progetto di Interactive Graphics - DragonSnake21

La Sapienza Università di Roma

Michele Ciciolla Flavio Lorenzi Francesco Cassini Matteo Ginesi

1869990 1662963 785771 1832198

prof. Schaerf

INDICE

1 Breve Presentazione del lavoro

L' applicazione qui descritta è stata pensata come un riadattamento del celebre gioco Snake arricchendone l' ambiente con texture e personaggi.

Tutto il team ci tiene a sottolineare che il lavoro è stato costruito da zero in tutte le sue parti. I quattro modelli gerarchici e le relative animazioni sono state curate componente per componente.

In base alla scelta dell' ambiente in cui giocare sono stati creati 3 ambienti ben distinti che hanno in comune solo la struttura dei modelli sopra citata. Per ognuna di essere infatti cambiano texture e musica.

Il funzionamento si basa sull' interazione dell' utente mediante la tastiera con i comandi WASD per il movimento planare e ZX per quello spaziale. Lo scopo del gioco è quello di sopravvivere il più possibile cibandosi dei target che sono presenti sulla mappa: un uovo, una pecora ed una papera. All' aumentare dello score raggiunto aumenta anche la velocità di movimento che rende il gioco più difficile.

L' unico modo per generare un gameOver è infatti quello di oltrepassare i limiti mappa stessa.

2 Ambiente di sviluppo ed aspetti tecnici

Il lavoro è stato pensato ed implementato partendo dalle linee guida su WebGl offerte durante il corso ampliandone il ventaglio di strumenti grazie alla ben più ricca libreria THREE.js.

2.1 THREE.js e WebGL

Il nostro gioco è basato principalmente su due processi: una pagina HTML di intro col titolo ed i bottoni, in cui si chiede all' utente di scegliere un ambiente di gioco; ed il gioco vero e proprio implementato invece usando THREE.js e WebGl.

2.1.1 HTML

La sezione head dello script html introduce il canvas in cui è fissato la background image, gli input specifici per i tre bottoni a cui sono collegate le classi alignleft e alignright che permettono di agire in modo univoco su di essi. Tutti i titoli seguono lo stile CSS di una delle due classi gametitle e gamevariant. Inoltre ogni ID è specificato con # includendo la sua posizione nello spazio e visibilità iniziale. Un esempio di bottone è quello associato alla scelta di ambientare il gioco su Marte (The Red Planet) al cui click permette il passaggio alla schermata di gioco associata; cosi proposto:

```
<button class="link alignleft" style="font-size:2.5em;" id="Mars">The Red Planet</button>
<img src="./textures/intro-mars.png" style="position :absolute;"id="intro-mars">
```

La scelta dell' utente è registrata grazie alla funzione document.getElementById().onclick presente sullo script app.js che legge l'ID specificato e avvia la procedura al suo interno.

2.1.2 Javascript

Dal lato Javascript a WebGl è stato delegato il background del lavoro, è infatti possibile notare nello script app.js come l' oggeto camera, renderer e tutto il ciclo di animazione si basi su tale libreria. Di fondamentale utilizzo sono cui i metodi setTitle() per la gestione delle scritte a schermo, main() per la creazione dell' oggetto gioco, game.music per la scelta dell' audio da riprodurre.

THREE.js è stato di fondamentale utilità nella creazione dell' oggetto scena (new THREE.Scene()), dei componenti dei modelli protagonisti del gioco e delle interazioni con l' utente grazie alla libreria OrbitControls.

2.2 Oggetti Map e Game

Come specificato qui sopra l' oggetto Game è la colonna portante del lavoro, esso viene costruito mediante una chiamata new Game che successivamente crea la scena, la telecamera, i controlli utente e l' oggetto renderer. Associata a tale procedure vi è la creazione della mappa di gioco conclusa dalla classe Map che sfrutta la libreria THREE.PlaneGeometry per creare il campo da gioco e le relative caratteristiche materiali. Il processo seguente aggiuge gli oggetti luce (game.addLights()) di tre tipologie: un oggetto THREE.SpotLight, THREE.HemisphereLight ed THREE.DirectionalLight che garantiscono un buon effetto visivo di ombre e riflessi. L' inizializzazione del gioco si conclude con l' aggiunta delle texture e dei modelli di gioco ai quali è dedicato un paragrafo più in basso.

