## **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* sia all’interno del sistema solare che all’interno del pianeta e *sparare* e attivare il *raggio traente* solo all’interno del pianeta. I comandi quindi cambiano in base a dove ci si trova, e sono indicati nel lato destro della schermata.

Inizialmente la navicella si trova all’interno del primo Sistema Solare dove vi sono diversi pianeti generati random (sia il numero che la loro posizione), rappresentati con il carattere “O” verde.

Lo scopo è quello di distruggere tutti i pianeti e ottenere il punteggio più alto possibile (in quanto il gioco è infinito), per fare ciò il giocatore deve dirigersi con la navicella all’interno di un pianeta e distruggere i Bunker al suo interno.

Dopo aver distrutto tutti i pianeti è possibile accedere ad un nuovo sistema solare: scorrendo verso il bordo destro è possibile accedere al sistema solare successivo (che deve essere completato), mentre scorrendo verso il bordo sinistro si può entrare nel sistema solare precedente già completato (a meno che non si è all’interno del primo sistema solare), utile per recuperare eventuale carburante lasciato indietro.

I parametri della navicella e più in generale del gioco sono:

* la vita
* il carburante
* il punteggio

Il gioco termina quando sono finite le vite oppure l’ammontare del carburante è pari a zero.

Si perdono vite:

* venendo colpiti dai proiettili dei vari bunker
* interagendo (andando a sbattere) con i caratteri che compongono la superfice del pianeta.

Carburante:

* Dove si perde: continuamente all’interno del pianeta, ad ogni ciclo viene decrementato di un’unità, non dipende dal movimento della navicella ma è indipendente da tutto.
* Dove non si perde: all’interno del sistema solare, si può al suo interno girare tranquillamente alla ricerca del pianeta dove entrare senza la preoccupazione di perdere carburante

Il pianeta al suo interno ha una superficie generata anch’essa random con vari dislivelli (è generata random ma comunque all’interno di certi parametri per ottenere unna superfice adatta al gioco, quindi è controllato il random)

Elementi sulla superfice:

* Bunker1 (carattere “b”): spara in due direzioni
* Bunker2 (carattere “B”): spara in tre direzioni
* Ricariche carburante: sono di due tipi, una rappresentata dal “c” che incrementa il carburante di 100 unità e una rappresentata dal carattere “C” che incrementa di 500 unità.

Per uscire da un pianeta e tornare nel sistema solare basta salire fino alla cornice superiore e si tornerà nel sistema solare nella posizione in cui si era prima di entrare nel pianeta, per distruggerlo bisogna abbattere tutti i Bunker presenti al suo interno, sparando con la navicella. Una volta uscito il pianeta verrà rappresento con una “X” blu.

Ogni qualvolta venga distrutto un Bunker lo score (il punteggio) aumenta di 3 o di 5 in base al tipo di Bunker.

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

Classi:

* SolarSystem
* PlanetSurface
* Spaceship
* Bunker1
* Bunker2
* BulletsList
* Mapp

Classe **SolarSystem**:

All’interno di questa classe ci sono due tipi di strutture a rappresentare delle liste: Planet e SS

Struttura Planet

Campi:

* int xPlanet: coordinata x del pianeta
* int yPlanet: coordinata y del pianeta
* ptr\_PlanetSurface planetSurface: puntatore all’oggetto PlanetSurface, dal quale poi si entra all’interno del pianeta
* ptr\_listaBunker1 listBunker1: puntatore alla lista di Bunker1, serve per indicare al gioco che una volta che il puntatore punta a NULL (insieme a quello listBunker2), allora il pianeta è stato distrutto
* ptr\_listaBunker2 listBunker2: puntatore alla lista di Bunker2, serve per indicare al gioco che una volta che il puntatore punta a NULL (insieme a quello listBunker1), allora il pianeta è stato distrutto
* Planet\* next: puntatore al pianeta successivo
* Planet\* prev: puntatore al pianeta precedente

Struttura SS (SS indica sistema solare)

Campi:

* int numberSolarSystem: indica il numero del sistema solare (si parte con uno solo, poi a mano a mano che si distruggono i sistemi solari ne verranno aggiunti dei nuovi)
* ptr\_Planet puntatore\_planet: puntatore alla lista di pianeti
* bool completed: restituisce se il sistema solare in considerazione è stato distrutto o meno
* SS\* next: puntatore al sistema solare successivo
* SS\* prev: puntatore al sistema solare precedente

Descritte queste due strutture arriviamo alla classe principale, ovvero “SolarSystem”

