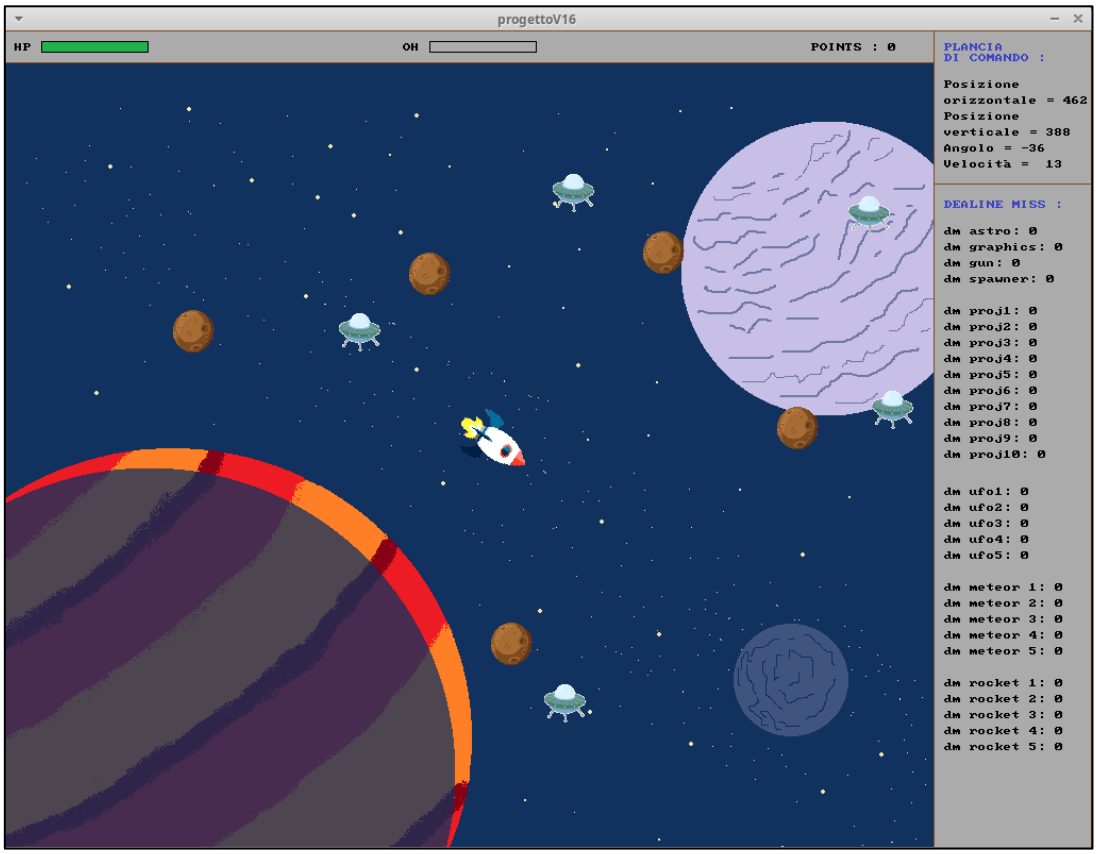
Università di Pisa
Dipartimento di ingegneria dell'informazione
Ingegneria robotica e dell'automazione

CORSO DI INFORMATICA E SISTEMI IN TEMPO REALE

"ROBOTICA E AUTOMAZIONE INVADERS"

Implementazione di un'applicazione in tempo reale



Data consegna: 08 – 01 – 2024

Cesare Palamara matricola: 559345 mail: c.palamara@studenti.unipi.it

Alberto Regoli matricola: 563246 mail: a.regoli7@studenti.unipi.it

1 DESCRIZIONE GENERALE DEL PROGETTO

Il progetto realizzato è un'applicazione in tempo reale sottoforma di videogioco che consiste nel manovrare un'astronave e cercare di distruggere una serie di navicelle nemiche prima di essere abbattuti.

2 MODELLO FISICO E COMPORTAMENTO DELLE ENTITA'

Nell'applicazione vi sono quattro tipi di corpi:

- l'astronave controllata dall'utente;
- le navette nemiche (ufo);
- i meteoriti;
- i proiettili (sia quelli sparati dall'utente che i missili generati dagli "ufo").

