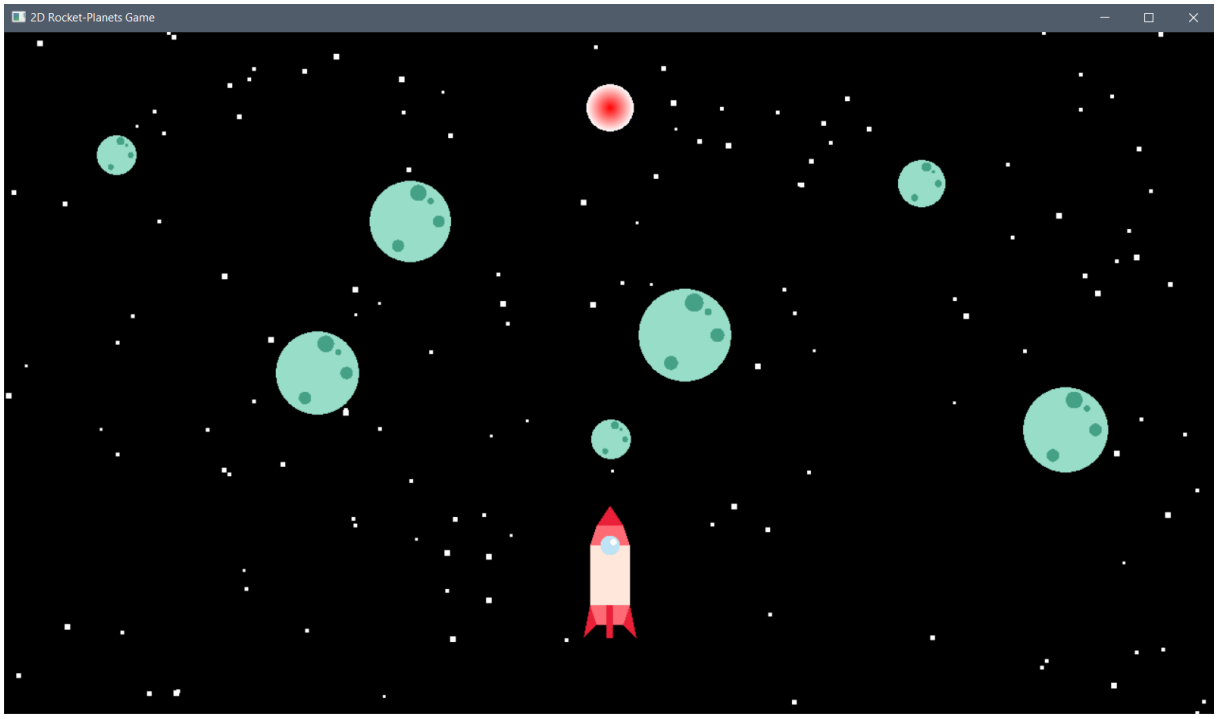
FONDAMENTI DI COMPUTER GRAPHICS M

LAB 02

Il secondo laboratorio ha previsto la creazione di un gioco in 2D mediante le API fornite da OpenGL. L'obiettivo è stato quello di utilizzare correttamente le varie primitive offerte e comprendere la dinamica di aggiornamento (refresh) dello schermo per dare dinamicità alla scena. In particolare, nel gioco creato (vedi immagine sottostante), per vincere bisogna riuscire a portare il razzo sul "checkpoint", ovvero fare in modo che il razzo tocchi il cerchio bianco e rosso nella parte superiore della finestra, evitando però i sette pianeti che si muovono orizzontalmente.



1. Componenti e composizione della scena

I componenti principali della scena sono, dunque:

- Il **razzo** che è composto da un corpo principale (costituito da un triangolo, due trapezi e un rettangolo), gli "alettoni" inferiori (costituiti da due triangoli e un rettangolo) e l'oblò (costituito da due cerchi, ovvero l'oblò vero e proprio e il riflesso della luce sullo stesso);
- I **pianeti** (sette): ogni pianeta ha dimensione diversa (assegnata casualmente all'avvio del gioco) ed è costituito da 5 cerchi (uno per il corpo principale e 4 per i crateri);
- Il **checkpoint**, che è un semplice cerchio di dimensione e posizione fissa;
- Le **stelle**, che sono semplici punti di dimensione variabile.

