INFORMATICA

Appunti di LCG

di informatica applicata

A.A. 2023/24

JAVASCRIPT BASI

• VARIABILI

① `let heigh = 10;`
↳ modificabile
Es. `heigh = 15;`

② `const facade color = "#FF0000";`
↳ non cambia valore

• Può essere locale o globale
↳ `let totale = 0;` → globale

```
function Area (base, height) {
    let area; → locale
    crea = base * height;
    totale += area;
    return area;
}
Area (5,5);
Console.log(totale); //Stampa 25
```

• Attenzione!

```
let totale = 0; globale
function Area (base, height) {
    let area; → locale
    crea = base * height;
    let totale = 0
    totale += area; // Scrive 25
    return area;
}
Compute_area(5,5);
Console.log (totale); // Stampa 0
```

• FUNZIONI = + istruzioni insieme e (es. // Dichiarazioni) — poi la chiamo

```
function compute_area (base, height) {
let area = base * height;
return area;
}
// Invocazione
const area1 = compute_area (125, 88);
const area2 = compute_area (1925, 42);
```

• TIPI DI DATO

① Primitivi
- Boolean = true - false
- Number = interi e con virgola (es. 3.14)
- String = sequenza di caratteri (es. "ciao pippo")
- Null = assenza di valore
- Undefined = assenza di variabile

② Oggetti = es. Array

```
const friends = ["Marco", "Elena", "Claudia", "Pino"];
const amico1 = friends [0]; → indice
const amico2 = friends [1];
console.log (amico1 + "and" + amico2)
```
↳ Marco e Elena

• OGGETTO

```
let car = {
    brand: "Toyota",   → attributo
    model: "Yaris",
    year: 2020
};
```
The car: Toyota Yaris (2020)

```
console.log("The car: " + car.brand + " " + car.model +
"(" + car.year + ")";
```

• Cambiare un valore

```
let car = {
    brand: "Toyota",
    model: "Yaris",
    year: 2020
};
car.year = 2016;
```

• Al posto del punto anche
↳ `car ['brand']` (chiave)

• Proprietà dinamiche

```
let car = {
    brand: "Toyota",
    model: "Yaris",
    year: 2020
};
car.color = "black"
```

• Metodi = funzioni

```
    ...
    year: 2020
    showInEo: function() {
        console.log (this.brand...);
    }
};
car.showInEo();
```

• OPERATORI MATEMATICI

a = a + 5 ↔ a += 5

Binari:
- + Somma
- − Differenza
- * Moltiplicazione
- / Divisione
- % Resto divisione intera
- ** Elevamento a potenza

Unari:
- ++a Pre-incremento
- a++ Post-incremento
- −a Pre-decremento
- a−− Post-decremento

• OPERATORI DI CONFRONTO

- == Uguale
- != Diverso
- === Strettamente Uguale
- !== Strettamente Diverso
- > e >= Maggiore e Maggiore Uguale
- < e <= Minore e Minore Uguale

5 == 5 e 5 === 5 sono veri
5 == "5" vero
5 === "5" falso
5 !== "5" vero

• OPERATORI LOGICI = x i booleani

- && And = entrambi operatori veri
- || Or = almeno uno falso
- ! Not = inverte il valore dell'operando

• COSTRUTTI CONDIZIONALI

IF - ELSE = 2 strade (else si può omettere)

es.↳
```
const a = 7.367;
const b = 6618;
if (a > b) {
    console.log ("a > b!");
}
else {
    console.log ("b >= a!");
}
```

• COSTRUTTI ITERATIVI

- Ciclo for

```
const friend ["Marco", "Elena", "Ugo", "Lia"]
for (let i = 0; i < friend.length; i++){
    console.log (friend[i]);
}
```
i = variabile locale

- Do...While = esegue istruzioni almeno 1 volta
- While = può non fare nemmeno 1 volta
- For...in, for...of = stampa anche se non ci sono elementi

- PASS - BY - VALUE

```
function modifica (x){
  x = x + 10;
}
let valore = 5;
modifica(valore);
console.log (valore); // Stampa 5
```

