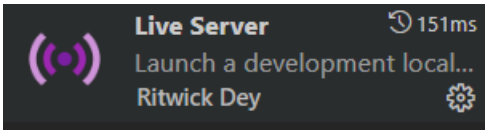
Guida Phaser 3

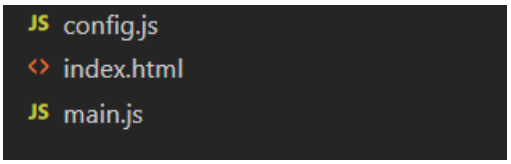
Phaser è una **libreria open source** per la creazione di videogiochi 2D desktop e mobile. Utilizzando **HTML5 Canvas** o **WebGL** per la grafica del gioco e **JavaScript** per la logica del gioco.

Configurare il sistema di sviluppo Phaser

Un gioco Phaser è necessario di essere eseguito mediante un **server**, anche un server locale. Per questo utilizziamo un'estensione del Visual Studio Code: **Live Server**



Una volta installato l'estensione, creiamo una cartella con le seguente file:



confing.js: contenente la configurazione globale e il entry point del gioco.

index.html: come tutte i codici JavaScript, anche il gioco Phaser è contenuto all'interno della pagina HTML.

main.js: la logica del gioco.

index.html

All'interno di questo file HTML importiamo Phaser come librerie:

```
<script src="//cdn.jsdelivr.net/npm/phaser@3.55.2/dist/phaser.js" ></script>
```

e il file confing.js.

Siccome l'interfaccia del gioco è contenuta in questa pagina HTML, creiamo un **contenitore**, di solito un div, e diamoli un **id**.

```
<div id="game"></div>
```

Phaser utilizzerà questo id per creare il gioco all'interno della pagina HTML.

config.js

```
import Main from "./main.js";

var config = {
    // required
    width: number,
    height: number,
    parent: string,
    physics:{
        default:"arcade",
        arcade:{
            gravity:{
                y:number
            },
            debug: true
        },

    },
    scene: [Main],
    //optional
    backgroundColor: 0xCCFFFF
};

let game = new Phaser.Game(config);
```

Tutte le **configurazioni globale** vengono mappate tramite un oggetto config e preso come **parametro** dal metodo **Phaser.Game**(config), che è **l'entry point** del gioco stesso.

Significato dei keys:

- **width**: larghezza del gioco. [number]
- **height**: altezza del gioco. [number]
- **parent**: id del contenitore precedentemente assegnato al div del index.html. [string]
- **physics**: configurazione riguardante la fisica del gioco [object literal].
    - **default,** indica quale **physic system** utilizzare, possiamo scegliere tra *arcade* e *matter,* ma utilizzeremo *arcade*.[string]
    - **arcade** [object literal]**,** configurazioni riguardante il sistema *arcade*, di cui **gravity.y** [number] è l'accelerazione verticale del gioco; il flag **debug** [boolean] ci permette di visualizzare le proprietà fisica dei singoli elementi che ha un corpo fisico all'interno del gioco.

debug = false

debug = true

- **scene**: è un array di oggetti derivati dal **Phaser.Scene,** il primo dei quali sarà il primo ad essere visualizzate. [array]
- **backgroundColor:** colore dello sfondo, opzionale, default nero.

Dopo aver assegnato tutte le configurazioni necessarie, usiamo il metodo **Phaser.Game(config)** per ottenere una istanza di **Game**, cioè il nostro gioco.

Scene

Ogni gioco Phaser ha bisogno di almeno una scena e ogni scena ha una struttura specifica:

```
class Main extends Phaser.Scene{

    constructor(){
        super("Main")
    }

    preload() {

    }

    create(){

    }

    update(time,delta){
    }

}

export default Main;
```

Prima di tutto ha un costruttore che richiama il costruttore di **Phaser.Scene** con un parametro di tipo stringa, questa stringa sarà il **nome identificativo** di questa scena, che può essere usato dalle atre scene.