Ogni componente è rappresentato per mezzo di primitive base, ovvero punti, linee e triangoli, per cui trapezi, triangoli e cerchi sono rappresentati come insiemi di triangoli (la generazione dei triangoli è fatta mediante metodi dedicati come `drawRectagle`). Nelle immagini sottostanti sono mostrati i profili dei triangoli che generano i vari oggetti della scena (da sinistra a destra: razzo, pianeta, checkpoint).

2. Dinamica

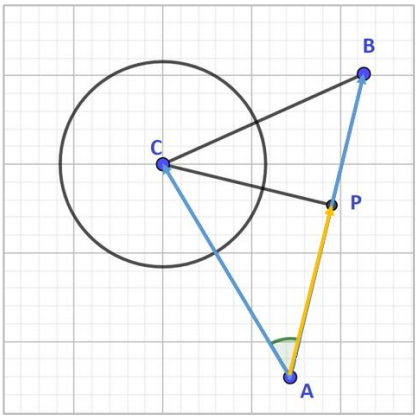
La dinamica del gioco è stata gestita mediante le funzioni di update fornite da glut (una per ogni entità), per cui ad ogni update, oltre a modificare i valori delle posizioni/dimensioni degli elementi, il programma invoca automaticamente la funzione drawScene, che scatena a sua volta il refresh del frame. Per quanto riguarda la dinamica degli oggetti presenti nella scena:

- Il razzo può muoversi nelle quattro direzioni mediante i tasti 'w', 'a', 's', 'd' e la pressione di tali tasti va ad incrementare la velocità del razzo. Per cui, il moto è modellato secondo la legge $x_t = x_{t-1} + v_t$. Inoltre al moto orizzontale è stata aggiunta anche un'inclinazione (max 20°), che viene incrementata seguendo $a_t = a_{t-1} + 1$. Nel caso in cui vengano raggiunti i bordi destro e sinistro della finestra, la direzione della velocità viene invertita e il valore dimezzato, mentre l'angolo viene posto a 0°;
- I pianeti, invece, hanno una direzione di movimento prefissata (a differenza della dimensione) e la posizione viene incrementata di 1 in tale direzione ad ogni update. Tuttavia, una volta che il pianeta si è allontanato di d dalla posizione iniziale, la direzione di movimento viene invertita e rimane tale fino al raggiungimento di una distanza *-d*, sempre dalla posizione inziale;
- I punti che rappresentano le stelle sono fissi, ma variano in dimensione, all'interno di un range: partono da una dimensione iniziale assegnata casualmente e poi si incrementa la dimensione di ± 1 ad ogni update.

3. Collision detection

Ad ogni update della posizione del razzo (e dei pianeti), è inoltre necessario controllare che non vi sia stata una collisione tra razzo e pianeti o che il razzo abbia raggiunto il checkpoint della vittoria. È necessario, dunque, un algoritmo di collision detection che si occupi di questo

controllo, in particolare andando a verificare l'assenza (per ogni pianeta, presenza per il checkpoint) di intersezioni tra il cerchio di centro *C* (centro del pianeta) e raggio *r* (raggio del pianeta) e il poligono che costituisce l'hitbox del razzo.



Per verificare che non vi sia collisione tra un cerchio e un poligono, è sufficiente verificare che tutti i suoi lati non collidano con il cerchio in questione. Quindi, basterà iterare sui lati del poligono e applicare un algoritmo di collision detection tra cerchio (come decritto sopra) e un segmento $\overline{AB}$ (lato del poligono):

- Per prima cosa, bisognerà verificare la distanza tra gli estremi del segmento e il centro del cerchio: se questa fosse minore del raggio *r*, allora l'algoritmo di collisione terminerebbe immediatamente.
- Altrimenti, sarà necessario calcolare la distanza tra *C* e $\overline{AB}$ come $\|P - C\|$ dove *P* è il punto del segmento $\overline{AB}$ più vicino al centro *C*. In particolare, si sfrutta il prodotto scalare per calcolare la proiezione di *C-A* su *B-A*, così da ottenere il vettore *P-A:*

$$P - A = \frac{\langle P-A\,, B-A\rangle}{\|B-A\|\|B-A\|}(B - A)$$

A questo punto, per ottenere *P* sarà sufficiente calcolare *A + (P-A)*. Dopo aver verificato che *P* appartenga effettivamente a *P*, basterà calcolare la distanza tra *C* e *P* e verificare che questa sia maggiore/minore del raggio *r*.