Obiettivi Monete e Punti

Ogni gioco ha bisogno di uno scopo, semplice o complicato che sia, altrimenti non ha senso di esistere. Per approfondire questi aspetti di progettazione, è utile leggere la Guida Game Design.

Poiché non ci basta sconfiggere dei temibilissimi ombrellini volanti, introduciamo la possibilità di raccogliere delle monete per accumulare un punteggio e arrivare ad un portale di uscita, che ci permetterà di accedere al livello successivo.

Creiamo quindi un nuovo file Coin.js contenete "la classe" ***Coin*** per le monete, che in realtà è una ***function*** che eredita da ***GameObj***:

```
function Coin(x, y) {
    // costruttore
    GameObj.call(this, x, y);
    // sprite e metodo draw
    this.SetSprite(game.sprCoin);
    // centra offset dello sprite
    this.OffsetCenter();
    // ferma l'animazione
    this.animSpeed = 0;
    this.hit = false;
    // variabile per l' ondeggiamento
    this.boing = Math.random() * 3;
    this.bbox = new BoundingBox(x-this.xOffset, y-this.yOffset, this.sprite.w,
this.sprite.height);
    this.Update = function () {
        // se la moneta ? stata presa, va verso l'alto e svanisce
        if (this.hit) {
            this.y-=10;
            if (this.life-- <= 0) {
                this.Destroy();
            }
        } else {
            // la moneta ondeggia nell'aria
            this.boing += 0.1;
            this.y += Math.cos(this.boing) / 2;
            if(this.bbox.Collides(game.player.bbox)) {
                //se collide col player, inizia a ruotare e va verso l'alto
                this.life = 0;
                this.hit = true;
                this.animSpeed = 0.5;
                //aggiunge punteggio
                game.score++;
                //avvia il suono della moneta raccolta
                AudioPlay(game.sndCoin);
            }
        }
    }
}
// eredita da GameObj
Eredita(Coin);
```

Carichiamo lo ***sprite*** della moneta e aggiungiamo al gioco anche una musica di sfondo e gli effetti sonori per relativi alla raccolta delle monete: 

```
// dentro Game nella funzione Init
this.sprCoin = this.gr.LoadSprite("immagini/coin.png", 4);
//caricamento suoni
  this.sndCoin = this.gr.LoadSound("audio/coin.mp3");

// ...
```

Nel file Utils.js inseriamo in fondo questa nuova funzione per riprodurre dei suoni:

```
function AudioPlay(source) {
    if (window.ActiveXObject != undefined) {
        source.pause();
        source.currentTime = 0;
    } else {
        source.load();
    }
    source.play();
}
```

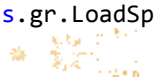
Oltre agli ombrelli, nel livello caricato ci sono anche altri oggetti, tra cui le monete. In ***LoadLevel***, aggiungiamo un altro case, per istanziare i nostri "**Coin**":

```
// dati sugli oggetti
var objects = levels[lev][3];
for(var i = 0; i < objects.length; i++) {
    //...
    switch(...) {
        case "umbrella":
            //...
        case "coin":
            new Coin(x+this.cellSize/2, y+this.cellSize/2);
            this.coinsCount++;
            break;
    }
}
```

Importiamo il file Coin.js.in Index.html. Proviamo a far partire il gioco, se tutto è andato bene, dovremmo avere un pò di monete da raccogliere.

Aggiungiamo qualche abilità al player ad esempio facciamo in modo che possa sparare ai nemici. Per prima cosa ovviamente carichiamo gli sprites necessari nella funzione Init:

```
this.sprBullet = this.gr.LoadSprite("immagini/bullet.png", 1);
this.sprBulletHit = this.gr.LoadSprite("immagini/bullethit.png", 3);
```

bullet:  bulletthit: 

Creiamo un nuovo file Bullet.js che conterrà l'oggetto proiettile e ovviamente lo importiamo in index.html:

```
function Bullet(x, y, hspeed) {
    //constructor di GameObj
    GameObj.call(this, x, y);
    //imposta sprite e metodo Draw
    this.SetSprite(game.sprBullet);
    this.hSpeed = hspeed;
    //centra l'offset dello sprite
    this.OffsetCenter();
    //crea il bounding box
    this.bbox = new BoundingBox(x - this.xOffset, y - this.yOffset, this.sprite.w,
this.sprite.height);
    this.Update = function () {
        // se collide con un blocco, si distrugge
        if (this.GetCollision(game.blocks, this.hSpeed / 2, 0)) {
            this.Destroy();
        }
        //se collide con un nemico, gli riduce la vita
        //e il proiettile viene distrutto
        if (inst = this.GetCollision(game.nemici, this.hSpeed / 2, 0)) {
            inst.life--;
            this.Destroy();
        }
        this.x += this.hSpeed;
        this.bbox.Move(this.x - game.sprBullet.width/2, this.y - game.sprBullet.height/2);
        //se il proiettile esce dalla visuale, viene distrutto
        if (this.x < 0 || this.x > game.AreaW * game.cellSize) {
            this.Destroy();
        }
```

```
    }
    //quando viene distrutto
    this.OnDestroy = function () {
        //viene cancellato dall'array bullets
        game.bullets.splice(game.bullets.indexOf(this), 1);
        //crea una piccola esplosione
        new BulletExplosion(this.x, this.y);
    }
    game.bullets.push(this);
}
//eredita da GameObj
Eredita(Bullet);
```

