UNIVERSITÀ DI PISA

INGEGNERIA ROBOTICA E DELL'AUTOMAZIONE

Corso: Real Time Systems

SPACE BATTLE

Studente: Marco Borraccino

N° matricola: 501794

Email: m.borraccino@studenti.unipi.it

23 Giugno 2021

Introduzione

Il progetto consiste nello sviluppo di un gioco, in stile anni ’90, in cui due navicelle spaziali combattono tra loro. Come ogni gioco di questo tipo, un menu semplice permette di scegliere uno dei vari livelli di difficoltà disponibili. La navicella del giocatore è controllata da tastiera mentre quella nemica agisce in modo casuale ma seguendo determinati schemi. Al crescere del livello sarà sempre più difficile colpire l’avversario e vincere.

Descrizione

Il programma è diviso in due sezioni distinte: il *menu di gioco* e il gioco vero e proprio.

MENU DI GIOCO

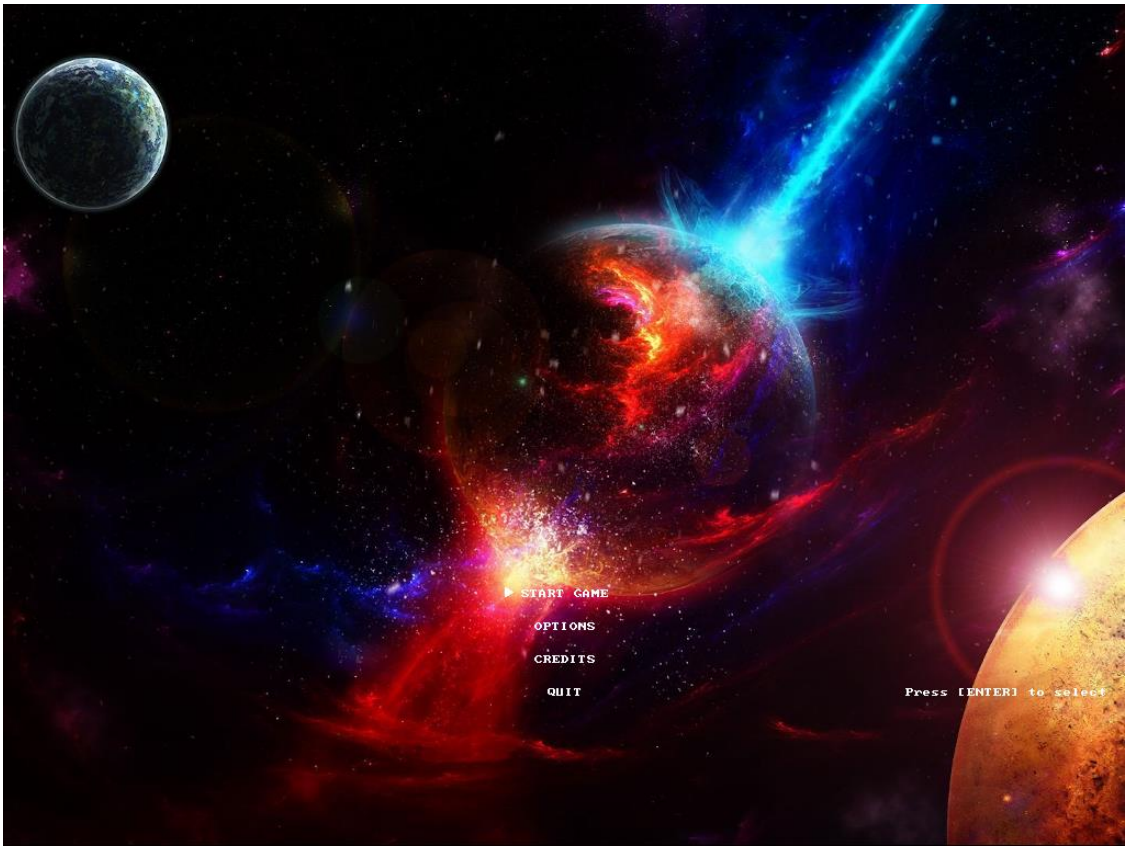

Figura 1: screenshot del menu principale

Nel menu di gioco è visualizzata una schermata (*figura 1*) invitando l’utente a scegliere tra quattro opzioni:

- **Start Game**: l’utente sceglie di iniziare una nuova partita e quindi in questa schermata si chiede di selezionare il livello di difficoltà della navicella nemica;
- **Options**: in questa schermata sono presentati i vari comandi a disposizione del giocatore durante il gioco;
- **Credits**: scegliendo questa opzione si possono leggere informazioni sugli sviluppatori del gioco;
- **Quit**: questa opzione semplicemente termina l’esecuzione del programma.