Come campo ha solamente un elemento che è un puntatore alla lista SS (lista di sistemi solari). Inizialmente ci sarà soltanto un sistema solare, poi con il proseguo del gioco ne verranno aggiunti dei successivi

Campi classe SolarSystem:

* ptr\_SS SSystem: puntatore di unna lista di sistemi solari

Metodi classe SolarSystem:

* SolarSystem(Spaceship p): costruttore della classe. Genera un numero di pianeti random (da 4 a 6), poi crea una nuova sistema solare, gli assegna il numero 1 (essendo il primo), crea una lista di pianeti usando il metodo creaListaPianeti() con un numero di pianeti pari al numero precedentemente generato, setta il campo completed a false in quanto non ancora distrutto e setta a NULL i due puntatori ai sistemi solari adiacenti in quanto al momento è l’unico sistema solare esistente
* void addSolarSystem(Spaceship p): metodo per aggiungere un sistema solare in coda, verrà usato una volta completato il sistema solare precedente
* ptr\_Planet creaListaPianeti(Spaceship p, ptr\_Planet head, int numberPlanets): metodo per creare la lista di pianeti, gli assegna le coordinate e le accoda nella lista
* bool ifPianetiSovrapposti(): controlla se a due pianeti sono state assegnate le stesse posizioni
* void setMappUniverso(Spaceship p): setta la matrice del sistema solare, quindi la cornice, le informazioni sulla destra, la navicella e i vari pianeti in base alla loro posizione. Si occupa anche di aggiungere un nuovo sistema solare se quello attuale è stato completato
* void spostamentoUniverso(char& moveSpaceshipUniverso): assegna un valore ad un carattere, con il quale poi si indica alla navicella quale interazione deve avere con il gioco (cosa dovrà fare)
* char interationSpaceship(Spaceship& p, bool& b, int& numeroSS): metodo per fare interagire la navicella con il gioco all'interno dello scenario "sistema solare" (spiega le varie interazioni in cosa consistono). Permette di modificare i parametri della navicella, oppure di cambiare sistema solare, oppure ancora permette di entrare all’interno di un pianeta
* ptr\_PlanetSurface pianetaCorrente(Spaceship p, char n): metodo che restituisce il puntatore all’oggetto PlanetSurface in cui si è cercato di entrare
* void solarSystemDestroyed(): controlla se per tutti i pianeti presenti nella lista di pianeti di quel sistema solare hanno entrambe le liste di bunker che puntano a NULL, in quel caso vuol dire che il sistema solare in questione è completato
* void solarSystemChange(int i): metodo che grazie a dei confronti sposta il puntatore al sistema solare corrente in avanti o indietro, di fatto serve per cambiare il sistema solare
* bool returnIfDestroyed(): restituisce se il sistema solare è stato distrutto
* void aggiornamentoListaBunker (Spaceship p, char n, ptr\_listaBunker1 head1, ptr\_listaBunker2 head2): metodo per aggiornare le liste di bunker della struttura Planet. Questo perché durante il gioco esse vengono modificate all’interno del pianeta e quindi nella classe PlanetSurface, però poi servono anche alla classe SolarSystem in quanto devono indicare su un pianeta è stato distrutto o meno

Classe **PlantSurface**:

All’interno di questa classe avviene il “gioco d’azione”, oltre che a muoversi qui si può anche sparare e attivare il raggio traente.

Qui inoltre si possono perdere vite e finire il carburante, quindi è all’interno di questa classe che si rischia di fare finire il gioco.

Questione Bunker: sono parametri della classe SolarSystem, quindi di una classe “superiore” se cosi si può dire; queste liste pero vengono create all’interno di questa classe in quanto servono le coordinate “x” e “y” per poter posizionare i vari bunker sopra la superfice di gioco, ma appunto questa superfice viene generata all’interno del costruttore PlanetSurface e quindi occorre creare anche le due liste di bunker all’interno del costruttore PlanetSurface (in questo modo viene generata prima la superfice e poi vengono passate le posizioni dei vari caratteri di quest’ultima cosi che i vari bunker possano essere posizionati un carattere sopra all’interno della matrice)

Campi:

* ptr\_listaBunker1 LBunker1: puntatore alla lista di Bunker1 del pianeta
* ptr\_listaBunker2 LBunker2: puntatore alla lista di Bunker1 del pianeta

Metodi:

