Presentazione Gioco informatica 4 DSA

Presentatori : Giulia Stoppa e Mattia Pellizzaro

Shadow's Knight

Scaletta / Indice

- Presentazione del gruppo e del gioco [*Mattia*]
- Spiegazione come funziona
    - Linguaggi pruincipali : Html 3% , Css 5%, Js 91%
    - Come siamo partiti
        - Impostando una pagina semplice Html
        - Abbiamo utilizzato un framework Phaser.js
        - Tre parti
            1. Preload, vanno a caricare gli elementi grafici necessari [eseguita 1 volta]
            2. Create, creare tutte le entita del gioco e funzioni base [eseguita 1 volta]
            3. Update, aggiorna ogni fps del gioco ed esegue task di controllo [eseguita costantemente]
        - Animazioni, il danno, i movimenti vengono gestite dalle funzioni eseguite durante l'update
        - Per avere delle performance migliori i nemici vengono caricati solo quando visibili nelle visione del giocatore
        - Per poter gestire più nemici in una volta sola è l'entità del giocatore a far partire le call per le funzioni se si trova in prossimità di un nemico invece che ogni entità nemica svolga la stessa routine.
    - Creazione Mappa di gioco ed importazione in Js
        - Tiled, simile a Photoshop *Giulia*

        - Importazione in Js
    - Parti interessanti del gioco che lo rendono unico
        - Animazioni torce, che vanno in continuo
        - Animazione di Idle (Da fermo) del giocatore
        - Cursore modificato
- Raccontare la Storia / Trama del gioco
- Difficoltà e problemi
    - Spada che segue il cursore

Linguaggi utilizzati e impostazione progetto

Abbiamo utilizzato la struttura di una semplice pagina Web e con i linguaggi tradizionalmente usati nella maggior parte dei siti internet. Javascript è la protagonista nel progetto avendo oltre il 90% dei file totali interamenti dedicati alle funzioni del gioco, Css e Html sono stati utilizzati per cambiare e progettare la resa grafica del menù di gioco formata da un paio di link e titoli.

Fondamentale è stato invece il file principale che permette al gioco di "partire" una volta raggiunto l'url del sito. L' *index.html* richiama lo script principale index.js del gioco nel *body* e importa con i link nell' *head* il *framework Phaser.js* e un *component di Phaser* per gestire le vite dei personaggi.

Framework Phaser.js

Phaser è un Game Framework Javascript con una struttura interamente modulare e progettato per creare giochi web con HTML 5.

Praticamente è un insieme di librerire che forniscono funzioni di routine già pronte e testate, sono utili perchè seppur un buon programmatore sa benissimo creare un gioco da zero aiutano in quelle funzioni e pattern che si ripetono nel corso del progetto.

É completamente gratuito da utilizzare e open-source, ciò vuol dire che ognuno può contribuire al suo sviluppo. Utilizza in modo intelligente sia i Canvas che WebGl per rendirizzare il gioco, con la possibilità di utilizzare solo Cavans se il browser non supporta WebGl.

Link utili per Giulia e Matt :

- Phaser (game framework) - Wikipedia
- Welcome to Phaser 3 - Phaser3 - Phaser

Come funziona il gioco

La struttura

La struttura che ci offre Phaser per creare un gioco si basa su scene. Le scene sono macro-funzioni che vengono eseguite secondo un ordine definto dal programmatore, le tre principali e che abbiamo utilizzato anche noi sono :

1. Preload
2. Create
3. Update

Preload

É dove tutti gli asset vengono caricati. Immagini, spritesheet, tilemaps e suoni vengono caricati e assegnati a delle *namekey* che poi saranno utilizzate per riferirsi a quel determinato asset nel gioco.

Gli asset principali non sono solo immagini o elementi volti solo all'utilizzo grafico ma anche plugin esterni che vengono caricati tramite link. Noi ne abbiamo utilizzato uno chiamato AnimatedTilemamps e ci permette di animare e far ripartire in loop le animazioni che abbiamo creato per gli elementi decorativi nella mappa tipo le torce o le trappole nel terreno.

É molto importante che venga specificato correttamente il percorso dei file altrimenti Phaser non capirà dove sono gli asset e non ci permetterà di continuare ponendoci di fronte ad una schermata bianca.

