GUIDA PHASER

Audisio Nicolò I.T.I.S. G. Rivoira sez. Informatica classe 3C

Phaser è un framework, ovvero un sistema che consente di estendere le funzionalità del linguaggio di programmazione che viene usato, consentendo a chi sviluppa comandi semplici e veloci. In questo caso il framework viene applicato sul linguaggio di programmazione JavaScript, in aggiunta con HTML 5 e CSS per realizzare le versioni online dei giochi.

PREPARAZIONE

1. Creare una cartella con nome a scelta che contenga 2 sotto cartella chiamate assets (dove verranno inserite tutte le immagini) e script (dove verranno inseriti tutti i file .js del nostro gioco).
2. Aprire la cartella con un editor di testo a scelta, in questo caso viene usato Visual Studio Code.
3. All'interno della main directory creare un file "index.html" ovvero il file principale da dove partiranno tutti i collegamenti con i vari file che strutturano il nostro videogioco.
4. All'interno del file "index.html" si deve importare la struttura dell'HTML, se si è su Visual Studio Code, basta scrivere il comando "html:5" e verrà generata automaticamente tutta la struttura, che si suddivide in 2 parti, l'head e il body.
    a. All'interno dell'head troviamo:
        i. <meta charset="UTF-8"> serve a importare la codifica dei caratteri, in questo caso la codifica avviene a 8 bit;
        ii. <meta http-equiv="X-UA-Compatibile" content="IE=edge"> serve per comunicare con il brower Edge ti utilizzare la modalità più alta disponibile per la versione di IE;
        iii. <meta name="viewport" content="width=device-width, initial-scale=1.0"> serve al motore di ricercar sul quale sta girando il nostro file, per capire le proporzioni che deve usare, in questo caso sono impostate su una scala a 1, ovvero senza alcuna scala di riduzione o ingrandimento;
        iv. <titile></title> serve per dare un nome alla pagina, che si legge in alto all'interno delle schede, serve principalmente all'utente per capire in che pagina sta navigando;
        v. Questo punto è facoltativo, se non viene scritto in questa porzione dell'head viene scritto in un file apposito che prende il nome di "style.css" nel quale possiamo modificare la parte grafica del nostro sito; nel caso che lo style venga fatto all'interno del file index.html si struttura in questo modo:

```
<style>
        body{
                background-color: black; (per impostare il nero come colore di sfondo)
        }
</style>
```

    b. All'interno del body non troviamo nessuna struttura particolare già inserita, ma in questa sezione di codice inseriamo tutti i vari div e tutti i file che ci servono. Per realizzare il nostro progetto Phaser dobbiamo creare un div con il nome "thegame" o un nome che ricordiamo perché sarà dove tutto il nostro gioco avrà inizio. Es:
    c. Successivamente bisogna importare i file del framework di Phaser;
    d. Creare un file ".j" il quale conterrà tutte le impostazioni e le caratteristiche del nostro gioco;
    e. All'interno del file "index.html" bisogna aggiungere tutti i file che ci serviranno che abbiamo preparato in precedenza, il file di Phaser e la configurazione, per importare un file .js all'interno dell'HTML, si va nel body e si scrive: <script src="*percorso del file*"></script>, per esempio per importare il mio file di configurazione chiamato "script.js" contenuto nella cartella "js" devo scrivere: < script src="./js/script.js"></script>;
    f. Adesso bisogna iniziare a programmare il gioco, quindi accedere al file ".js" creato in precedenza e iniziare a scrivere il nostro gioco. Il nostro file si dovrà dividere in 4 parti.

CONFIGURAZIONE

ESEMPIO:

```
let config = {
    //configurazione
    type: Phaser.AUTO,
    width: 1200,
    height: 700,
    backgroundColor: 0x000000,
    parent: "contenitore",
    // pixelArt: true,

    //fisica
    physics: {
        default: "arcade",
        arcade: { gravity: { y: 400 }, debug: false },
    },

    scene: {

        preload: car,
        create: cre,
        update: up
    }
};
```

- All'inizio del file viene creata la variabile config, tramite il comando "let config = { }", il quale al suo interno contiene le configurazioni dello spazio di gioco (width e height), il colore di sfondo (backgroundColor), il div dell'HTML (parent) e la possibilità di pixellare il gioco (pixelArt); successivamente troviamo tutti i comandi della parte fisica del gioco, trovati all'interno di "physics: { }" dove al tuo interno troviamo il default, ovvero la tipologia di fisica che verrà inserita nel gioco e successivamente la gravità, dove è possibile modificare tramite la variabile "y" la potenza della forza di gravità e tramite il comando "debug: true" sarà possibile attivare il debug per poter vedere più nello specifico il comportamento di ogni singolo oggetto. In successione delle impostazioni inerenti alla fisica troviamo le scene, che normalmente sono 3: preload, create e update.