* PlanetSurface(Spaceship spaceship, ptr\_listaBunker1& head, ptr\_listaBunker2& head2): costruttore della classe PlanetSurface. Ha il compito di settare le parti generali della schermata come cornice, comandi, informazioni della navicella ecc.. ha poi il compito di generare la superfice in modo random e di settare i vari caratteri per rappresentarla nella schermata, infine ha il compito di creare in modo random le due liste di Bunker1 e Bunker2 e di settare anch’essi sulla schermata insieme ai caratteri del carburante (anche quest’ultimi piazzati in modo random)
* ptr\_listaBunker1 creaBunkerList1(ptr\_listaBunker1 head, int xB, int yB): metodo per creare la lista di tipo Bunker2, inserimento in coda
* ptr\_listaBunker2 creaBunkerList2(ptr\_listaBunker2 head, int xB, int yB): metodo per creare la lista di tipo Bunker2, inserimento in coda
* void spostamentoPlaetSurface(char& moveSpaceshipUniverso): assegna un valore ad un carattere, con il quale poi si indica alla navicella cosa deve fare
* char interationSpaceshipPlanetSurface(Spaceship& p, bool& pianetaDistrutto): metodo per fare interagire la navicella con il gioco all'interno dello scenario "pianeta" (spiega le varie interazioni in cosa consistono). Permette di modificare i parametri della navicella, oppure di attivare/ritirare il raggio traente, oppure di aggiungere un Bullet alla lista di bullet della navicella, e quindi di sparare. Inoltre aggiorna la posizione dei proiettili della navicella e modifica se necessario le liste dei bunker
* void setMappPlanetSurface(Spaceship& p, char n, bool& b): in base all’interazione che deve svolgere, modifica graficamente la schermata, settando quindi i vari caratteri. Inoltre ha il compito importante di controllare se si vuole uscire dal pianeta, in tal caso modifica un booleano passato in input (che viene usato per sapere se si è dentro ad un pianeta o meno e quindi quale scenario bisogna stampare)
* void raggioTraenteUscente(Spaceship& p): metodo per rappresentare graficamente il raggio e per aggiornare il parametro “carburate” della navicella
* void raggioTraenteEntrante(Spaceship p): metodo per cancellare graficamente il raggio
* void sparaBunker(Spaceship& p, int& i): metodo per aggiornare automaticamente gli spari dei bunker, all’interno vi è un ciclo numerico che ogni volta che arriva a 12 si resetta e serve per dettare i tempi degli spari
* void refreshBunker1(Spaceship& p, ptr\_listaBunker1& head1, int i): metodo per aggiungere i proiettili alle due liste di proiettili di ciascun bunker all’interno della listaBunker1; ha il compito di aggiungere i proiettili, di modificarne le coordinate, di settare la matrice correttamente seguendo l’andamento del proiettile; ha anche il compito di eliminare i proiettili quando escono dalla schermata di gioco
* void refreshBunker2(Spaceship& p, ptr\_listaBunker2& head2, int i): stesso ragionamento per il metodo refreshBunker1()
* void refreshSpaceship(Spaceship& p, ptr\_listaBunker1& head1, ptr\_listaBunker2& head2): metodo per aggiungere i proiettili alla lista di proiettili della navicella; ha il compito di aggiungere i proiettili, di modificarne le coordinate, di settare la matrice correttamente seguendo l’andamento del proiettile; ha anche il compito di eliminare i proiettili quando escono dalla schermata di gioco. Ha anche il compito di modificare le liste di bunker nel caso in cui dei bunker al loro interno vengono distrutti (quando un proiettile della navicella incontra un bunker)
* bool assenzaBunker(): restituisce se sono presenti o meno dei caratteri che rappresentano dei bunker
* ptr\_listaBunker1 modificaLista1(Spaceship& p, ptr\_listaBunker1& head, int x, int y): metodo per modificare la lista di Bunker1, se presente un bunker con le vite azzerate allora lo elimina dalla lista ed incrementa il punteggio del gioco
* ptr\_listaBunker2 modificaLista2(Spaceship& p, ptr\_listaBunker2& head, int x, int y): stesso ragionamento di modificaLista1, ma con i Bunker2
* void eliminareSpariBunker1Distrutto(ptr\_listaBunker1& head): metodo con il compito di azzerare le liste di proiettili e cancellarli graficamente
* void eliminareSpariBunker2Distrutto(ptr\_listaBunker2& head): stesso ragionamento di modificaLista1, ma con i Bunker2
* void azzerareListeBullets(Spaceship& p, bool& b): metodo per cancellare tutti i proiettili presenti in gioco, sia quelli della navicella che quelli di tutti i bunker (sia graficamente sia per azzerare tutte le varie liste di proiettili), viene usato quando si esce dal pianeta per pulire la schermata
* ptr\_listaBunker1 ritornoBunkerList1(): restituisce la lista di Bunker1;
* ptr\_listaBunker2 ritornoBunkerList2(): restituisce la lista di Bunker2;

Classe **Spaceship**:

Questa classe rappresenta la navicella.