Di questi solo i primi tre sono dotati di massa, poiché l'inerzia dei proiettili viene trascurata. I corpi sono stati approssimati come dischi che si muovono su un piano privo di attrito e che possono urtare tra di loro. Di seguito verrà analizzato più nello specifico il comportamento di ciascun corpo.

2.1 ASTRONAVE

L'astronave è stata modellata come un veicolo dotato di due propulsori "principali", uno anteriore ed uno posteriore; il veicolo può, inoltre, ruotare su sé stesso e muoversi liberamente in tutte le direzioni. I comandi vengono forniti dall'utente tramite il mouse, per controllare la direzione, e la tastiera, per controllare i propulsori. La spinta generata da questi ultimi è costante e orientata come l'astronave; quando i propulsori vengono disattivati, il veicolo conserva la quantità di moto che aveva fino a quel momento e continua a muoversi con velocità costante. Per quanto riguarda l'orientazione, l'astronave cerca di allinearsi con la congiungente tra essa e il cursore grazie ad un controllo proporzionale. Per migliorare la manovrabilità del veicolo, vi è un limite massimo sul modulo della velocità ed è presente un comando per "frenare" l'astronave, indipendentemente dalla direzione in cui si sta muovendo.

2.2 NAVETTE NEMICHE (UFO)

Queste entità possiedono due stati distinti: uno passivo, in cui si muovono con moto randomico, ed uno attivo, nel quale entrano se l'astronave controllata dall'utente supera una la loro distanza di ingaggio; a quel punto gli ufo inizieranno ad inseguire l'astronave, cercando di portarsi in prossimità di essa. La velocità di inseguimento è proporzionale alla distanza tra ufo e astronave, e se quest'ultima viene raggiunta gli ufo inizieranno ad orbitarle intorno con moto circolare uniforme. Infine, gli ufo cercano di mantenere una certa distanza minima tra di loro, indipendentemente dallo stato in cui si trovano.

2.3 METEORITI

Sono corpi passivi, ovvero vengono inizializzati in una posizione randomica e rimangono fermi finché non entrano in collisione con un altro corpo (astronave, ufo, o un altro meteorite). Dopo la collisione, modellata come urto elastico, il meteorite si muoverà con moto rettilineo. A differenza degli altri corpi, però i meteoriti hanno un damping sulla velocità: questa scelta è stata fatta per evitare di rendere il campo di gioco troppo caotico.

2.4 PROIETTILI

Come accennato in precedenza, esistono due tipi di proiettili:

- quelli sparati dall'astronave controllata dall'utente, che si muovono con moto rettilineo uniforme finché non collidono con un ufo o un meteorite, o escono dai bordi dell'area di gioco;
- quelli sparati dagli ufo, che si comportano come dei missili a ricerca grazie ad un controllo di tipo proporzionale sulla direzione del vettore velocità mantenendo costante il modulo. Ogni ufo può sparare al massimo un missile alla volta, e solo se sta ingaggiando l'astronave.

È prevista, inoltre, che l'arma dell'astronave si "surriscaldi" se vengono sparati troppi colpi senza interruzione. A quel punto l'utente non potrà sparare fino a che l'arma non sarà completamente raffreddata.

3 INTERFACCIA UTENTE

L'interfaccia utente è composta da due rettangoli: uno nella parte superiore dello schermo, su cui sono riportate la barra della salute dell'astronave, lo stato di surriscaldamento dell'arma e i punti totalizzati, e un altro sul bordo destro dove sono riportate la posizione, la velocità (in modulo), l'orientazione dell'astronave e le "deadline miss" accumulate da tutti i task durante l'esecuzione dell'applicazione. Per quanto riguarda i comandi, questi ultimi vengono visualizzati, insieme alle regole di gioco, in una schermata separata prima di entrare nella finestra di gioco vera e propria. L'interfaccia è rappresentata nella copertina della relazione.

4 SCELTE DI DESIGN DEL CODICE

Per poter realizzazione l'applicazione, sono state inizialmente create due librerie, da affiancare a quelle standard: "ptask" e "utility_lib"; successivamente si è passati alla scrittura del codice sorgente.

4.1 PTASK

La libreria "ptask" fa da interfaccia alla libreria standard "pthread". In essa sono definite le strutture e funzioni necessarie per la gestione di task concorrenti, tra cui la creazione e l'attivazione di task, la scelta della politica dello scheduler, l'attesa del periodo successivo e il conteggio delle deadline miss. Sono inoltre definite delle funzioni che permettono una gestione facilitata del tempo e delle funzioni per estrapolare e modificare alcuni parametri dei singoli task, come il periodo o la deadline.