2.3 Textures

Una texture è un'immagine bidimensionale riprodotta su una o più facce di un modello poligonale tridimensionale. Per arricchire il gioco e renderlo di qualità abbiamo utilizzato una vasta quantità di immagini, a seconda del livello/mondo in cui ci si trova. Gli script in cui vengono gestite sono app.js (scena e mondo) e snake.js (per il drago). I formati delle immagini trattate sono tutti jpg e png. Come abbiamo visto in app.js è presente selectWorld che a seconda del suo valore permette un rendering grafico diverso. Qui abbiamo quattro diverse funzioni:

```
if (textureAttive) {
game.createRiver(5, 0, 50, -0.4, 0);
game.createFloorSx(20, 0, 50, 12.5, -0.4, 0);
game.createFloorDx(20, 0, 50, -12.5, -0.4, 0);
game.createSkyBox(); }
```

createRiver() ,createFloorSx() e createFloorDx() si occupano della pedana di gioco e in ogni mondo caricano le texture associate in questo modo: in una variabile Geometry è definito un THREE.BoxGeometry di tre dimensioni, viene chiamato il metodo applyTexture presente in utils.js in cui grazie alla libreria

THREE.ImageUtils.loadTexture è salvata la texture scelta nella rispettiva location; questa è mappata nella variabile Material grazie a THREE.MeshBasicMaterial. Infine, come appreso a lezione, viene effettuato il Mesh tra Geometry e Material e aggiunto l'output alla scena.

createSkyBox() agisce in maniera simile, ma poiché si occupa di allestire il mondo esterno, attraverso THREE.ImageUtils.loadTextureCube(urls) carica sei diverse texture (una per ogni lato interno del cubo di scena: front,back,right,left,up,down) creando una boxGeometry (cubo che fa da contorno alla scena di dimensione 4096x4096x4096).

L'oggetto aggiunto alla scena sarà quindi:

```
sky = new THREE.Mesh(new THREE.BoxGeometry(4096, 4096, 4096), skyMaterial)
```

caratterizzato da un mesh tra la Geometry descritta sopra e una variabile Material definita dalla libreria THREE.ShaderMaterial. Quindi prendendo ad esempio selectWorld = 1 (Mars level) avremo:

```
iver = "textures/mars/river.jpg";
floor = "textures/mars/floor.jpg";
directory = "textures/mars/";
satellite = "textures/mars/sat.png";
```

L'oggetto "satellite" in questo caso è un semplice rendering di un'immagine png utilizzata per arricchire la scena, con la particolarità che anche ruotando la telecamera essa si muove di conseguenza rivolgendosi sempre verso il nostro point of view: ciò è garantito da THREE.Sprite(satMaterial).

In snake.js invece ci siamo occupati di arricchire head e body del protagonista grazie alla funzione:

```
function chooseTexture() {
if (selectWorld == 0) {
// land
skinFile = 'textures/land/skin.jpg';
headFile = 'textures/land/head.jpg'; }
if (selectWorld == 1) {
// mars
skinFile = "textures/mars/skin.jpg";
```

```
headFile = "textures/mars/head.jpg"; }
if (selectWorld == 2) {
// dark
skinFile = "textures/dark/skin.jpg";
headFile = "textures/dark/head.jpg"; } }
```

A seconda di selectWorld cambierà il suo look a seconda dell'immagine jpg passata. Qui ogni texture è caricata in modo semplice così: var bodyTexture = new THREE.TextureLoader().load(skinFile); var materiale = new THREE.MeshBasicMaterial({ map: bodyTexture }); var blockMesh = new THREE.Mesh(this.blockGeometry, materiale); con i relativi castShadow e receiveShadow settati a true.

3 Modelli implementati

parte di francesco

4 Interazioni e collisioni

parte di matteo

5 Conclusioni

In conclusione in queste pagine sono state descritte le varie features del progetto proposto che ha visto la sinergia di quattro componenti ognuno dei quali ha ovviamente curato un aspetto in particolare. La scelta di misurarci in un riadattamento di un gioco arcade cosi famoso ci ha permesso di capire le vere potenzialità di questa piattaforme per campi più affini alla robotica come la simulazione di processi industriali o modellistica 3D.