- Successivamente dobbiamo inserire all'interno della variabile game, la configurazione `let game = new Phaser.Game(config);`
- Adesso andiamo a caricare all'interno della scena car tutte le immagini che ci serviranno all'interno del gioco;

```
function car() {
   this.load.image('sfondo', './assets/background.png');
    this.load.image('entrata', './assets/carrello.png');
    this.load.image('uscita', './assets/portaofoglio.png');
    this.load.image('omino', './assets/tomato.png');
    this.load.image('erba', './assets/floor.png');
    this.load.image('cuori', './assets/apple.png');
}
```

Per caricare le immagini all'interno della funzione car, dobbiamo scrivere this.load.image('*nome*', '*percorso immagine*');

- Successivamente dobbiamo creare la funzione di create, dove andremmo a impostare la dimensione di ogni immagine, le collisioni e tutte le impostazioni che ci servono per gestire i nostri oggetti

```
function cre() {
    this.add.image(600, 350, 'sfondo').setScale(2);
    this.omino = this.physics.add.sprite(1000, 520, 'omino').setScale(6).setCol-
lideWorldBounds(true);
    //impostazioni erba
    this.erba = this.physics.add.staticGroup({
        key: 'erba',
        repeat: 1,
        setXY: { x: 290, y: 680, stepX: 645 },
        setScale: { x: 1.2, y: 1.5 }
    });
    //impostazioni portale entrata
    this.entrata = this.physics.add.staticGroup({
        key: 'entrata',
        setXY: { x: 130, y: 565},
        setScale: { x: 1, y: 1 }
    })
    //impostazioni portale uscita
    this.uscita = this.physics.add.staticGroup({
        key: 'uscita',
        setXY: { x: 1170, y: 200},
        setScale: { x: 2, y: 2 }
    })
    //impostazioni cuori
    this.cuori = this.physics.add.group({
        key: 'cuori',
        repeat: 5,
        setXY: { x: 300, y: 0, stepX: 100 },
        setScale: { x: 0.75, y: 0.75 }
    });
```

```
    this.erba.setDepth(10);
    this.cuori.setDepth(10);
    this.physics.add.collider(this.erba, this.omino);
    this.physics.add.collider(this.erba, this.cuori);
    this.tasto = this.input.keyboard.createCursorKeys();
    this.cuori.children.iterate(function(child) {
        child.setBounceY(Phaser.Math.FloatBetween(0.4, 0.8));
    });

    this.physics.add.collider(this.omino, this.entrata, (omino, entrata) => {
        omino.setVisible(false);
        omino.setPosition(1100, 150);
        omino.setVisible(true);
    })

    //controllo e aumento punti
    let punti = 0;
    let puntitxt = this.add.text(16, 16, 'Apple Coin: 0', { fontSize: '40px',
color: '#7300bf' });
    this.physics.add.collider(this.omino, this.cuori, (omino, cuori) => {
        cuori.disableBody(true, true);
        punti += 1;
        puntitxt.setText("Apple Coin: " + punti);
    });
}
```

- All'interno della prima riga, abbiamo importato la nostra immagine di sfondo, con tutte le dimensioni inerenti all'immagine, come larghezza, altezza e la scala;
- Nella seconda riga abbiamo importato il nostro omino, con le sue dimensioni, la scala e grazie al comando .setCollideWorldBounds(true) abbiamo attivato le collisioni tra questo oggetto e gli altri
- Successivamente abbiamo iniziato a gestire separatamente tutti gli elementi che abbiamo importato, per prima gestiamo il pavimento, dove all'inizio vicino al nome della variabile inseriamo "this.physics.add.staticGroup" per implementare la fisica di un oggetto statico, mentre all'interno delle parentesi graffe gestiamo la posizione sugli assi X e Y (setXY: {x: ,y: },), quante volte viene ripetuta l'immagine (repeat:) e la scala sugli assi X e Y (setScale: {x: ,y: },); questi paramenti li ripetiamo per gli altri oggetti presenti all'interno del gioco;
- Tramite il comando "this.nomeVariabile.setDepth(valore)" possiamo decidere la profondità che avrà l'oggetto, il valore 0 è la profondità predefinita;
- Tramite il comando "this.physics.add.collider(this.*nomeVariabile2*, this.*nomeVariabile1*);" possiamo impostare le collisioni tra due oggetti, per esempio se viene impostato tra la mela e il pavimento, dal momento che la mela dall'alto cade verso il basso, si ferma dal pavimento e non lo attraversa;
- Con "this.tasto = this.input.keyboard.createCursorKeys();" Andiamo a impostare dentro la variabile "*tasto*" l'inserimento da tastiera dell'utente, questo ci serve per poter comandare l'omino oppure interagire con il gioco;
- this.cuori.children.iterate(function(child) {
  child.setBounceY(Phaser.Math.FloatBetween(0.4, 0.8));
  }); Grazie a questo comando noi possiamo impostare il rimbalzo dei nostri cuori, secondo l'asse Y;