Questa parte è gestita in modo semplice tenendo conto della posizione del selettore e della scelta che compie l’utente premendo il tasto *[ENTER]*. A quel punto una delle varie schermate viene mostrata dalle quali si potrà solo tornare indietro al menu principale. Per cominciare una nuova partita bisogna selezionare *Start Game* e scegliere il livello di difficoltà.

GIOCO

Dopo il tipico countdown di tre secondi la partita inizia e sulla schermata si visualizzano le due navicelle nel campo di gioco, le barre della salute e dell'energia in alto, comandi supplementari e impostazioni in basso.

Premendo il tasto *[0]* si potranno inoltre visualizzare in tempo reale altre informazioni (*figura 2*) come la direzione di tiro di entrambe le navicelle (linee rosse e blu), il vettore velocità della navicella del giocatore (linea bianca) e la posizione che il nemico intende raggiungere (cerchio bianco), come mostrato in figura.

Figura 2: screenshot della schermata di gioco con informazioni aggiuntive

Lo scopo del gioco è colpire l'altra navicella e portare a zero la sua *salute*, cercando di farlo per primi. È consentito quindi muoversi liberamente nello spazio per evitare di essere colpiti e contemporaneamente sparare e cercare di intercettare l'altro. Ogni navicella pertanto inizierà il match con la propria salute al massimo (100) e i danni subiti non potranno essere più recuperati. I colpi disponibili invece sono illimitati ma ognuno di questi consuma l'*energia* della navicella, che nel tempo si ricarica costantemente. È possibile quindi sparare raffiche solo se si ha abbastanza energia disponibile, altrimenti la cadenza di tiro è più contenuta.

La navicella del giocatore ha dei propulsori lungo il proprio piano longitudinale che permettono quindi di acquisire velocità solo in quella direzione, e altri laterali che consentono di variare il proprio angolo di imbardata, ruotando la navicella in senso orario o antiorario. Per questioni di manovrabilità, si è scelto di simulare la presenza di attrito viscoso tramite il parametro *friction*, che nel tempo agisce sulla velocità della navicella e ne diminuisce la sua intensità. Nello specifico propulsori e attrito agiscono solo sulla velocità, la posizione effettiva nel campo di gioco poi è valutata periodicamente in base ad essa, secondo la formula:

$$position_{(x,y)}^{k+1} = position_{(x,y)}^{k} + velocity_{(x,y)}^{k+1} \cdot \frac{GAME\ TASK_{period}}{100}$$

I proiettili sono espulsi dalla punta della navicella e la loro velocità risente anche della velocità di trasporto della navicella stessa:

$$vel_{bullet} = VELMAX_{bullet} + vel_{spacechip}$$

L'angolo di tiro, tuttavia, non è esattamente uguale a quello di orientamento della navicella ma è addizionato ad un rumore uniformemente distribuito, a media nulla:

$$angolo\ di\ tiro = angle_{spaceship} + w$$

$w \in \{-0.5 \div 0.5\}\ rad$, in base al valore di *PLY_FIRE_ANGLE_ERR*

Differentemente la navicella nemica è libera di muoversi in ogni direzione e punta sempre verso di noi. Il suo comportamento è strettamente correlato al livello di difficoltà come mostrato in tabella.