4.2 UTILITY_LIB

Nella libreria "utility_lib" sono state definite la struttura di stato comune a tutti i corpi descritti in precedenza, una struttura per la rappresentazione di vettori di R^2 e le funzioni ausiliarie utilizzate dai vari task dell'applicazione. Nella struttura di stato sono riportate per ogni corpo i seguenti campi:

- le componenti di posizione e velocità;
- la massa e la dimensione del corpo;
- la "salute" dell'oggetto (come nel caso di astronave e ufo);
- una variabile che indica se l'oggetto è attivo o meno.

Va specificato che non tutti i corpi utilizzano tutti i campi della struttura, ma dato che la maggior parte dei campi è comune a tutti i corpi, come posizione, velocità e dimensioni, si è scelto di utilizzare una struttura unica. Per quanto riguarda le funzioni ausiliari, alcune si occupano della gestione dello stack che viene utilizzato dai task dell'arma e dei proiettili; altre, invece, servono per raggruppare procedure di calcolo comuni a tutti i task, come prodotti scalari, distanze, moduli e risoluzione di collisioni, in modo da compattare il codice sorgente.

4.3 CODICE SORGENTE

Il codice sorgente racchiude tutto ciò che è necessario per la realizzazione dei task (costanti, variabili globali e istruzioni del task) e l'attivazione di essi.

4.3.1 COSTANTI

Nel codice sorgente sono state prima di tutto definite le costanti globali, come ad esempio il numero massimo di corpi attivi per tipologia, i valori massimi della velocità per i vari oggetti, la salute massima per l'astronave e gli ufo, il punteggio da raggiungere per vincere, e i bordi della finestra e dell'area di gioco.

4.3.2 VARIABILI GLOBALI

Sono state poi definite le variabili di stato globali per ciascuna entità, utilizzando la struttura definita nella libreria di utility, insieme ad alcune variabili per raccogliere informazioni dai task, come lo stato di surriscaldamento dell'arma o il punteggio, al fine di utilizzarle nel task grafico, ed un flag di terminazione per tutti i task; infine, vengono definiti i semafori per proteggere le sezioni critiche.

4.3.3 INIT() E MAIN()

Prima della realizzazione dei task, è stata creata una funzione "init()", che si occupa dell'inizializzazione dello stato di ciascun corpo e delle altre variabili globali; quest'ultima verrà richiamata nel "main()", insieme alle funzioni di creazione e attivazione dei task. Tramite il "main()", vengono inoltre richiamate le istruzioni per mostrare la schermata iniziale, le regole e i comandi, e, dopo il termine dei task principali, la schermata di "vittoria" o "game over" in base all'esito della partita.

5 TASK

Per l'applicazione sono stati utilizzati 8 task: uno per ogni tipologia di oggetto (astronave, meteoriti, ufo, proiettili e missili), 2 task che si occupano di gestire l'attivazione di determinati corpi, come ufo e proiettili, che durante il gioco vengono "creati" o "distrutti", ed infine un unico task grafico che si occupa di aggiornare la schermata di gioco e l'interfaccia utente.

Tutti i task si appoggiano a variabili locali per svolgere i calcoli, acquisendo le informazioni sullo stato al passo corrente dalle variabili globali, per poi aggiornarle al passo successivo; questa scelta è stata fatta per ridurre le sezioni critiche a semplici operazioni di lettura e scrittura. Per rendere omogenee le operazioni di integrazione è stata utilizzata la funzione "task_period()", che estrapola il periodo dal task, il quale viene poi usato per calcolare il "delta" di integrazione.

Di seguito verranno analizzati i singoli task.

