Tank Game

Revision & improvements



Sommario

[Introduzione 3](#_Toc463470619)

[La struttura generale del codice 3](#_Toc463470620)

[Generazione del terreno 3](#_Toc463470621)

[Terreno “infinito” 4](#_Toc463470622)

[Carro nemico e Intelligenza artificiale 4](#_Toc463470623)

[Dinamica dei proiettili 4](#_Toc463470624)

[Animazioni 3D e suoni 4](#_Toc463470625)

[Intelligenza Artificiale 5](#_Toc463470626)

[Inclinazione del carro seguendo il terreno 5](#_Toc463470627)

[Camera improvements 6](#_Toc463470628)

[Bibliografia 7](#_Toc463470629)

## Introduzione

Abbiamo ereditato dal progetto sviluppato in classe il modello del carro e del proiettile e la gestione del movimento delle varie parti che lo compongono. In seguito descriviamo in ogni distinto paragrafo tutti i miglioramenti che sono stati realizzati nel nostro progetto. Abbiamo provveduto a commentare il codice per avere una spiegazione più approfondita delle parti di codice che abbiamo creato.

## La struttura generale del codice

## Generazione del terreno

La generazione del terreno avviene in maniera diversa rispetto a quella utilizzata a lezione. Infatti in precedenza veniva creato un modello FBX del terreno ed esso veniva importato all’interno del gioco come un modello distinto. Nel nostro progetto il terreno è diventato una vera e propria “**heightmap**”, ovvero il codice contenuto nella classe “**Terrain.cs**” genera il terreno a partire da un file immagine in scala di grigi da **128x128 pixel**. Ad ogni sfumatura di grigio viene associata una diversa altezza, passando dal punto più basso della mappa rappresentato dal nero al punto più altro rappresentato dal bianco. Il modello del carro, essendo di dimensioni molto grandi rispetto alla mappa, è stato scalato di un fattore 0.001. Questo fattore è stato tenuto in considerazione per tutti i calcoli eseguiti nel codice.

Attraverso lo shader “**terrain.fx**” dotiamo il terreno generato di 3 differenti texture, a seconda dell’altezza del terreno in ogni punto.

### Terreno “infinito”

Infine abbiamo provato a rendere il terreno “infinito” per il giocatore, in modo che il carro possa avanzare indefinitamente nello spazio circostante. Per fare ciò abbiamo per prima cosa generato la nebbia all’orizzonte (codice contenuto all’interno dello shader “**terrain.fx**”). In secondo luogo abbiamo generato attorno alla heightmap principale in cui si svolge il gioco, altre 8 heightmap formando una griglia da 3x3 terreni (figura sottostante). Il giocatore una volta arrivato al bordo del confine della mappa di gioco, viene teletrasportato dalla parte opposta della mappa avendo l’illusione di continuare ad avanzare all’infinito. E’ stato possibile realizzare quest’illusione grazie all’aiuto della nebbia e grazie ad una discreta simmetria spaziale della mappa centrale (dove si svolge il gioco)

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | GIOCO |  |
|  |  |  |

Tabella - Griglia mappa di gioco



Figura - Heightmap centrale

## Carro nemico e Intelligenza artificiale

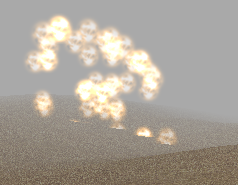
Abbiamo deciso di generare un altro carro armato uguale a quello del giocatore e simulare quindi uno scontro armato tra i due carri. Abbiamo gestito tutta la dinamica dei proiettili, creato un’animazione 3D che riproduce le esplosioni e inoltre abbiamo dotato del nemico di un intelligenza artificiale in grado di inseguire il nostro carro per poterlo colpire.

### Dinamica dei proiettili

La funzione che gestisce la dinamica dei proiettili è void UpdateBullet(Tank Shooter, GameTime gameTime) che riceve come parametro in ingresso, oltre al **gameTime**, anche il modello che lo ha generato (sparato).

La funzione è contenuta in un ciclo foreach nell’ Update() nel quale vengono iterati entrambi i carri. Se il proiettile si trova ancora in aria, viene aggiornata la sua posizione. Altrimenti se il proiettile si trova a contatto con il terreno, viene fatto esplodere. Nel caso in cui il proiettile sia in aria da più di 6 secondi, viene cancellato poiché non toccherebbe mai il suolo probabilmente. Inoltre si verifica se il proiettile ha colpito l’altro carro con la funzione bool CollisionCheck(Tank tank, Bullet bullet) che ritorna **true** in caso positivo. Nel caso sia colpito il carro nemico dal giocatore, una routine si occupa di generare il carro nuovamente in un intorno del carro del giocatore.