Azioni della navicella nemica	Easy	Normal	Expert
Mira verso la posizione attuale della navicella del giocatore	•	•	
Cambia posizione periodicamente nello spazio di gioco	•	•	•
Cambia posizione quando è sotto il tiro della navicella del giocatore		•	•
Mira verso la posizione predetta della navicella del giocatore			•

La cadenza di tiro nemica è casuale e ad ogni colpo stima la posizione di quella dell'utente con un certo errore in base al parametro RADAR_ERROR. Una volta determinata questa, se il livello è *Easy* o *Normal*, viene sparato immediatamente un colpo. Se invece il livello è *Expert*, viene assunta nota la velocità della navicella del giocatore e stimata la posizione futura. Nel dettaglio si calcola il tempo di volo del proiettile:

$$t = \frac{spaceship\ distance}{bullet\ velocity} = \frac{\sqrt{(player_x - enemy_x)^2 + \left(player_y - enemy_y\right)^2}}{BUL2\_VEL}$$

e con questo dove sarà la navicella del giocatore nel futuro:

$$position_{future} = position_{current} + velocity_{spaceship} \cdot t$$

Si calcola quindi il nuovo angolo di tiro come l'angolo della retta che passa per la posizione futura appena calcolata e quella della navicella nemica. Questo metodo si basa su un'approssimazione geometrica tanto valida quanto la velocità del proiettile è maggiore di quella delle navicelle. Ciononostante, l'errore ottenuto è dello stesso peso di quello che si propaga dal rumore aggiunto inizialmente.

Per quanto riguarda invece la posizione e il movimento della navicella nemica nello spazio di gioco, periodicamente è generata in modo casuale una *posizione desiderata* nello spazio e tramite un controllo proporzionale la navicella nemica la raggiunge. Questo succede ad ogni livello e una nuova posizione viene generata anche quando la navicella nemica è sotto il tiro di quella del giocatore, se la difficoltà è *Normal* o *Expert*.

La sezione in basso nella schermata di gioco invece suggerisce che si può cambiare il livello di difficoltà in qualsiasi momento, si può mettere in pausa il gioco e si può variare il parametro *friction*.

Game Over e Statistiche

Una volta che una delle due navicelle ha salute pari a zero il gioco finisce e appare la schermata di *Game Over* (*figura 3*). Qui viene annunciato il vincitore e sono riportate alcune statistiche: la salute di entrambi i giocatori a fine partita, i colpi sparati e quelli andati a segno, il livello di difficoltà. Da qui sarà possibile ritornare al menu principale e cominciare una nuova partita o chiudere il gioco.

Figura 3: screenshot della schermata di gameover con statistiche di gioco

Interfaccia

Per questo programma è necessario il solo utilizzo della tastiera. I comandi sono acquisiti in due modi differenti a seconda del tipo di funzione che hanno e del momento in cui sono premuti.

Nella parte di navigazione nel menu di gioco e nelle varie schermate, i tasti che permettono di spostarsi e selezionare le proprie scelte sono acquisiti sequenzialmente e solo quando vengono premuti e rilasciati. Durante il gioco invece si legge lo stato di tutti i tasti utili periodicamente per permettere di compiere più azioni nello stesso momento, come muoversi, ruotare e sparare.

I comandi disponibili durante il gioco sono mostrati nella schermata *Options* (*figura 4*) e alcuni sono suggeriti anche in basso nella schermata di gioco.