Poi abbiamo tre metodi con funzionalità diverse:

- **preload()**: contiene le codice di inizializzare delle risorse che richiedono tempo come le immagini, viene chiamato prima di entrare nel gioco.
- **create()**: contiene le codice che saranno eseguite solo una volta.
- **update(time, delta)**: contiene le codice che saranno eseguite ogni *delta* di tempo.

Fin ora abbiamo fatto le configurazioni iniziare del gioco, adesso programmiamo il gioco vero e proprio.

Usare un'immagine come sfondo

Un'immagine per essere utilizzato deve sempre essere caricata **prima** di entrare nel gioco, quindi nel metodo *preload()* della **scena.**

Per caricare immagine usiamo il seguente codice:

*this.load.image(**key**, **path**);*

Il ***key*** è id dell'immagine caricato;
Il ***path*** è il percorso **relativo** o **assoluto** dell'immagine.

Caricato immagine, possiamo usarlo per creare lo sfondo; siccome lo sfondo le basta creare una sola volta, nel metodo *create()* aggiungiamo:

*this.add.image(**x**, **y**, **key**);*

x e ***y*** indicano dove l'immagine sarà esposto nel gioco.
key è key dell'imagine da usare, precedentemente definito.

Aggiungere uno sprite

Come tutti i giochi che sono composti da uno o più personaggi, lo sprite, anche in Phaser possiamo aggiungerli.

Siccome i sprite sono degli immagini, dobbiamo prima caricare nel *preload().*

Poi nel *create():*

*let sprite = this.add.sprite(**key**);*

key è key dell'immagine del sprite.

Lo sprite creato da questo metodo è **statico** cioè non gode di alcune **proprietà fisiche** (movimento, accelerazione, collisione, …), ma Phaser ci dà anche la possibilità di creare sprite dinamico:

*let sprite = this.physics.add.sprite(**x**, **y**, **key**);*

x e ***y*** sarà le coordinate dello sprite all'interno del gioco.
key è key dell'immagine dello sprite.

A volte l'immagine dello sprite è troppo grande o piccolo rispetto al grafica del gioco, per questo usiamo il metodo *scale(**x**, **y**)* dello sprite, dove ***x*** è scala orizzontale e ***y*** verticale.

Vediamo che lo sprite non si ferma al bordo inferiore del gioco; perciò, usiamo il metodo dello sprite *setCollideWorldBounds(boolean)* per fermarle al margine del gioco.

Gruppo

Non sempre tutti i sprite che aggiungiamo al gioco sono diversi tra loro, per esempio una fila di monete, per questo possiamo creare un gruppo in cui tutti gli elementi sono uguali e distanziati ugualmente tra di loro.

Esistono due tipi di gruppi, uno **statico** e uno **dinamico**, le loro differenze sono uguali allo sprite dinamico e statico.

Per creare un gruppo statico:

*let group = this.physics.add.staticGroup(**config**);*

Per creare un gruppo dinamico:

*let group = this.physics.add.group(**config**);*

Il parametro ***config*** è un object literal con delle proprietà del gruppo tra cui:

key: key dell'immagine

setXY: object literal con 4 proprietà **x** e **y** che indicano la posizione iniziale del gruppo e stepX e **stepY** che indicano rispettivamente la distanza orizzontale e verticale tra gli elementi del guppo.

Movimento di un corpo

Ogni sprite dotato di fisicità può avere una propria **velocità** o **accelerazione**.

In Phaser per dare un movimento ad uno sprite, dobbiamo prima sapere la **direzione** del vettore velocità catturando il tasto premuto dall'utente.

siccome l'utente può premere qualsiasi tasto in qualsiasi tempo, il nostro programma necessità di controlla sempre questo evento, perciò i codice di controllo scriviamo nel metodo *create()*:

- **Creiamo** il tasto:

  let key = this.input.keyboard.createCursorKeys();

  il variabile *key* conterà il tasto premuto dall'utente, il tasto può essere →[right]↓[down]←[left]↑[up] ,[shift], [space].