Inizializziamo l'array bullets all'interno di ResetLevel

```
//...
this.bullets = [];
//...
```

Definiamo anche ***BulletExplosion***, una semplice animazione che avviene quando il proiettile si distrugge e per non creare troppi file questo oggetto lo inseriamo in Bullet.js:

```
//oggetto che visualizza l'esplosione del proiettile
function BulletExplosion(x, y) {
    //constructor
    GameObj.call(this, x, y);
    //avvia un suono
    AudioPlay(game.sndHit);
    //imposta sprite e funzione di draw
    this.SetSprite(game.sprBulletHit);
    //imposta l'offset al centro
    this.OffsetCenter();
    //velocità di animazione
    this.animSpeed = 0.25;
    //distruggi al termine dell'animazione
    this.OnAnimationEnd = function () {
        this.Destroy();
    }
}
//eredita da gameobject
Eredita(BulletExplosion);
```

aggiungiamo a questo file anche un oggetto esplosione che visualizzeremo quando distruggiamo un nemico:

```
function Explosion(x,y){
       GameObj.call(this, x, y);
       AudioPlay(game.sndHit);
       this.SetSprite(game.sprExplosion);
       this.animSpeed = 0.2;
       this.OffsetCenter();

       this.OnAnimationEnd = function(){
             this.Destroy();
       }
}
Eredita(Explosion);
```

aggiungiamo al file Umbrella. Js la funzionalità per visualizzare l'esplosione quando viene distrutta modificando la funzione onDestroy come segue::

```
//quando viene distrutto, cancella dall'array dei nemici
    this.OnDestroy = function () {
        new Explosion(this.x, this.y);
        game.nemici.splice(game.nemici.indexOf(this), 1);

    }
```

Prima di andare avanti, dobbiamo caricare un paio di suoni, tra cui ***sndHit***, utilizzato nel codice precedente.

Inseriamo in Init di main.js il codice seguente:

```
this.sndHit = this.gr.LoadSound("audio/hit.mp3");
```

```
this.sndShot = this.gr.LoadSound("audio/shot.mp3");
```
Possiamo quindi far sparare il nostro player quando viene premuto il tasto spazio. Come prima cosa aggiungiamo alle variabile del player due nuove variabili per gestire gli spari:

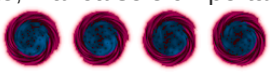
```
//variabile per gestire gli spari
             this.shotTime = 0;
          this.canShot = true;
```
poi in fondo al metodo Update del player subito dopo le due istruzioni:

```
game.viewX =  Math.floor(Math.lerp(game.viewX, targetX, 0.2));
game.viewY =  Math.floor(Math.lerp(game.viewY, targetY, 0.2));
```

aggiungiamo il codice per la gestione degli spari ad intervallic prestabiliti:

```
//se è possibile sparare
if (this.canShot) {
 if (Inputs.GetKeyPress(KEY_SPACE)) {
  var bullet = new Bullet(this.x + 10 * this.flip, this.y - this.height / 2, this.flip * 4);
        //avvia il suono di sparo
        AudioPlay(game.sndShot);
        //imposta una pausa di qualche frazione di secondo,
        //prima di poter sparare un altro colpo
        this.canShot = false;
        this.shotTime = 10;
    }
}
else {
    //timer di attesa per sparare il prossimo colpo
    this.shotTime--;
    if (this.shotTime <= 0) {
        this.canShot = true;
    }
    //cambia sprite in base allo status corrente
    //se salta, imposta lo sprite di sparo durante il salto
    //se è fermo, imposta lo sprite fermo in modalità sparo ecc..
    switch (this.sprite) {
        case game.sprPlayerRun:
            this.sprite = game.sprPlayerIdleShot;
            this.curFrame = 0;
            break;
        case game.sprPlayerIdle:
            this.sprite = game.sprPlayerIdleShot;
            this.curFrame = 0;
            break;
        case game.sprPlayerFall:
            this.sprite = game.sprPlayerFallShot;
            break;
        case game.sprPlayerJump:
            this.sprite = game.sprPlayerJumpShot; break;
}
}
```

Adesso possiamo provare se il tutto funziona.