Create

Create è una delle due scene più importanti, qui andiamo appunto a creare tutte le entità del gioco. É necessario capire molto bene quando dichiarare ogni elemento per ottenere il risultato desisderato, altrimenti funzioni o asseganzioni create in futuro non funzioneranno o non sarà proprio possibile crearle.

Noi abbiamo utilizzato 3 tipi di oggetti nel gioco :

- sprite

- tilemaps
- animations

Le sprite servono per associare le immagini che abbiamo importato nel Preload con un oggetto di gioco caratterizzato da un fisica di gioco, animazioni e proprietà.

La tilemaps prende un file Json e associa ogni informazione contenuta al corrispettivo layer di gioco (*Operzione che andiamo a fare noi, non è automatica*) sommandoli poi in un'unica mappa utilizzabile dal giocatore.

Le animazioni come suggerisce il termine ci permettono di creare dei loop con le immagini che importiamo, possiamo decidere la frequenza di aggiornamento, l'inizio dell'animazione e la fine e se quando la eseguiamo deve continuare a ripetersi oppure quando finisce non si ripete.

Tutto questo è possibile in modo relativamente semplice grazie alle funzioni e al sistema di denominazione degli asset con le *namekey* che Phaser ci mette a dispozione.

Update

Ultima scena, ci dà il controllo di ogni singolo fotogramma rendirizzato nel gioco. Qui tutte le funzioni che andremmo a creare avranno come obbiettivo quello di rispondere ad eveneti e chiamate ripetute, molti anche con frequenze elevate.

Nel nostro progetto lo abbiamo utilizzato per poter gestire i movimenti del giocatore, captando costantemente tutti gli input della tastiera e rispondendo con i relativi movimenti nel gioco, il sistema di danni e l'ottimizzazione delle performance in game.

É importante notare che l'ordine con cui andremo a dichiarare le variabili e richiamare le funzioni influenzerà tantissimo il risultato nel gioco, per esempio se io dico al mio giocatore di controllare le collisioni con i muri prima di aggiorare i suoi movimenti avrò constanti bug dove attraverserà i muri. Se invece io gli dico di controllare dopo aver aggiornato i suoi movimenti i bug saranno ridotti quasi a zero.

Punti principali

Gestione delle performace

Per poter avere una mappa che fosse interessante c'era la necessità di poterla riempire con molti elementi come nemici, bauli, trappole etc. Questo però ovviamente poneva dei limiti importanti sulla giocabilità da dispositivi non molto potenti e problemi non trascurabili su dispositivi più performanti, inoltre un gioco da browser non avrà mai tutte le risorse e la fluidità di un gioco nativo quindi abbiamo dovuto pensare ad una soluzione.

La soluzione che ci è venuta in mente è stata quella di aggioranre e renderizzare tutti gli elementi che fossero presenti solo all'interno della visione del giocatore. Ogni elemento al di fuori di essa non verrà renderizzato e non subirà routin di update costanti ed inutili.

Questo è gestito da un ciclo for che viene eseguito per ogni array di entità e solo se sono visibili li renderizzirà e continuerà con le task e update normali di gioco.

Nel seguente esempio il ciclo dei Goblin:

```
for (let goblinEnemy of goblinGroupEntity) {
    if (
      this.cameras.main.worldView.contains(
        goblinEnemy.goblin.x,
        goblinEnemy.goblin.y
      )
    ) {
      goblinEnemy.goblin.setActive(true);
      goblinEnemy.goblin.setVisible(true);
      if (
        Phaser.Math.Distance.BetweenPoints(player1.player, goblinEnemy.goblin) <
        240
      ) {
        eCt += 1;
        goblinEnemy.goblinChasePlayer(player1.player, goblinGroupBody);

      } else {
        goblinEnemy.goblin.body.setVelocity(0);
        goblinEnemy.goblin.anims.play("goblin-idle");
      }
    } else {
      goblinEnemy.goblin.setActive(false);
      goblinEnemy.goblin.setVisible(false);
    }
  }
```

Tante enità

Una volta risolto il problema del renderizzare troppe entità è sorto un altro problema, quello delle troppe chiamate per ogni singolo nemico.

Come funzionava un nemico, prima?

Ogni nemico aveva una suo ciclo di funzioni che venivano svolte costantemente, vuol dire che per ogni elemento dell'array `goblinGroupEntity` vi erano ulteriori cicli di funzioni per verificare se il giocatore fosse nelle vicinanze e simili. Questo portava ad avere un numero talmente grande di chiamate ed eventi che faceva chrashare il gioco.