5.1 TASK_ASTRO

Questo task si occupa di aggiornare lo stato dell'astronave, in termini di posizione, velocità e salute, di tenere conto del punteggio totalizzato durante la partita, e infine di attivare il flag di terminazione di tutti i task in caso di vittoria o di sconfitta, o di chiudere l'applicazione utilizzando il tasto ESC. Nel codice viene utilizzata la funzione "thrust()" definita nella libreria di utility per l'integrazione dell'accelerazione, la quale viene controllata premendo i tasti SPACE e S, relativi rispettivamente ai propulsori posteriore e anteriore, o con il tasto W che riduce la velocità dell'astronave. Per la gestione dei comandi sono stati usati degli "if()", invece che uno "switch()", per far sì che propulsori e freno potessero essere usati insieme per compiere manovre più precise. Inoltre, quando l'astronave si avvicina troppo ai bordi dello schermo, la funzione "edges()" la riporta al bordo opposto. Questa gestione avviene anche per ufo e meteoriti. Infine, il task si occupa anche di rilevare collisioni con i missili nemici per poi aggiornare la salute, e, se questa è minore del valore massimo, ma maggiore stretta di zero, viene rigenerata passivamente. La condizione di sconfitta dipenderà fortemente da questa.

5.2 TASK_PROJ

Questo task si occupa di gestire il singolo proiettile sparato dall'astronave. È un task con comportamento sia periodico che aperiodico. Il codice è composto da due cicli annidati per gestire questa doppia natura: quello esterno, relativo al comportamento aperiodico, in cui il task attende il segnale di attivazione del proiettile, viene inizializzata la posizione e direzione di quest' ultimo al momento dello sparo, e quello interno, relativo al comportamento periodico, in cui vengono svolti i calcoli del moto e rilevato se il proiettile è uscito dai bordi. Quando i task proiettile vengono creati, il loro indice viene copiato all'interno di uno stack per tener conto di quanti colpi vengono usati.

5.3 GUN

Questo task si occupa dell'attivazione dei proiettili e del surriscaldamento dell'arma. Ogni volta che viene premuto il tasto sinistro del mouse, l'indice che in quel momento è in cima allo stack viene estratto, e viene attivato il proiettile corrispondente; quando il proiettile si distrugge, l'indice ritorna nello stack e il proiettile può essere nuovamente sparato se l' arma non ha superato il limite massimo di surriscaldamento.

5.4 TASK_UFO

Questo task si occupa della gestione degli ufo e dell'attivazione dei loro missili. In modo analogo a quanto visto per il task proiettile, anche questo ha comportamento sia aperiodico, in cui oltre alla posizione viene inizializzata anche la salute, che periodico. Il codice della parte periodica è costituito principalmente da un "if()", che fa passare l'ufo da comportamento passivo ad attivo e viceversa, e da una sezione comune a entrambi gli stati per il calcolo dei danni e la gestione dei bordi.

5.5 UFO_SPAWNER

Questo task si occupa dell'attivazione degli ufo dopo che essi vengono distrutti. A differenza del task gun, non viene usato uno stack, ma ogni ufo ha un timer che conta quanto tempo è trascorso dalla sua ultima distruzione. Quando il timer raggiunge un certo valore, esso viene azzerato, e l'ufo corrispondente viene riattivato.

5.6 ROCKET

Questo task ha un comportamento analogo a quello dei proiettili dell'astronave e si occupa della gestione dei missili sparati dagli ufo. A differenza dei proiettili, questi missili hanno un "tempo di vita" limitato, ovvero vengono distrutti entro certo limite di tempo se non collidono con l'astronave.

5.7 TASK_METEOR

Questo task si occupa della gestione dei meteoriti e delle collisioni con gli altri oggetti. Come già accennato in precedenza, è stata implementata la funzione "collision()" nella libreria di utility per la risoluzione delle collisioni tra due oggetti ed è usata dal task nel caso di collisioni meteorite-meteorite, meteorite-astronave e meteorite-ufo, semplificando la scrittura del codice. Le collisioni