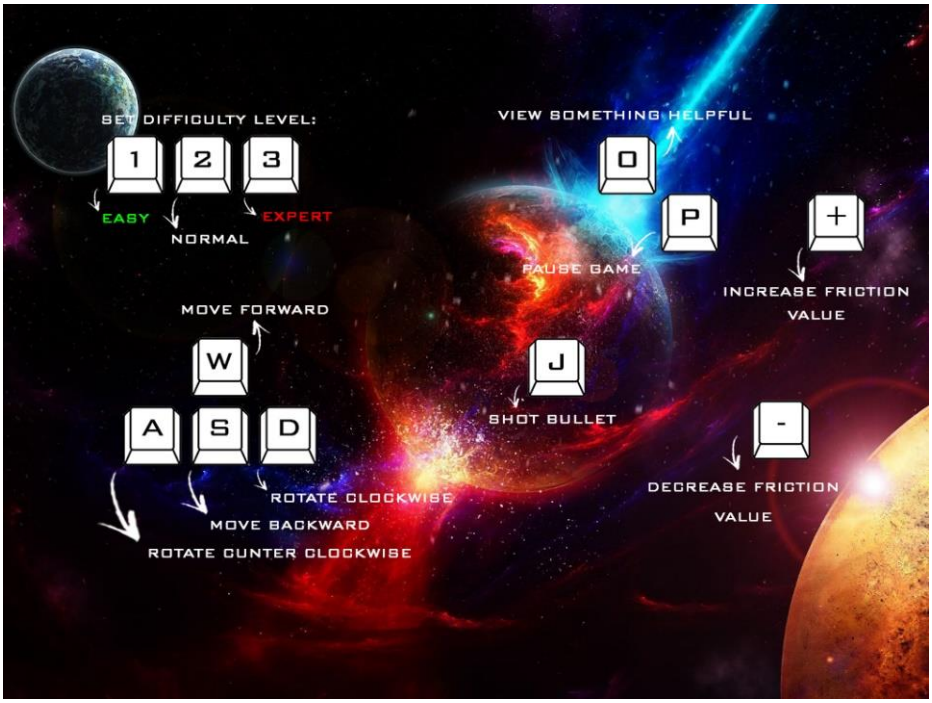

Figura 4: schermata di Options

Considerazioni sul codice

Il codice è stato sviluppato in linguaggio C, usando le librerie Allegro 4.4 [1] e pthread [2]. Il codice è diviso tra diversi file: `main.c` contiene la funzione principale e tutte le funzioni generali di ogni task, in `menu.c` troviamo tutte le funzioni per gestire il menu, in `game.c` ci sono le funzioni che gestiscono il flusso di gioco durante una partita e in `graphics.c` sono contenute le funzioni di disegno. Parametri e variabili relative ad ogni sezione sono nei rispettivi library file.

Principali strutture dati globali

Le principali strutture sono divisibili in due sezioni: quelle relative al menu e quelle del gioco effettivo.

Per navigare nel menu ci serviamo di:

- variabili booleane per indicare in quale schermata del menu ci troviamo, cioè: `menu_state`, `start_state`, `options_state`, `credits_state`, `game_state`, `pause_state`;
- una struttura `selector` per tenere traccia della posizione del selettore;

Durante il gioco invece le strutture globali principali utilizzate sono:

- due strutture `spaceship` utilizzate per ognuna delle navicelle in cui sono contenute tutte le informazioni quali salute, energia, velocità, posizione e angolo di orientamento;
- due vettori di strutture `bullet`, in cui ogni elemento corrisponde ad un proiettile nel campo di gioco, con le informazioni di posizione, velocità, danno, angolo e stato;

Inoltre, i vari task comunicano con il task di interpretazione dei comandi con delle variabili usate come flag, assegnando ad ogni azione il valore di un bit. Tutte queste strutture e le variabili relative ai comandi da tastiera sono protette da semafori *MUTEX*, per evitare problemi dati da accessi multipli.

Accessibile in ogni momento è la varabile `lvl` che contiene l'attuale livello di difficoltà.

Task

Diversi task periodici sono attivi durante l'esecuzione del programma e sono inizializzati solo se necessario. I primi resi attivi sono il task di interfaccia con la tastiera e il task che gestisce le schermate del menu. Durante il gioco invece sono attivi un task per aggiornare la grafica di gioco, un task per gestire e aggiornare lo stato di navicelle e proiettili e un task per simulare l'intelligenza artificiale del nemico. Questi tre sono attivi solo durante la partita e ogni volta che è terminata verranno distrutti.

Task	Periodo	Deadline	Priorità
COMM_TASK	30	30	20
MENU_TASK	30	30	22
GAME_TASK	30	30	22
AI_TASK	30	30	22
GRAPHICS_TASK	20	20	22

Nel dettaglio il task GAME_TASK che gestisce l'aggiornamento delle variabili globali di gioco, dopo aver inizializzato le strutture delle navicelle e dei proiettili, ha diverse funzioni, quali:

- Una per il controllo se ci sono nuovi comandi da parte dell'utente tramite tastiera per modificare la velocità della navicella o generare nuovi colpi;
- Una funzione per propagare nello spazio i proiettili e aggiornarne lo stato;
- Una funzione per calcolare la nuova posizione e lo stato delle navicelle.