- Per realizzare un portale, o spostare il nostro omino da una posizione all'altra dello schermo, dobbiamo digitare il seguente comando:
  this.physics.add.collider(this.omino, this.entrata, (omino, entrata) => {
  omino.setVisible(false);
  omino.setPosition(1100, 150);
  omino.setVisible(true);
  });
  Grazie a questo comando, appena il nome omino entra in collisione con il portale d'entrata, verrà resto invisibile (omino.setVisible(false);), spostato nella posizione che vogliamo (omino.setPosition(1100,150);) e reso di nuovo visibile (omino.setVisible(true););
- Volendo se si vuole incrementare un conteggio, basta inserire il seguente comando:
  let punti = 0;
  let puntitxt = this.add.text(16, 16, 'Apple Coin: 0', { fontSize: '40px', color: '#7300bf' });
  this.physics.add.collider(this.omino, this.cuori, (omino, cuori) => {
  cuori.disableBody(true, true);
  punti += 1;
  puntitxt.setText("Apple Coin: " + punti);
  });
  Con questo comando viene creata una variabile punti (let punti=0;), viene creato il testo (let puntitxt=this.add.text(*posizioneX, posizioneY,* '*testo*', {fontSize: '30px', color: '#7300bf'}));

```
function up() {
    if (this.tasto.right.isDown) {
        this.omino.setVelocityX(300);
    } else {
        if (this.tasto.left.isDown) {
            this.omino.setVelocityX(-300);
        } else {
            if (Phaser.Input.Keyboard.JustDown(this.tasto.space)) {
                this.omino.setVelocityY(-400);
            } else {
                this.omino.setVelocityX(0);
            }
        }
    }
}
```

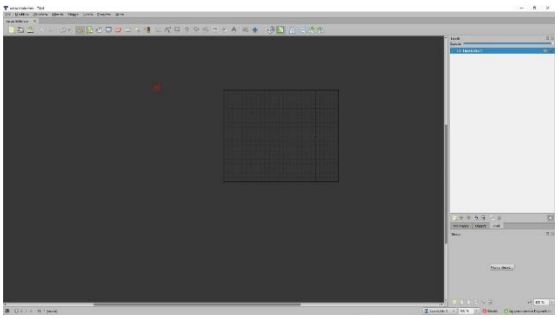
- Adesso andiamo a creare la funzione di update, come qua sopra, all'interno della quale noi andiamo a impostare la velocità, e i tasti per far muovere il nostro omino.
  Si crea un semplice if annidato dove si va a esclusione dei tasti, si inizia chiedendo al computer se il tasto premuto è il destro, se è vero l'omino di sposta sull'asse X in senso positivo, se invece è falso va a controllare se è il tasto sinistro, se è vero si sposta sull'asse X in senso negativo, se invece è falso controlla se è il tasto "space/spazio", se è vero l'omino si muove secondo l'asse Y in senso negativo (se vogliamo impostare che il tasto non può essere premuto, viene scritto il comando "Phaser.Input.Keyboard.JustDown(this.tasto.space)") , se invece è falso l'omino resterà fermo;

AGGIUNTE PER MIGLIORAMENTI

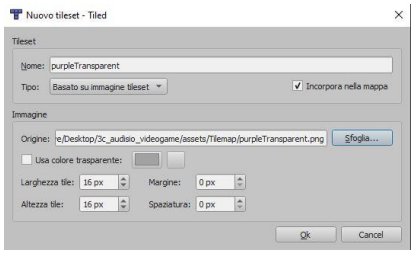
Prima abbiamo visto le impostazioni basi su come si realizza un classico gioco in phaser, però come tutto, c'è la versione base e la versione migliorata, che sembra un traguardo irraggiungibile, quando invece basta veramente poco. In seguito vedremo come realizzare mappe tramite il programma Tiled, come implementare le scene all'interno del nostro codice, come impostare le camere per seguire il nostro personaggio e dare un senso di movimento al nostro gioco.