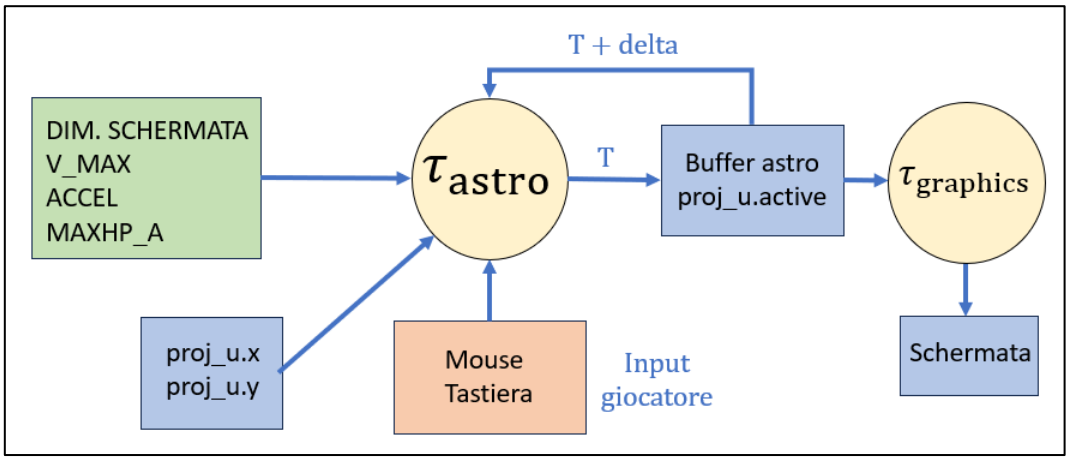
con proiettili o missili sono state gestiste solamente con la “distruzione” di questi ultimi, senza modificare la quantità di moto del meteorite.