Come funziona, ora?

Ora è molto più semplice perchè è il giocatore stesso, inteso come entità del giocatore, a controlalre se nelle sue vicinanze vi è un nemico o una serie di nemici. C'è ancora un ciclo tra tutte gli elementi di tipo nemico ma per ognuna di esse viene esguito solo un `if` che verifica se il nemico è più vicino di un certo range, se lo è attiva la funzione di movimento altrimenti resetta la sua velocità a zero e fa partire la animazione di idle.

Creazione Mappa di gioco e importazione in Js

Tiled

Per creare la mappa di gioco abbiamo utilizzato un software chiamato Tiled. É un level editor 2D che permette di create tilemaps in modo semplice ed intuitivo, ma non fa solo questo.

Ci ha permesso di decidere dove posizionare tutti gli elementi del gioco come nemici, bauli, chiavi etc tramite un sistema di cordinate dove ,una volta scelte le posizioni, ci restituiva un array di valori.

Inoltre abbiamo potuto creare delle piccole animazioni come quelle delle torce o delle trappole e poterle posizionare nella mappa in modo semplice come se fossero un blocco normale di un tileset.

Link utili per Giulia e Matt :

- Introduction — Tiled 1.8.2 documentationIntroduction — Tiled 1.8.2 documentation
- https://www.mapeditor.org/

Importazione in Js

Una volta finità mappa bisognava importarla nel gioco ma Tiled è venuto in nostro aiuto anche qui. Infatti possiamo esportare le mappe create in file Json che poi Phaser sarà in grado di leggere ed interpretare, associando ogni nome ed ID al corrispettivo valore nei tilest che gli forniamo, devono essere gli stessi usati in Tiled.

Parti interessanti

Torce

L'effetto delle torce che bruciano in loop continuamente siamo riusciti ad ottenerlo con unendo più tecniche. Per prima cosa abbiamo creato un'animazione breve ma efficace della torcia, da non risultare troppo pesante, poi l'abbiamo importata in Tiled per poterla posizionare nella

mappa ed infine in Js abbiamo usato un plugin che permette appunto di animare le Tilemaps, risparmiandoci righe e righe di codice per cui non avevamo tempo, per far andare in loop l'animazione.

Cursore modificato

Con Phaser si possono modificare vari aspetti di come un giocatore interagisce con la pagina Web mentre gioca, uno di questi è il cursore. Abbiamo cambiato il cursore con un mirino per poter rendere l'esperienza di gioco più piacevole e tolto la possibilità di utilizzare il tasto destro come normalmente siamo abituati, quindi non ci sarà nessun menù se si va a cliccarlo.

Animazione se il giocatore è fermo

Per poter gestire questa animaizio o comunque le animazioni del giocatore in generale il nostro primo metodo utilizzato è stato quello di verificare se il tasto corrispondente alla direzione della animaizione fosse premuto. Questo però si è rivelato fin dalle prime prove non la soluzione ideale, quindi abbiamo iniziato ad utilizzare la velocità sugli assi X e Y per determinare quale animazione far partire e quando su entrambi gli assi la velocità è zero l'animzaione di default è quella di Idle.

Difficoltà e problemi

Spada che segue il cursore

Per poter far si che la spada segui il cursore correttamente bisogna prima di tutto calcolare l'angolo tra il giocatore e il cursore prendendo le rispettive cordinate di ognuno e sommando a quelle del cursore quelle delle camera di gioco per poter colmare l'offset tra questa e il mondo di gioco, questo ci dà una base da cui partire.

```
this.sword.rotation = Phaser.Math.Angle.Between(
   this.player.x,
   this.player.y,
   this.game.input.mousePointer.x + this.game.cameras.main.scrollX,
   this.game.input.mousePointer.y + this.game.cameras.main.scrollY
);
```

Poi, se questo angolo che viene fornito in radianti si trova nel lato di sinistra bisongna girare la sprite della spada cambiandola con una specchiata, mentre se si trova nella parte di destra non serve cambiare e si può utilizzare la sprite normale.

Ora la spada segue il cursore correttamente e cambia quando si va da destra a sinistra rimanendo con il verso giusto. Anche la sprite del giocatore segue la spada e si ruota di

conseguenza, però non cambiando sprite ma semplicemente "flippandola" e cambiando il centro della hitbox con dei valori predefiniti decisi testando la funzione.