- **Verificare** se è premuto:

  key.[nome del tasto].isDown();

  Questo metodo ritorna vero se il tasto è stato premuto fino al rilascio del tasto.

Una volta ricevuto l'evento, possiamo dare al nostro sprite una spinta, cioè una velocità:

*sprite.setVelocity(**x**,**y**);*
*sprite.setVelocityX(**x**);*
*sprite.setVelocityY(**y**);*

x è la veocità orizzontale e ***y*** verticale

Oppure un'accelerazione:

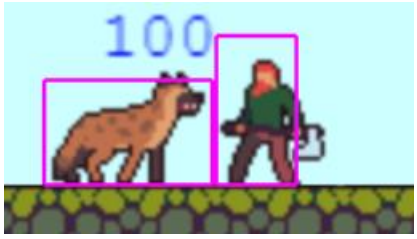
*sprite.setAcceleration(**x**,**y**);*
*sprite.setAccelerationX(**x**);*
*sprite.setAccelerationY(**y**);*

Si può vedere che una volta dato l'accelerazione o velocità il corpo non si ferma, come oggetto nel mondo reale senza attrito, per questo essiste un metodo dello sprite:

*sprite.setDrag(**x**,**y**);*
*sprite.setDragX(**x**);*
*sprite.setDragY(**y**);*

Gestire le collisioni

Collisioni sono degli **eventi** emessi quando due corpi dotati di fisicità si **incontrano** tra di loro.

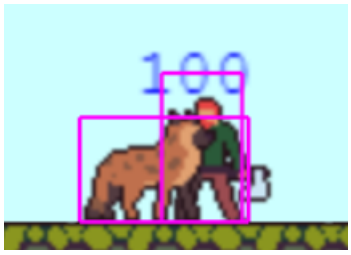


Per gestire questi eventi Phaser usa due metodi: *collider* e *overlap.*
Entrambi i metodi hanno lo stesso struttura:

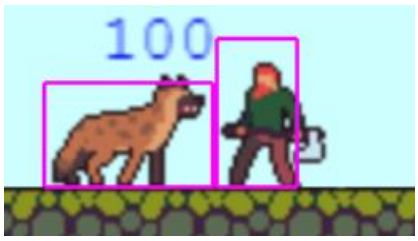
*this.physics.add.collider(**first**, **second**, **function**(**first**, **second**){});*
*this.physics.add.overlap(**first**, **second**, **function**(**first**, **second**){});*

I primi due parametri possono essere sia un array di sprite, uno sprite o un gruppo di sprite, per questo la funzione callback ha due parametri, tutti due sono oggetti realmente in collisione.

La differenza tra il metodo overlap e collider è che il primo quando si incontrano, i sprite possono sovrapporre, mentre il secondo i due sprite si fermano quando si incontrano.



overlap



collider

Aggiungere testo nel gioco

Spesso è utile mostrare agli utenti alcuni dati del gioco per questo Phaser ci dà la possibilità di creare testi all'interno del gioco.

let text = *this.add.text(**x**,**y**,**testo**,**style**);*

I parametri ***x*** e ***y*** indicano posizione del testo all'interno del gioco, mentre il parametro ***style*** è un object literal con le proprietà del testo come colore, font, altezza, larghezza…

Esempio:

let text = this.add.text(200, 200, "esempio", {color: "red", width: 20px});

Animazioni

Phaser ha delle animazioni **prestabilite** che possiamo usare per animare il nostro gioco.

Per creare queste animazioni usiamo il seguente codice:

let animazioni = this.tweens.createTimeline();

animazioni.add({

targets: [array] (array dei sprite interessati)***,***

duration: [number] (durata dell'animazione)***,***

ease: [string] (tipo di animazione)***,***

x: [number] (limite di spazio orizzontale)***,***

y: [number] (limite di spazio verticale)

});

animazioni.play();

Nella prima riga di codice creiamo l'oggetto animazione e poi con il suo metodo *add()* indichiamo quali sprite o gruppo di sprite deve eseguire un determinato animazione, infine l'animazione per essere visibile deve chiamare il metodo *play()*.