- Eccezione

```
function modifica(x) {
   x.elemento = x.elemento + 10;
}
let valore = {elemento:5}; → oggetto
modifica (valore);
console.log (valore.elemento); // Stampa 15
```

```
function modifica(x) {
     x = {elemento: 15};
}
let valore = {elemento:5}; → oggetto
modifica (valore);
console.log (valore.elemento); // Stampa 5
```

POLIPHASER

STRUTTURA DI UN PROGETTO

```
nome-progetto → cartella principale
├ assets → risorse tipo immagine, audio...
│  └ ...
├ index.html → il file HTML che verrà aperto nel browser
├ lib → risorse javascript aggiuntive (librerie)
│  └ ...
│     └ phaser.min.js
│     polyphaser.js
└ src
   ├ main.js → i sorgenti javascript
   └ ...
```

main.js → istanziare il gioco

```
const config = {
    canvas_width: 1280,
    canvas_height: 800,
    canvas_id: "game_area",
    background_color: 0x0000FF,
    debug_mode: true,
    gravity_value: 20,
};
PP.game.create(config);
```

PP = namespace che contiene funzioni di PolyPhaser

(canvas_width/height: l e A in pixel; canvas_id: ID del tag che conterrà il gioco; background_color: colore esadecimale)

```
<!DOCTYPE HTML>
<html lang="en">
<head>
    <title>NOME PROGETTO</title>
    <script src="lib/phaser.min.js"></script>
    <script src="lib/polyphaser.js"></script>
    <script src="src/scene/scene1.js" type="module"></script>
    <script src="src/scene/scene2.js" type="module"></script>
    <script defer src="src/main.js"></script>
</head>
<body>
    <div id="game_area" style="text-align:center;"></div>
</body>
</html>
```

(in modo di scene multiple non vadano in conflitto con nomi di funzioni usuali; prima scena è quella che viene avviata x prima. Mentre l'ordine delle altre non conta)

(defer → script eseguito dopo di tutto resto della pagina sarà caricato)

LA SCENA (sfondo-elementi interattivi, personaggi, GUI-menù in sovrapposizione)

```
function preload(s) {
}
```
→ parametro spazio; invocato 1 volta, caricamento risorse multimediali

```
function create(s) {
}
```
→ invocato 1 volta, aggiunge elementi alla scena

```
function update(s) {
}
```
→ ciclo infinito, istruzioni comportamenti elementi di scena

```
function destroy(s) {
}
```
→ ultimo metodo invocato x rimuovere elementi e distruggere scena

```
PP.scene.add("nome_scena", preload, create, update, destroy);
```
→ nome univoco x richiamare la scena (es. cambi scene)

(scene1.js)

CARICAMENTO IMMAGINE (preload)

```
let img = PP.assets.image.load(s, "assets/images/img.png");
```

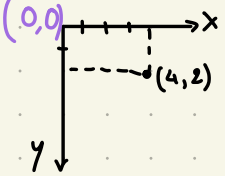
variabile che contiene oggetto che rappresenta l'immagine; oggetto spazio; percorso del file

AGGIUNGERE ELEMENTO ALLA SCENA (create)

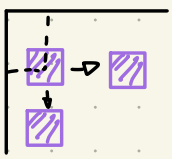
```
let player = PP.assets.image.add(s, img, pos_x, pos_y, pivot_x, pivot_y);
```

oggetto in variabile x gestirlo poi; oggetto che rappresenta l'immagine; coordinate; pivot (centro di rotazione)

COORDINATE NELLO SPAZIO



COORDINATE E MOVIMENTO (update)



```
image_quadrato.geometry.x += 90;
image_quadrato.geometry.y += 70;
```
in pixel

PROPRIETÀ IMMAGINI

```
image_quadrato.geometry.scale_x = 2   → raddoppia
image_quadrato.geometry.scale_y = 2

image_quadrato.geometry.flip_x = true   → ribalta
image_quadrato.geometry.flip_y = false

image_quadrato.geometry.angle = value;
image_quadrato.geometry.angle += value;
```
ruota