### Animazioni 3D e suoni

Abbiamo inserito nel gioco alcuni suoni che simulano lo sparo e l’esplosione del proiettile attraverso la classe SFXManager già inclusa nelle librerie di Monogame. Inoltre abbiamo creato un’animazione 3D che raffigura l’esplosione di un proiettile del carro armato contro un oggetto. Sono state create le classi **Particles** e **ParticlesManager** e lo shader **Particles.fx** che si occupa della gestione grafica dell’animazione.

Per generare l’esplosione nel codice basterà eseguire la funzione void ParticleManager.MakeExplosion(Vector3 position, int particleCount) che ingresso contiene la coordinata nella quale generare l’esplosione e il numero di particelle che comporranno l’esplosione. Abbiamo scalato opportunamente le particelle per essere più realistiche e abbiamo sperimentato che il numero di particelle ottimale è 50.

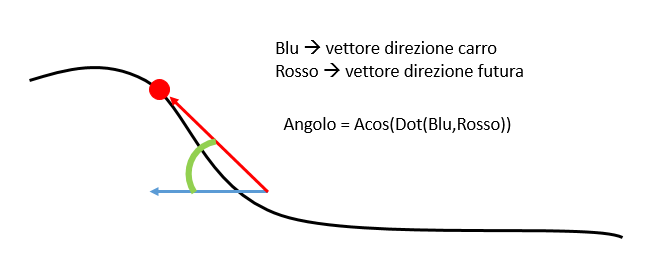
### Intelligenza Artificiale

Abbiamo dotato il carro nemico di un’intelligenza artificiale che fosse in grado di sparare in maniera autonoma verso il carro del giocatore. La funzione che si occupa della gestione dell’IA è void EnemyAutoFiring(GameTime gameTime, Tank Enemy). In questa funzione il carro nemico, una volta che il giocatore rientra entro una certa distanza limite, orienta la sua torretta e il suo cannone verso il bersaglio e comincia a sparare. Il codice è stato opportunamente commentato per favorirne la comprensione.

## Inclinazione del carro seguendo il terreno

Abbiamo cercato di ottimizzare il gioco cercando di fare seguire al carro armato del giocatore il profilo del terreno. Per fare ciò abbiamo dovuto implementare metodi in grado di inclinare il carro in modo conforme al terreno sottostante. Grazie all’utilizzo di un’**heightmap** è stato possibile conoscere in tempo reale l’altezza di ogni singolo punto della mappa. Così abbiamo potuto sfruttare queste informazioni per poter inclinare il carro dell’angolo corretto per seguire la pendenza del terreno in ogni punto. La funzione principale che si occupa di gestire l’inclinazione è void DoTankTransform(Tank tank).

In questo metodo vengono inizializzati un vettore contente la direzione del carro e un altro perpendicolare a quest’ultimo. Nella funzione float NextInclination(Vector3 tankDirection) viene calcolato l’angolo di inclinazione rispetto alla verticale.



In maniera analoga viene calcolato anche l’angolo di inclinazione rispetto all’asse Z tramite la funzione float ZInclination(Vector3 tankPerpDirection).

Una volta calcolati entrambi gli angoli di inclinazione, compio una trasformazione del carro che integra contemporaneamente la rotazione attorno all’ asse Y e le due inclinazioni tramite il comando:

PlayerTank.BoneTransform(0, Matrix.CreateRotationY(tank.BodyRot) \*

Matrix.CreateFromAxisAngle(tankAxis, tank.yinclination) \*

Matrix.CreateFromAxisAngle(tankZAxis, tank.zinclination));

## Camera improvements

Infine abbiamo migliorato la telecamera, in modo che riesca a seguire il giocatore automaticamente seguendo i suoi spostamenti. Nel caso di inclinazione del carro, la camera zooma leggermente sul carro per rendersi conto dell’effettiva inclinazione. E’ sempre possibile modificare la vista della camera ruotando il mouse per guardarsi attorno.

# Bibliografia

* Xna 3.0 Game Programming Recipes, Riemer Grootjans, APress, 2009
* <http://www.riemers.net/> - Tutorial per XNA
* <http://rbwhitaker.wikidot.com/monogame-tutorials> - Tutorial per Monogame
* Aulaweb course