Campi:

* P1 e P2 sono le coordinate “x” e “y” rispettivamente della navicella all’interno del sistema solare
* P3 e P4 sono le coordinate “x” e “y” rispettivamente della navicella all’interno del pianeta.
* Life: indica la vita della navicella
* Fuel: indica il carburante della navicella
* Score: indica il punteggio della navicella
* LP (Lista Proiettili): è il puntatore della classe “BulletList” e serve per far sparare la navicella

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).

Metodi:

* Spaceship(): costruttore, setta i parametri iniziali della navicella
* void moveSpaceshipSolarSystem(char n): gli viene passato un carattere, in base al suo valore rappresenta una direzione della navicella. Esegue prima un controllo per far si di non uscire dalla cornice e se verificato setta le nuove coordinate P1 e P2 (attenzione: non modifica la posizione della navicella graficamente, ma solo le sue coordinate)
* void moveSpaceshipPlanet(char n): ha lo stesso scopo della funzione precedente, con la differenza che modifica le coordinate P3 e P4 e non c’è il controllo nel bordo superiore in quanto all’interno del pianeta è consentito superarlo per potervi uscire
* int returnParameter(int i): in base al numero in input, restituisce il valore di un campo
* void fuelMinus(): diminuisce il carburante di una unità
* void lifeMinus(): diminuisce la vita di una unità
* void setScore(int i): aumenta il punteggio di 3 o di 5 in base al numero in input
* void setInitialPosition(): metodo usato per impostare sempre la stessa posizione iniziale della navicella all’interno di ciascun pianeta (P3=3, P4=3), a differenza delle coordinate del sistema solare P1 e P2 che rimangono le stesse quando si entra in un pianeta
* void fuelPlus(int i): aumenta il carburante di 100 o di 500 in base al numero in input
* bool spaceshipIsAlive(): controlla se la vita e il carburante sono maggiori di uno, e quindi controlla se la navicella è ancora in vita per proseguire il gioco
* void modificaCoordinateInCasoDiNuovoSistemaSolare(int i): metodo che serve per modificare le coordinate del sistema solare P1 e P2 in modo tale da avere la navicella nella stessa riga e nella posizione corretta una volta che si cambia sistema solare verso destra o verso sinistra
* void addBulletSPaceship(): aggiunge un Bullet alla lista di bullet (BulletList)
* void deleteBulletSPaceship(): elimina un Bullet alla lista di bullet (BulletList)
* void deleteAllBulletsSpaceship(): elimina tutti i Bullet presenti nella lista di bullet della navicella, viene nel caso in cui la lista non sia vuota al momento in cui la navicella esce dal pianeta
* ptr\_BulletsList returnLP(): ritorna la lista di bullet

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

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

All’interno di questa struttura abbiamo un puntatore all’oggetto Bunker e un puntatore alla struttura successiva (per formare una lista di Bunker)

Le due classi sono quasi uguali, l’unica cosa che le differisce è che una (Bunker1) ha due puntatori di tipo “ptr\_BulletList” che le consentono di avere due liste di Bullet (quindi due file di proiettili separate), mentre l’altra (Bunker2) ne ha tre.

Campi della classe Bunker1/Bunker2:

* int xBunker1: coordinata x del bunker
* int yBunker1: coordinata y del bunker
* int lifeBunker1: numero di vite del bunker (3)
* ptr\_BulletsList LPsx: puntatore alla lista sinistra di proiettili
* ptr\_BulletsList LPdx: puntatore alla lista destra di proiettili
* ptr\_BulletsList LPcentrale: puntatore alla lista centrale di proiettili (solo in Bunker2)

Metodi:

Usati quelli della classe Bunker1, ma come gia detto sono equivalenti alla classe Bunker2