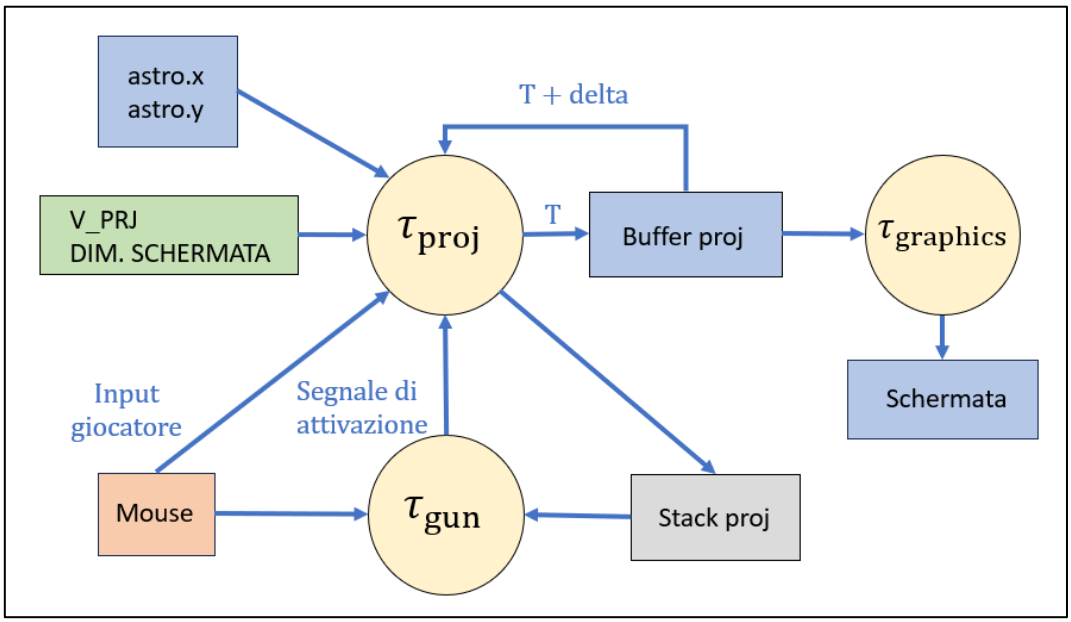
5.8 GRAPHICS

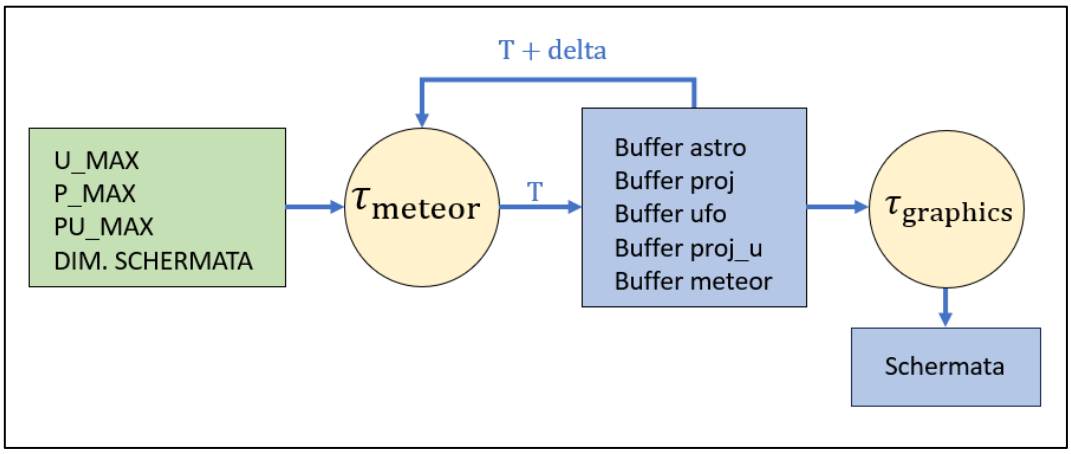
Questo task si occupa della realizzazione della finestra grafica. Esso contiene le istruzioni per inizializzare i colori, caricare e disegnare le bitmap dei vari oggetti e rappresentare l’interfaccia grafica. Per evitare sfarfallii, è stata utilizzata una bitmap come buffer grafico, sulla quale, ad ogni frame, viene disegnata la finestra di gioco prima di essere visualizzata a schermo.

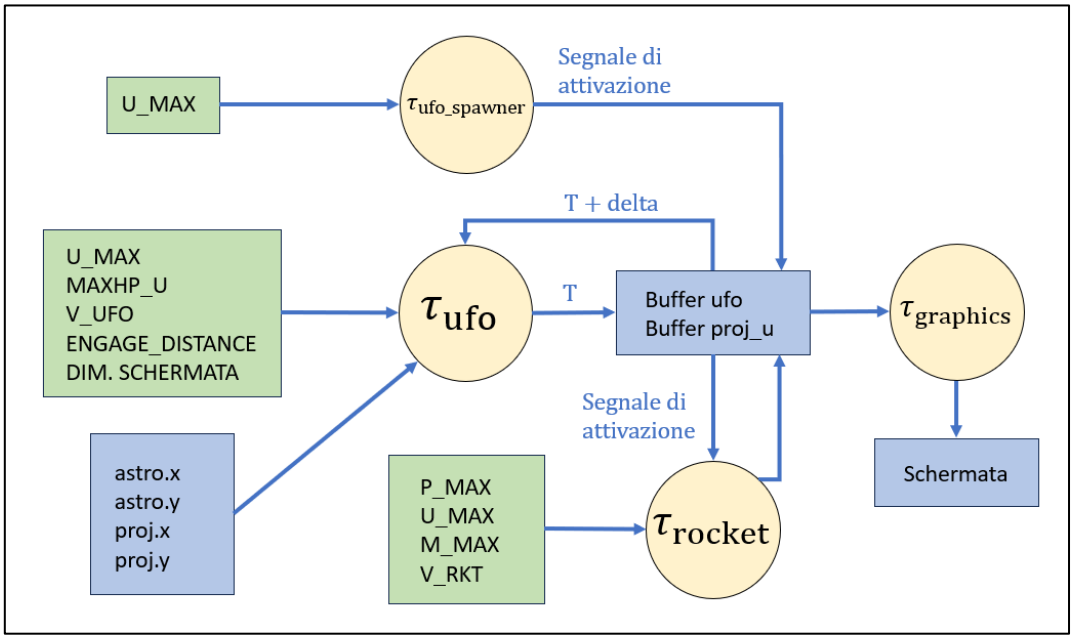
6 DIAGRAMMI TASK-RISORSE

Qui sotto vengono riportati quattro diagrammi in cui si mostra l’interazione dei task tra di loro e con le risorse necessarie, ovvero con i singoli buffer, gli input dati dal giocatore e le costanti globali del problema. Queste ultime vengono raccolte nei rettangoli verdi e per quelle relative alla dimensione dello schermo vengono racchiuse in “dim. schermata”.









7 PARAMETRI DEI TASK

I task sono caratterizzati da quattro parametri: periodo (T), deadline relativa (D) che in questo caso coincide con il periodo, la priorità (P) e il numero di volte che il task viene usato (N).