TILED

Prima abbiamo assemblato una mappa come fosse un Lego, dove andavamo noi manualmente a inserire ogni singolo oggetto che ci serviva per realizzare la nostra mappa, però c'è un metodo molto più semplice, dove noi tramite il programma "Tiled" possiamo andare a montare la nostra mappa come voglio ci piace, andando a prendere dai nostri *Tileset* tutti i pezzettini e posizionarli dove vogliamo, potendo anche inserire le collisioni. Innanzitutto bisogna andare sul sito di Tiled, schiacciare sul tasto "Download on itch.io" e una volta scaricato l'eseguibile, avviarlo e iniziare l'installazione.



Avviato il programma, dobbiamo schiacciare su "Nuova mappa…" e impostare le dimensioni che meglio preferiamo, una volta fatto schiacciare su "ok".
All'interno del programma notiamo subito una griglia in centro allo schermo, quello sarà il nostro spazio per creare la nostra mappa.
Sopra alla griglia troviamo i soliti comandi come la gomma per cancellare, il secchiello per riempire tutto uno spazio di un unico colore, selezione rettangolare per poter selezionare più caselle insieme, i tasti per riflettere verticalmente, orizzontalmente e ruotare a destra e sinistra.
Per iniziare a creare la nostra mappa dobbiamo andare su in basso a destra dove troviamo un pulsante con scritto "Nuovo tileset…", cliccando su questo abbiamo la possibilità dando un nome e incollando il percorso del Tileset di importarlo per poter creare la mappa.
Per realizzare un tileset basta raggruppare in una cartella tutte le immagini che vogliamo inserire all'interno della nostra mappa, poi recarsi un foto editor (Photoshop oppure gimp), creare un foglio di una grandezza a nostra scelta, inserire nel foglio una griglia con i quadrati di grandezza 16x16 px e infine esportarla; N.B. un oggetto deve stare dentro a un singolo quadrato, o suddiviso tra più quadrati, perché successivamente all'interno del programma potremmo incollare sulla nostra griglia ogni singola casella realizzando così la nostra mappa.



Quando abbiamo completato la realizzazione delle nostra mappa, ci rechiamo su file>Esporta come…e dovremmo salvare il file con l'estensione ".json" all'interno della cartella con tutti gli altri file. Per importarlo all'interno del nostro gioco dovremmo inserire nel preload() il comando "this.load.tilemapTiledJSON('*nome*', '*../percorso*');" e successivamente nel create() dovremmo inserire il comando const map = this.make.tilemap({key: '*nome*'}) e al di sotto const tileset = map.addTilesetImage('*nomeTileset*', '*tiles*'), const worldLayer = map.createStaticLayer('world', tileset, 0, 0) e infine worldLayer.setCollisionByProperty({collides: true});

SCENE

Le scene all'interno di un gioco o di un qualsiasi progetto servono per suddividere il lavoro tra diversi file, ad esempio in un gioco possiamo suddividere il gioco in una scena e la parte grafica (composta da scritte e animazioni) in un'altra scena in modo da rendere più facile la lettura.
Noi per realizzare una scena, dobbiamo creare un nuovo file, salvarlo sempre con estensione ".js" e all'interno di quel file scrivere:

```
class GameScene extends Phaser.Scene {
    constructor() {
        super('GameScene')
    }
```

Per poter realizzare la scena "GameScene"; in Phaser le scene vengono create con il comando Phaser.Scene, e successivamente fuori dalle parentesi graffe tramite il create possiamo programmare la nostra scena.
Per importare una o più scene all'interno di un file, dobbiamo andare nel file dove si vuole importare la scena, e in alto scrivere:

```
import Nome from './Percorso'
```

CAMERE

Le camere all'interno di un gioco servono per dare un senso di movimento al gioco e per seguire il personaggio.
Per aggiungere una camera al nostro gioco dobbiamo inserire: game.camera.follow(player); come primo comando per far capire al nostro computer che deve seguire il nostro personaggio e successivamente inseriamo: game.debug.cameraInfo(game.camera, 32, 32); per impostare le dimensioni della nostra camera, ovviamente più la camera sarà piccola meno cose vediamo attorno al personaggio, al contrario invece se è troppo grande si rischia di vedere solamente la mappa e perdere il personaggio.