INPUT DA TASTIERA (update)

```
PP.interactive.kb.<metodo-specifico>
PP.key_codes.UP   (oppure DOWN, LEFT, RIGHT, A, SPACE ...)

if (PP.interactive.kb.is_key_down(s, PP.key_codes.LEFT)) {
    ...
}
```

- ANIMAZIONE E SPRITESHEET
  - Insieme di immagini con pose movimento
  - Pose disposte in matrice di frame
  - Animazione = sequenza di frame
  - Velocità = frame al secondo
  - Anche x effetti
  - In preload ↓
    - Pet ss_player = PP.assets.sprite.load_spritesheet (s, "assets/images/player.png", 200, 100);
      P, R
  - In create ↓
    - Player = PP.assets.add (s, ss_player, 100, 200, 0.5, 1);
      Immagine spritesheet · Posizione iniziale x · Pos. in z y · Pivot x · Pivot y
    - Anche in funzione a parte o create ↓
      - PP.assets.sprite.animation_add (player, "run", 0, 13, 8, -1);
        sprite · nome animazione · frame di partenza · frame arrivo · velocità · n. ripetizioni -1=∞, 0=finita
      - PP.assets.sprite.animation_add_list (player, "run", [0,1,4,20], 8, -1);
        sprite · nome animazione · lista frame · velocità · numero ripetizione
      - PP.assets.sprite.animation.play (player, "run");

- TILE SPRITE = riempire spazio e dare effetto movimento
  - Lo pattern ripetitivo
  - Si può caricando classicamente l'immagine
  - In create ↓
    Pet ts_clouds = PP.assets.tilesprite.add (s, img_clouds, 0, 0, 1280, 800, 0, 0);
    x y l h pivot x pivot y
  - In update ↓ movimento
    ts_cloud.tile_geometry.x += 1

- PARALLAX SCROLLING
  - Profondità
  - Carico le varie immagini (preload)
  - Poi aggiungo la tilesprite (create
    Pet background = PP.assets.tilesprite.add (s, img_background, 0, 0, 1280, 800, 0, 0);
    background.tile_geometry.scroll_factor_x = 0;
    background.tile_geometry.scroll_factor_y = 0;
    Lo così x tutti i livelli
  - Poi impostiamo la camera che segue il giocatore
    Lo PP.camera.start_Follow (s, player, 0, 250);
    offset rispetto asse x eg
  - Ora mettiamo movimento = update
    Lo background1.tile_geometry.x = PP.camera.get_scroll_x (s) * speed
    + vicino = + veloce
    ↑ ottengo valore corrente della posizione della camera

- UN GIOCO, MOLTE SCENE (sfondo, personaggio, menu...)
  - PP.scenes.start ("scene_name")
    ↳ salto al fondo delle varie scene create

- STATO DEL GIOCO = tenere traccia avanzamento
  - PP.gameState.set_Variable ("variable_name", <valore da salvare>);
  - PP.gameState.get_Variable ("variable_name");

- GUI / HUD
  - PP.shapes.text_add (scene, x, y, "text to insert");
    ogg. che rappresenta l'immagine · coordinate nello spazio · testo
  - PP.shapes.text_styled_add (s, x, y, "text", size, family, style, color, background_color);
    colore testo · colore sfondo · ogg. che rappresenta immagine · dimensione in pixel · del FONT
  - PP.shapes.changeTex (texObject, "New text");
    es. n. monete

- FORME
  - Rettangolo
    PP.shapes.rectangle_add (scene, x, y, width, height, color, alpha);
    0 = trasparente, 1 = opaco
  - Archi di circonferenza
    PP.shapes.arc_add (scene, x, y, radius, start, end, antiClockwise, color, alpha);
    raggio · Angolo di inizio e fine · direzione rotazione
  - Poligono
    PP.shapes.polygon_add (scene, x, y, points, color, alpha);
    Array di coordinate dei vertici dei lati