* Bunker1(int x, int y, int life): inizializza la posizione del bunker e la sua vita, inoltre crea un’oggetto di tipo BulletsList da cui poi si potranno aggiunger i Bullet e permettere al bunker di sparare
* int coordinateBunker1(bool b): restituisce la poisizione del bunker
* void minusLife(): toglie una vita al Bunker1 (nel caso venga colpito dal un "Bullet" della spaceship)
* int returnLife(): restituisce la vita del bunker
* void addBulletSBunker1(): metodo per aggiungere dei “Bullet” a ciascuna delle liste di bullet che fanno parte del Bunker (2 per Bunker1, 3 per Bunker2), verrà usato in automatico per garantire uno sparo continuo a tutti i bunker
* void deleteBulletSBunker1(): metodo per aggiungere dei “Bullet” a ciascuna delle liste di bullet che fanno parte del Bunker (2 per Bunker1, 3 per Bunker2), verrà usato nel momento in cui il Bunker viene distrutto cosi da eliminare anche i suoi spari dal gioco
* void deleteAllBulletsBunker1(): metodo utilizzato per eliminare tutti i “Bullet” presenti nelle liste, verrà usato quando si esce dal pianeta cosi che quando ci si rientra non si hanno già dei proiettili sparsi nel “cielo”
* ptr\_BulletsList returnLP(int n): restituisce un puntatore delle liste di proiettili in base al numero in input

Classe **BulletsList**:

Questa classe rappresenta i proiettili e la lista di proiettili sia della navicella che dei bunke.

Come campo ha un puntatore di tipo Bullet che è una struttura rappresentate il proiettile.

Struttura Bullet:

* int xBullet: coordinata x del proiettile
* int yBullet: coordinata y del proiettile
* bool eliminato: campo usato per indicare poi al metodo deleteBullets() se cancellare o meno il proiettile
* ptr\_Bullet next: indica il puntatore al proiettile successivo della lista di proiettili

Campi protected dell’oggetto BulletList:

* ptr\_Bullet listaProiettili: puntatore alla lista di proiettili

Metodi:

* BulletsList(): costruttore dell’oggetto BulletList, semplicemente imposta la lista di Bullet a NULL in quando inizialmente tutte le liste di proiettili sono vuote (poi quelle dei bunker verranno riempite automaticamente mentre quella della navicella premendo la barra spaziatrice)
* void addBullets(int x, int y, bool b): aggiunge i Bullet (proiettili) alla lista di bullet, gli aggiunge in coda (questo importante in quanto poi permette di strutturare il metodo di eliminazione in un determinato modo), gli passa le coordinate del proiettile e il valore falso al campo “eliminato” in quanto al momento non sono da eliminare
* void deleteBullets(): elimina i Bullet (proiettili) dalla lista di Bullet quando hanno il valore true nel campo “eliminato”, siccome questo valore lo assumono solo quando raggiungono la superficie del pianeta o una dei tre bordi superiori della schermata, solo il primo proiettile in testa alla coda può essere eliminato, di conseguenza il metodo è stato strutturato in maniera semplificata
* ptr\_Bullet retrunList(): ritorna la lista di proiettili

Classe **Mapp**

Classe che contiene tutti i metodi per impostare in modo grafico i vari componenti, dalla cornice, le schermate iniziali e finali, i parametri di gioco, i comandi ecc..

Campi:

* int xMatrice = 100;
* int yMatrice = 20;
* char matrice[101][21];

Metodi:

* Mapp();
* void printMapp(int i = 0, int a = 0): metodo per stampare a video i vari caratteri; imposta anche le varie dimensioni dello schermo, i vari colori, lo Sleep
* void setMapp(): metodo per settare la cornice della mappa (schermata), che sarà poi fissa perché presente in tutte le varie situazioni del gioco
* void setGeneralParameters(int fuel, int life, int score): metodo usato per settare nella matrice generale le informazioni generali che sono sempre presenti (le informazioni fisse come i comandi, e le informazioni variabili che rappresentanto i vari parametri del gioco
* void setSolarSystemParameters(int fuel, int life, int score, int universo): metodo per settare il numero del sistema solare (informazione presente solo nella schermata del sistema solare)
* void setPlanetSurfaceParameters(int fuel, int life, int score): metodo per settare le informazioni relative ai comandi di gioco all'interno del pianeta (come sparare e attivare il raggio traente)
* void setInitialMapp(): metodo per impostare la schermata iniziale, la schermata con le informazioni di inizio gioco
* void setFinalMapp(): metodo per impostare la schermata di fine gioco
* void setContinueMapp(): metodo per impostare la schermata una volta che si è finita la vita, propone due scelte: quella di terminare il gioco oppure quella di ricominciarlo da capo
* void setPlanetSurfaceParameters(int fuel, int life, int score): metodo per settare le informazioni relative ai comandi di gioco all'interno del pianeta (come sparare e attivare il raggio traente)
* void ShowConsoleCursor(bool showFlag): metodo per non mostrare il cursore nella schermata