Per la scelta dei periodi si è usato come specifica il frame rate dell' applicazione, che è stato fissato ad un massimo di 50 fps garantendo una certa fluidità dell' immagine. Quindi il periodo del task grafico assume il valore di 20 millisecondi; di conseguenza i task che si occupano di aggiornare lo stato degli oggetti da rappresentare dovranno avere un periodo al più uguale a quello del task grafico. Per quanto riguarda i periodi dei task "ufo_spawner" e "gun", non sono legati al task grafico, ma sono vincolati rispettivamente al tempo di rinascita degli ufo (10 secondi) e alla cadenza di fuoco dell'arma dell'astronave (4 colpi al secondo). Questi limiti garantiscono un margine più ampio di scelta dei periodi.

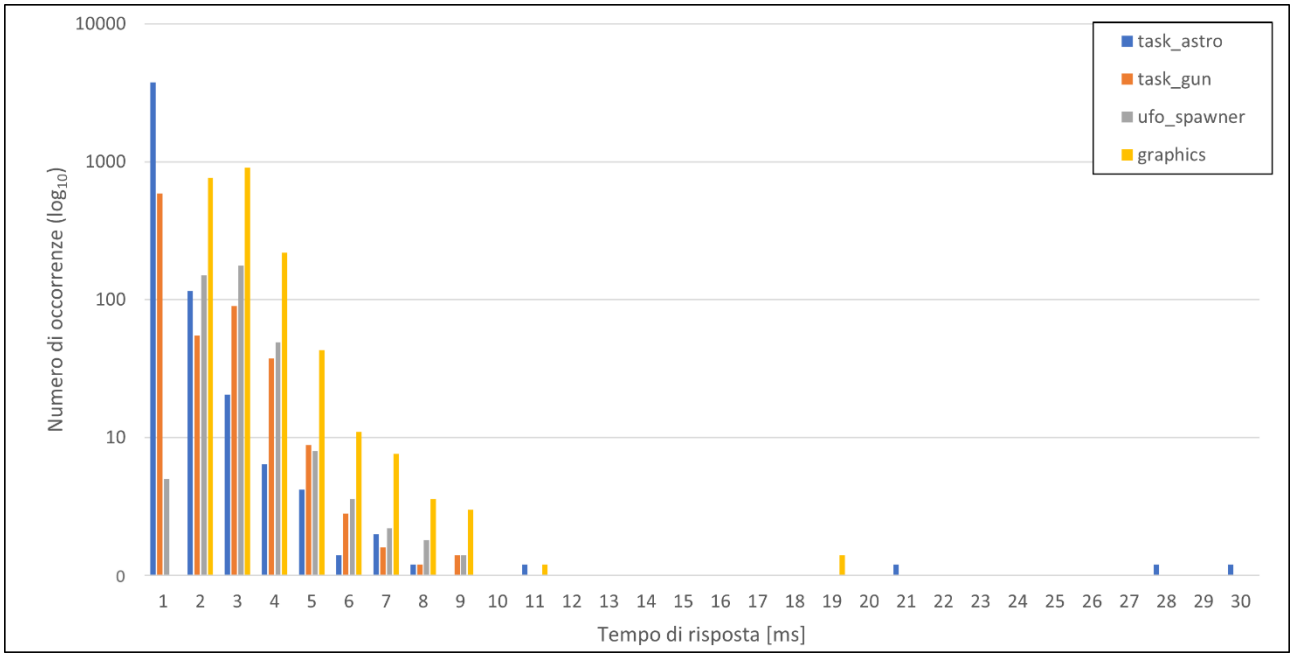
Per quanto riguarda le priorità, visto che periodi e deadline coincidono, queste sono state assegnate secondo il criterio Rate Monotonic.

Infine, la scelta del numero di task è stata fatta per garantire un numero ridotto di corpi a schermo e per poter implementare buffer di gestione come gli stack. Di seguito vengono riportati in tabella i valori di ogni task.