Parallelamente il task di intelligenza artificiale prende decisioni in modo pseudocasuale sul generare o meno nuove posizioni desiderate e quindi muovere la navicella nemica, o sparare nuovi colpi. Il task di grafica infine accede alle varie strutture e visualizza in tempo reale tutti i parametri, i colpi e le due navicelle mentre il gioco evolve nel tempo.

I valori dei periodi e delle deadline riportati in tabella sono stati scelti a valle di diverse simulazioni: per l'acquisizione dei comandi un periodo di 30ms sono risultati essere più che sufficienti e il valore di aggiornamenti della grafica ogni 20ms rende il gioco molto fluido. Valori più bassi di quest'ultimo generano delle *deadline miss* piuttosto frequenti quando le entità da graficare sono numerose.

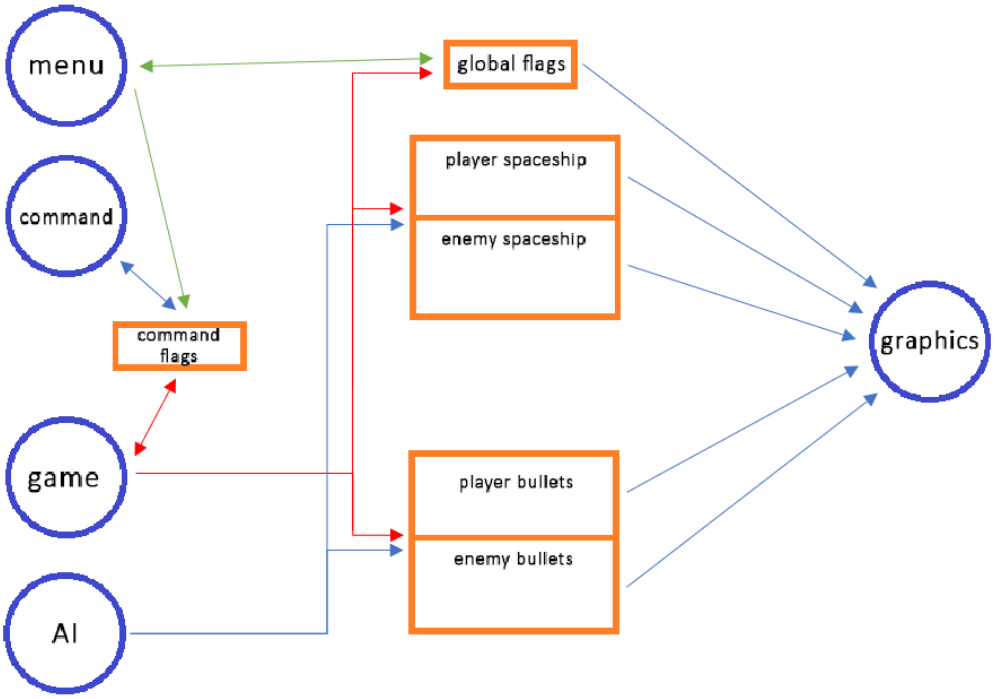

Figura 5: task-resource diagram

Parametri di gioco

Diversi parametri e variabili posso rendere diversa l'esperienza di gioco. I più importanti sono contenuti nel file game.h.

NAVICELLE

Parametro	Descrizione	Range	Valore suggerito
UNIT_ANGLE_ROTATION	Unità di rotazione angolare della navicella	$[0 \dots 2\pi]$	0.1
INC_VELOCITY	Incremento della velocità della navicella	$\mathbb{R}$	3
MAX_VELOCITY	Velocità massima	$\mathbb{R}$	30
START_HEALTH	Salute iniziale ad inizio partita	$[0 \dots 100]$	100
START_CHARGE	Energia iniziale ad inizio partita	$[0 \dots 100]$	100