- PULSANTI
  - Inserisco immagine (load - add)
  - In create aggiungo ↓
    PP.interactive.mouse.add (clicked_object, "type_of_event", <name_of_a_function>);
  - Input da mouse: "pointerdown" = click su elemento di gioco
    "pointerdownoutside" = click fuori area di gioco
    "pointermove" = puntatore del mouse si muove
    "pointerout" = puntatore mouse esce da ogg. interattivo
    "pointerover" = puntatore si muove sopra ogg. inter.
    "pointerup" = tasto mouse viene rilasciato
    "pointerupoutside" = tasto rilasciato fuori area gioco
    "wheel" = rotellina rotella
  - ATTENZIONE! se telecamera segue giocatore anche elementi qui
    <GUI_images>.tile_geometry.scroll_factor_x = 0
    <GUI_images>.tile_geometry.scroll_factor_y = 0

- TEMPO
  <variabile-del tempo-corrente> = PP.timers.getTime();
  PP.timers.addTimer (scena, delay, function_name, is_Loop);
  durata in millisecondi · nome funzione chiama alla termine · all termine deve ripetere?

FISICA (gravità)

- Main = parametro gravità
  - accelerazione di gravità in px/sec^2 all'asse y
- Elementi soggetti alla fisica = aggiunti alla fisica della scena (ovviamente aggiunti alla scena stessa

create → `player = pp. ... add( ... );`

`pp.physics.add (s, player, pp.physics.type. DYNAMIC);`

- s → oggetto scena
- player → oggetto da aggiungere alla fisica
- DYNAMIC → tipologia di interazione (STANC / DYNAMIC)
  - STANC → elementi fisici scena ma che devono interagire con gli altri
  - DYNAMIC → elementi mobili soggetti a forze

Movimento

```
pp.physics.set_velocity_x(player, 100);
pp.physics.set_velocity_y(player, -300);
```
} velocità lungo gli assi

Collisioni

Es. con confini mondo di gioco (x,y se no chiusi cadono)

create → `pp.physics.set_collide_world_bounds(player, true);`

Es. con un "pavimento"

→ creiamo un rettangolo e lo aggiungiamo alla fisica come statico con → `pp.physics.add_collider(s, player, floor);`

Es. funzione di callback

```
function collider_test (s, obj1, obj2){
    console.log("collisione con box")
}
function create(s){
    pp.physics.add_collider_E (s, player, box, collider_test);
}
```

- **Sovrapposizioni** = es. collezionare funghetti

→ `pp.physics.add_overlap (s, player, mushroom, overlap_function);`

- overlap_function → callback con determinate istruzioni

- **Rimbalzi** = alla collisione

→ `pp.physics.set_bounce_y (player, 0.5)`

- 0.5 → valore tra 0 e 1 dove 1 è rimbalzo costante

LAYER

- z-index = posizione oggetto nell'asse virtuale z che inizia da fondo schermo fronte giocatore

- Di default gli oggetti z = 0
- Vengono visualizzati nell'ordine di aggiunta

- Si possono creare livelli con
  - Uno z-index
  - Un valore di opacità (alpha)
  - Il valore hidden

create →

```
let nome_livello = pp.layer.create(0);
// ora posso aggiungere elementi
pp.layer.add_to_layer(nome_layer, rec2);
pp.layer.add_to_layer(nome_layer, rec1);
```

- ordine di aggiunta = ordine di visualizzazione (non conta ordine di creazione)

- A ogni layer possiamo assegnare uno z-index =

```
pp.layer.set_z_index(layer_background, 1);
pp.layer.set_z_index(layer_foreground, 5);
```

VISIBILITY

- <oggetto>.visibility.alpha → da 0 a 1 (trasparenza)
- <oggetto>.visibility.hidden → true / false (nascosto o meno)

- Se un oggetto ha un alpha di 0.5 ed è in un layer di alpha 0.6 allora l'oggetto sarà trasparente secondo

$0.5 * 0.6 = 0.3$

Portare anche quaderno e vecchie prove d'esame!