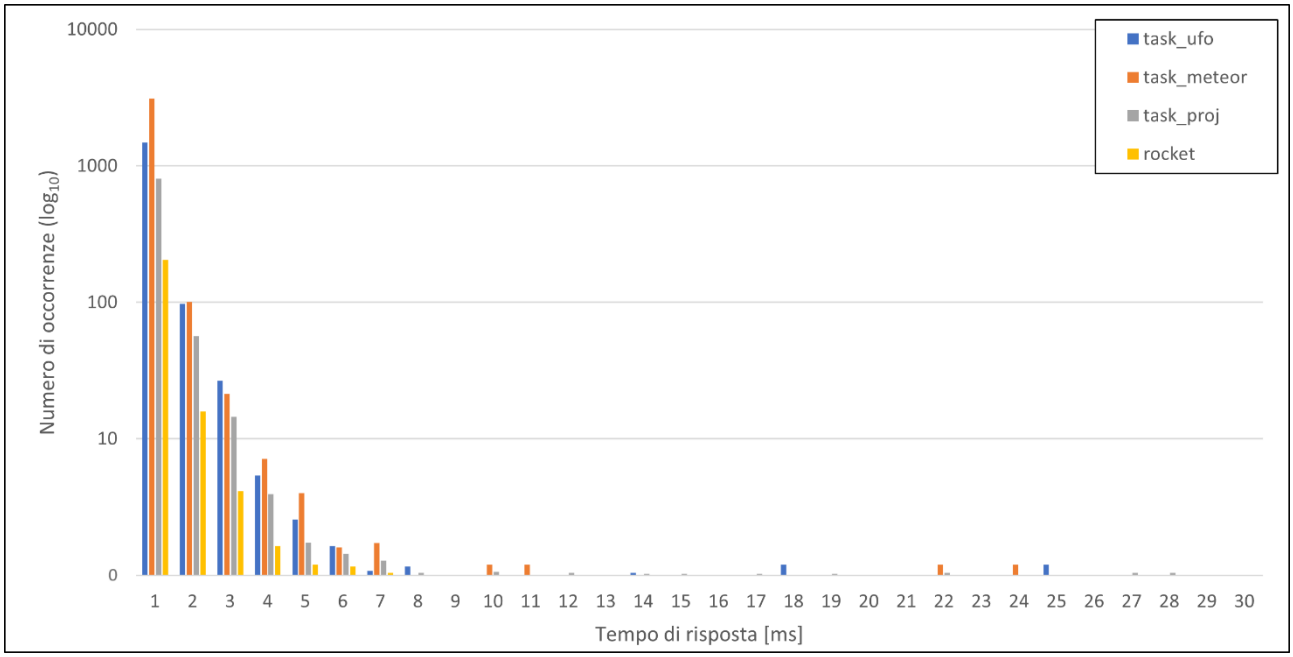
Task	Periodo (T)	Deadline relativa (D)	Priorità (P)	Numero di task (N)
task_astro	10	10	30	1
task_proj	5	5	40	10
gun	50	50	10	1
task_ufo	15	15	20	5
ufo_spawner	100	100	5	1
rocket	8	8	35	5
task_meteor	12	12	25	5
graphics	20	20	15	1

8 DATI SPERIMENTALI

Nei grafici sottostanti è stata riportata la distribuzione dei tempi di risposta dei singoli task per analizzare il comportamento dell'applicazione. Per la raccolta dei dati sono state eseguite una decina di partite dalla durata media di un minuto e successivamente sono stati processati i tempi.



Nel precedente istogramma vengono raggruppati i task singoli ed è possibile osservare che la maggior parte dei tempi di risposta sono molto piccoli per i 4 task tanto che si attestano tra 1 e 2 millisecondi. Tuttavia, sono state riscontrate per il task astro delle deadline miss, tutte quante avvenute in un'unica sessione; questo però non è un evento ricorrente e si può concludere che queste siano avvenute a causa di una semplificazione nella gestione dei tempi, ovvero non si è considerato i tempi degli interrupt e di eventuali bloccaggi delle risorse.



Nel secondo grafico vengono riportati i dati relativi ai task multipli. Anche in questo caso i tempi di risposta sono molto contenuti. È stato riscontrato nelle prove che il “task_proj” ha avuto due tipi di deadline miss: quelle sporadiche, come per il task relativo all’astronave, e quelle più ricorrenti, dovute a un tempo di risposta leggermente superiore alla sua deadline. Questo fatto è probabilmente dovuto alla scelta di un periodo troppo piccolo; infatti, alzando periodo e deadline al valore di 7 millisecondi le prestazioni del task sono mediamente migliori.

Trattandosi di un’applicazione non safety-critical, anche la presenza di qualche deadline miss sporadica è comunque tollerabile e non vi è una degradazione delle prestazioni.

In conclusione, l’applicazione funziona correttamente ed è possibile implementare in futuro funzionalità aggiuntive come: diversi tipi di armi, diverso controllo dell’astronave e modifica del comportamento dei nemici.