PROIETTILI

Parametro	Descrizione	Range	Valore suggerito
BUL1_VEL	Velocità massima proiettili del giocatore	$\mathbb{R}$	$4 * \text{MAX\_VELOCITY}$
BUL1_CHARGE	Energia usata per ogni proiettile dal giocatore	$[0 \dots 100]$	5
PLY_RECHARGE_RATE	Velocità di ricarica dell'energia del giocatore	$[0 \dots 100]$	5
PLY_FIRE_ANGLE_ERR	Errore dell'angolo di tiro del giocatore	$[1 \dots 10]$	5
bullet1-> damage	Danno alla salute dei proiettili del giocatore	$\mathbb{N}$	***[1]**
BUL2_VEL	Velocità massima proiettili del nemico	$\mathbb{R}$	$3 * \text{MAX\_VELOCITY}$
BUL2_CHARGE	Energia usata per ogni proiettile dal nemico	$[0 \dots 100]$	5
ENY_RECHARGE_RATE	Velocità di ricarica dell'energia del nemico	$[0 \dots 100]$	5
SHOT_RATE	Cadenza di tiro della navicella nemica	$[0 \dots 1]$	0.3
bullet2-> damage	Danno alla salute dei proiettili nemici	$\mathbb{N}$	1
RADAR_ERROR	Errore in pixel dell'acquisizione della posizione del giocatore	$\mathbb{N}$	80
KP_POS_CONTROLLER	Costante proporzionale per il controllo della posizione	$\mathbb{R}$	0.3
NEW_POS_RATE	Velocità di aggiornamento di una nuova posizione desiderata per la navicella nemica	$\mathbb{N}$	90
NEW_POS_ALERT	Velocità di aggiornamento della posizione nemica quando è sotto tiro	$\mathbb{N}$	30

***[1]** Livello *Easy*: $damage = 4$; livello *Normal*: $damage = 2$; livello *Expert*: $damage = 1$.

GIOCO

Parametro	Descrizione	Range	Valore suggerito
lvl	Livello di difficoltà	$[1\ 2\ 3]$	-
MAX_FRICTION	Valore massimo dell'attrito	$\mathbb{R}$	1.5
MIN_FRICTION	Valore minimo dell'attrito	$\mathbb{R}$	0.1

Considerazioni finali per differenti valori dei parametri

L'equilibrio di tutti questi parametri realizza un'esperienza di gioco gradevole e bilanciata. Durante la sua realizzazione e con prove successive, sono stati assegnati ai parametri valori ben precisi e cambiarne alcuni potrebbe portare il gioco ad essere troppo facile o troppo difficile.

Una buona manovrabilità della navicella del giocatore è strettamente collegata all'abilità dello stesso e fissata la velocità massima, il valore di INC_VELOCITY e dell'attrito *friction* competono a rendere il campo di gioco più o meno "scivoloso". Stesso discorso vale per BUL1_CHARGE e PLY_RECHARGE_RATE, e i corrispettivi BUL2_CHARGE, ENY_RECHARGE_RATE e SHOT_RATE per il nemico, che consentono di equilibrare in modo opportuno e bilanciato la cadenza di tiro di entrambe le navicelle.

Per quando riguarda invece i parametri dell'intelligenza artificiale, diverse simulazioni sono state fatte con differenti valori di RADAR_ERROR per valutare la precisione della mira dell'avversario. Data l'alta velocità dei proiettili, risulta quasi impossibile evitarli se sono stati sparati esattamente verso la posizione della navicella del giocatore, quindi un valore di $\mathrm{RADAR\_ERROR} = \pm 80\ pixel$ è un buon compromesso confrontato alla precisione che l'utente a sua volta potrebbe raggiungere.

I parametri NEW_POS_RATE e NEW_POS_ALERT rendono più frequente l'aggiornamento della posizione del nemico ed è quindi più difficile riuscire a colpirlo. Anche qui i valori suggeriti sono stati trovati per rendere il gioco bilanciato e pesato al livello scelto.

Bibliografia

[1] Manuale online Allegro 4 (https://www.allegro.cc/manual/4/)

[2] Wikipedia, POSIX Threads (https://en.wikipedia.org/wiki/POSIX_Threads)

[3] Giorgio Buttazzo. Slides del corso di Real Time Systems, 2020