Università di Cagliari
Facoltà di Scienze MM.FF.NN
Corso di Laurea in Informatica

TESINA CORSO SISTEMI OPERATIVI 1

Docente S. Carta (salvatore@unica.it)
Tutor L. Boratto (ludovico.boratto@unica.it)
A.A. 2011 -2012

GRUPPO

MARCO ARNONE NM 45895
MATTIA CADEDDU NM 45753
specifica versione 1.1

Questa versione di Space Invaders vuole essere una rappresentazione del tipico studente del CdL in Informatica di Cagliari. Difatti sull'astronave pilotata dall'utente è possibile notare una scritta “me” ovvero lo studente in questione, che si trova davanti 3 anni (3 livelli) di esami (di nemici) dal quale prendono il nome le navicelle nemiche.
Il comportamento di ogni elemento di gioco è descritto nell'apposito file dal medesimo nome, ad esempio il comportamento della navicella nemica alias spacecraft è descritto nel file spacecraft.c.
Questo comporta la creazione di una moltitudine di file necessari per l'esecuzione quindi il tutto è corredato da apposito makefile che gestisce tutte le operazioni necessarie per creare il file eseguibile.
Ogni elemento è gestito da un task a se, quindi avremo tanti task quanti sono gli elementi in gioco: 30 navicelle nemiche distribuite sui vari livelli + 4 rocce + 1 navicella gestita dall'utente + 10 possibili missili nemici + 2 missili dell'utente + 1 di controllo.
Nel main si inizializza tutto: grafica, strumenti di comunicazione, avvio dei task delle navicelle di primo livello, task della navicella pilotata dall'utente e ulteriori task delle rocce protettive. Una volta fatto questo si “trasforma” nel task che gestisce Engine.
E' intuibile capire che la funzione Engine (controllo) viene eseguita da un task che comunica, controlla, regola e stampa a video tutti gli altri task in gioco. Si occupa di far rispettare le varie regole che determinano rimbalzi, collisioni e proiettili andati a segno scagliati da entrambi le parti. Inoltre si occupa all'evenienza di creare navicelle di livello superiore o di far terminare il gioco comunicando l'esito con un messaggio stampato a video.
Come detto prima ogni elemento di gioco è descritto nell'apposito file dal medesimo nome. Alcuni sono molto semplici come il comportamento dei missili (sia nemici che non), altri un po' più complessi ma tutti hanno una struttura comune. In sostanza ogni elemento comunica a Engine il proprio stato di gioco. Attraverso un apposita struttura di nome GameObject gli invia molteplici informazioni attraverso l'input_pipe (quale elemento di gioco gestisce il task in questione, le vite residue, le coordinate, il livello etc.). Engine a seconda del tipo di elemento da cui riceve le informazioni dall'input_pipe, agisce di conseguenza con dovuti controlli specifici e eventuali risposte al task stesso attraverso la output_pipe, per esempio: *“se sei vicino ad un altra navicella, rimbalza cambiando direzione”*. Le eventuali risposte giungono al task specifico che a sua volta agisce secondo il comunicato ricevuto, per esempio: *“sono stato colpito, ora ho una vita in meno”*.
Ogni elemento è curato in modo tale da non uscire mai dal riquadro di gioco quando è in vita e ha la sua apposita funzione di inizializzazione dei dati del proprio GameObject.
Vite e dimensioni dei singoli elementi rispettano le specifiche dettate.
Anche la terminazione del gioco in esito positivo o negativo rispetta le specifiche.
I movimenti e la frequenza degli spari è dettata da una variabile che si incrementa di 1 per ogni passo (spara quando la variabile X vale tot). In questo modo le navicelle sparano e scendono in senso verticale in modo regolare (ogni tot passi e in maniera più frequente a seconda del livello) ma essendo la variabile in questione inizializzata random questo non avviene in contemporanea per tutte le navicelle.

In aggiunta sono presenti rocce, grafica, colori, effetti, stampe introduttive e finali al gioco e barra delle informazioni.
Le **rocce**, fungono da scudo protettivo, non si muovono e possono essere distrutte in due modi:

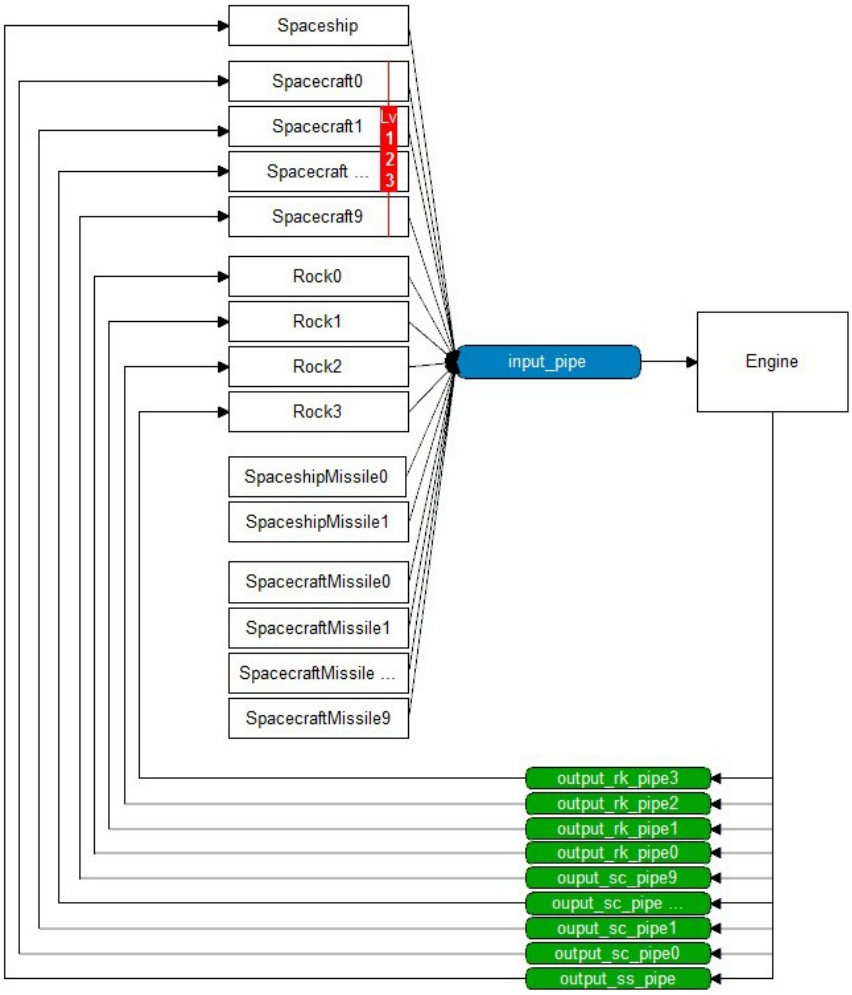
- vengono distrutte da proiettili sia nemici che amici (4 vite). E' presente una barra all'interno di esse la cui lunghezza e colore è dipendente dal numero di vite residue;
- collisione con una navicella nemica. In tal caso vengono distrutti entrambi gli elementi, roccia e navicella in questione a prescindere dalle vite residue.

La **parte grafica** (denominata texture) aggiunge informazioni cromatiche e grafiche per la più facile comprensione del gioco. Per le navicelle nemiche e per le rocce il colore rosso indica che con un altro colpo è possibile uccidere quell'elemento, il giallo indica vita media e potrebbero essere necessari 1 o 2 colpi se sono rispettivamente di 2° o 3° livello, il verde indica piena vita. Anche i proiettili, nemici o non, cambiano colore a ogni stampa.
Un riquadro delimita l'area di gioco rendendo comprensibile fino a dove un elemento può spostarsi.
La collisione tra proiettile e parete laterale provoca anch'esso un **effetto rimbalzo**, invertendo la direzione del proiettile.
Sotto l'astronave pilotata dall'utente è presente il reattore spaziale con tanto di fiamme che alternano colori gialli e rossi riproducento l'**effetto fuoco**.
Per un inizio meno brusco e violento è stata aggiunta una **schermata introduttiva** realizzata da due grandi composizioni in ASCII (per rimanere in tema il font è in stile *Star Wars*) che scorrono da sinistra verso destra

e viceversa componendo la scritta “Space Invaders” anch'essa colorata e lampeggiante.
La **difficoltà incrementa** più si sale di livello, ovvero una navicella di livello 3, spara e si muove in entrambi i sensi più velocemente di una navicella di livello 2. Stessa cosa accade per le navicelle di 2° livello con quelle del 1°.
Infine la **barra delle informazioni** visualizza i dati di gioco quali il punteggio accumulato, le vite residue, il livello attuale e ricorda all'utente le informazioni riguardanti i colori.

Lo schema sintetico seguente mostra le linee di comunicazione della prima versione dell'esercizio.

Unica pipe che con cui i vari processi comunicano a Engine i propri GameObject (molti a uno).
Tante singole pipe con cui Engine risponde con eventuali messaggi ai processi (uno a molti).



A causa della peculiarità bloccante delle pipe è stato necessario introdurre un meccanismo grazie al quale ogni task che scrive sull'input_pipe (missili a parte) ha una risposta in output_pipe e fino a quando non riceve questa risposta non va' avanti essendo la pipe bloccante.
Di conseguenza ogni GameObject dev'essere “autorizzato” con un messaggio “NOTHING” oppure corretto con un messaggio “COLLISION” oppure ancora informato di un colpo ricevuto “SHOOT”.

Abbiamo inoltre testato il programma per eliminare bug che si presentavano su diverse architetture hardware e su diverse distribuzioni ubuntu.
Altre eventuali scelte specifiche di gestione vengono commentate nel codice.