Qui si può trovare tutti i tipi di animazioni: https://rexrainbow.github.io/phaser3-rex-notes/docs/site/ease-function/.

Tilemap

Un tilemap consiste di creare l'interfaccia del gioco attraverso piccoli "tile" cioè tasselli.

Questi tile sono messi all'interno di un'immagine e hanno un'altezza e larghezza specifica che è uguale per tutti i tile.

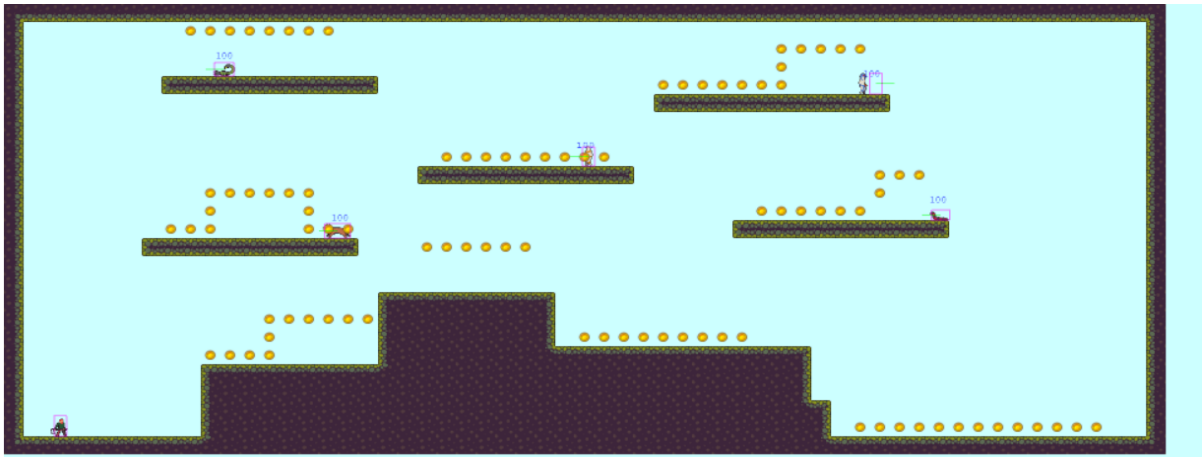
Ogni singolo tile hanno un **indice** specifica che è la loro posizione partendo da alto sinistra.

Usando tilemap non abbiamo più bisogno di creare tutti i singoli oggetti del gioco come un sprite e di controllare direttamente.

Per creare tilemap in Phaser:

*let map = this.make.tilemap(**data**, **tilewidth**, **tileheight**);*

Questo metodo crea una mappa, che viene divisa al suo interno in tanti quadratini(tile) di altezza ***tileheight*** e larghezza ***tilewidth*** e assegna ad ogni quadrato l'**indice** del tile in tileset a secondo della matrice **data*.***



*let tileImage = map.addTilesetImage(**key**);*

Questo metodo aggiunge l'immagine precedentemente caricato nel *preload()* del tileset(l'insieme di tile) alla mappa, una mappa può avere più di una tileset.

*Let layer = map.createDinamicLayer(**indice**, **tileImage**, **x**, **y**);*

Questo metodo crea il layer attraverso ***tileImage*** e la posiziona al coordinata **x** e **y**, l'indice è un id univoco del layer, siccome in una mappa può avere più di una layer.

Collisione

Per abilitare collisione dei tile dobbiamo chiamare la funzione *layer.setCollision(**tiles**),* dove ***tiles*** è un array degli indici che vogliamo abilitare la collisione.

Una volta abilitato, possiamo anche controllare collisione tra tile e un sprite attraverso la funzione *this.add.physics.collider(**sprite**, **layer**, function(**sprite**, **tile**){}),* o anche *overlap(**sprite**, **layer**, function(**sprite**, **tile**){}).*

Per sapere quale tile è in questione, usiamo la proprietà *tile.index* che ci dà l'indice del tile in collisione.