **Spaceship**:

Questa classe rappresenta la navicella.

Nel campo protected ha quattro variabili int (p1, p2, p3 ,p4) che rappresentano le coordinate (x, y) della navicella (p1=x, p2=y per il Sistema Solare) (p3=x, p4=y all’interno del pianeta), ha poi una variabile per la vita, una per il carburante, un per lo score e ho un oggetto di tipo BulletsList (LP).

Ci sono vari metodi per diminuire e/o aumentare la vita, il carburante e lo score, due metodi per settare le nuove posizioni della navicella dopo lo spostamento, uno per il sistema solare, l’altro per il pianeta, un metodo per ritornare l’oggetto BulletsList(LP). Altri metodi:

* addBulletSPaceship() -> aggiunge i proiettili richiamando il metodo addBullets di LP
* deleteBulletSPaceship() -> rimuove i proiettili richiamando il metodo deleteBullets di LP
* deleteAllBulletsSpaceship() -> rimuove tutti i proiettili che rimangono quando la navicella esce dal pianeta
* returnParameter(int i) -> mi serve per farmi ritornare i vari parametri della navicella (coordinate, fuel, life ecc..) ed usarli nelle altre classi
* modificaCoordinateInCasoDiNuovoSistemaSolare(int i) -> nel caso in cui passo ad un nuovo sistema solare, questo metodo mi risetta la posizioni iniziale della navicella
* spaceshipIsAlive() -> ritorna un booleano, TRUE se il carburante e la vita sono >=1, FALSE altrimenti

Classe **BulletsList**:

Al suo interno abbiamo creato una struttura di tipo lista, in modo tale da poterla utilizzare sempre anche in altre classi basta includere BulletsList.

Nel campo protected ha una lista di proiettili, che ci servirà per lavorarci sopra, con add e delete.

E a parte il costruttore ho tre metodi:

* addBullets() - > aggiungo proiettili.
* deleteBullets() -> rimuovo proiettili.
* returnList() -> mi serve ritornare la lista quando ci lavoro in altri classi.

Classi **Bunker1**/**Bunker2**:

Al loro interno abbiamo un struttura di tipo lista, ci serve per avere una lista di Bunker.

Le due classi sono quasi uguali, l’unica cosa che le differisce è che una (Bunker1) ha due oggetti di tipo BulletsList, mentre l’altra (Bunker2) ne ha tre.

Nel campo protected abbiamo due variabili int (xBunker1, yBunker1) per le coordinate, una sempre int (lifeBunker1) per la vita e due/tre oggetti di tipo BulletsList per i proiettili.

Ho un metodo che mi serve per ritornare le coordinate, uno che ritorna i punti vita, uno che decrementa di uno la vita del bunker e viene richiamato quando un proiettile della navicella colpisce il Bunker. Altri metodi:

* addBulletSBunker() -> aggiunge i proiettili alle due/tre liste di proiettili richiamando il metodo addBullets per ogni lista
* deleteBulletSBunker() -> elimina i proiettili dalle due/tre liste di proiettili richiamando il metodo deleteBullets per ogni lista
* deleteAllBulletsBunker() -> metodo che rimuove tutti i proiettili che rimangono quando i Bunker vengono eliminati
* returnLP(int n) -> in base all’intero che gli passo mi ritorna le varie liste di proiettili

Classe **SolarSystem**:

Qui ho due strutture di tipo lista. Una è una lista di pianeti, l’altra è una lista di sistemi solari, che al suo interno contiene la lista di pianeti. Inizialmente setto le liste a NULL, poi genero un numero random, esso sarà il numero di pianeti contenuti nel Sistema Solare, a quel punto vengono settati i vari pianeti. Come unica variabile protected ho un puntatore ad una lista di sistemi solari