A questo punto, manca solo un portale che ci condurrà al livello successivo. Come al solito, carichiamo l'immagine :  nella funzione Init ddell'oggetto game:

```
this.sprPortal = this.gr.LoadSprite("immagini/portale.png", 4);
```
Creiamo un nuvo file javascript di nome portale . js nel quale definiamo l'oggetto ***Portale***:

```
function Portale(x, y) {
    // costruttore
    GameObj.call(this, x, y);
    //imposto sprite e funzione Draw
    this.SetSprite(game.sprPortal);
    //velocità animazione
    this.animSpeed = 0.25;
```

```
    //offset al centro dello sprite
    this.OffsetCenter();
    this.bbox = new BoundingBox(x - this.xOffset, y - this.yOffset, this.sprite.w,
this.sprite.height);
    this.Update = function () {
        //se collide con il player, vai al livello successivo
        if (this.bbox.Collides(game.player.bbox)) {
            game.LoadLevel(game.level + 1);
        }
    }
}
Eredita(Portale);
```

Dentro ***LoadLevel***, aggiungiamo un case allo ***switch*** per istanziare il portale d'uscita:

```
//dati sugli oggetti
//...
for (var i = 0; i < objects.length; i++) {
    //...
    switch (objects[i][2]) {
        //...
        case "portale":
            new Portale(x, y);
            break;
    }
}
```

Allo stesso modo, **aggiungiamo il punto di ingresso**, caricando prima lo ***sprite***:

```
this.sprIngresso= this.gr.LoadSprite("immagini/ingresso.png", 3);
```

Per semplicità aggiungiamo al file portale.js anche la definizione dell'oggetto ***Ingresso:***

```
function Ingresso(x, y) {
    GameObj.call(this, x, y);
    this.SetSprite(game. sprIngresso);
    this.animSpeed = 0.25;
}
Eredita(Ingresso);
```

Dentro ***LoadLevel***, aggiungiamo un case allo ***switch*** per istanziare il portale d'uscita

```
// dati sugli oggetti
// ...
for (var i = 0; i < objects.length; i++) {
    // ...
    switch (objects[i][2]) {
        // ...
        case "inizio":
            //istanzia un oggetto spawn
            var inizio = new Ingresso(x, y);
            //imposta il punto di ingresso
            this.inizioX = x + this.cellSize / 2;
            this.inizioY = y - this.cellSize;
            break;
    }
}
```

Per prima cosa spostiamo l'struzione dell'istanza del player: `this.player = new Player();`
alla fine di LoadLevel

Cambiamo il codice per il posizionamento iniziale all'interno dell'oggetto Player

```
function Player(){
       // imposto le coordinate di partenza
       this.xStart = game.inizioX;
       this.yStart = game.inizioY;
```

Se proviamo a mandare in esecuzione dovremmo avere il player sotto il lampioncino, e alla fine della mappa sulla destra ci dovrebbe essere il portale per il passaggio al livello successivo.

Display e informazioni sullo stato del gioco

Tra le ultime cose che ci rimangono da fare, c'è l'aggiunta di un Display che ci mostrerà quante vite abbiamo, il punteggio relativo alle monete raccolte , dei tasti per il menu di pausa e fullscreen.

Innanzitutto carichiamo gli sprites necessari:

```
this.sprLife = this.gr.LoadSprite("immagini/life.png", 1);
this.sprLifeLost = this.gr.LoadSprite("immagini /lifeLost.png", 1);
this.sprPause = this.gr.LoadSprite("immagini /pause.png", 1);
this.sprResume = this.gr.LoadSprite("immagini /resume.png", 1);
```

All'interno del file display.js aggiungiamo un nuovo oggetto Display:

```
function Display() {
    this.Draw = function () {
        //disegna il numero di vite disponibili
        for (var i = 0; i < 5; i++) {
            if (game.player.lives > i)
                game.ctx.drawImage(game.sprLife, 70 + i * 50, game.canvas.height - 60);
            else
                game.ctx.drawImage(game.sprLifeLost, 70 + i * 50, game.canvas.height - 60);
        }
        //variabili per i bottoni
             buttonX = 10,
              buttonY = game.canvas.height - 60,
              buttonW = game.sprPause.width,
              buttonH = game.sprPause.height;
        //Se il gioco è in pausa, disegna il menu
        if (game.paused) {
            game.ctx.drawImage(game.sprResume, buttonX, buttonY);
            //menu di pausa
            game.ctx.shadowColor = "#000";
            game.ctx.shadowOffsetX = 1;
            game.ctx.font = "38pt 'Arial'"
            game.ctx.textAlign = "center";
            game.ctx.shadowBlur = 3;
            var cx = game.canvas.width / 2;
            var cy = game.canvas.height / 2;
            if (Inputs.MouseInsideText("Resume", cx, cy, "#eee", "#ea4") &&
Inputs.GetMousePress(MOUSE_LEFT)) {
                game.paused = false;
            }
            //se clicco main menu
            if (Inputs.MouseInsideText("Main Menu", cx, cy + 60, "#eee", "#ea4") &&
Inputs.GetMousePress(MOUSE_LEFT)) {
                //carica il menu principale
                game.LoadLevel(0);
            }
            //resetta l'ombra del testo e il centramento
            game.ctx.shadowOffsetX = 0;
            game.ctx.shadowBlur = 0;
            game.ctx.textAlign = "start";
        }
              else {
                   game.ctx.drawImage(game.sprPause, buttonX, buttonY);
                     }
            if (Inputs.GetMousePress(MOUSE_LEFT)) {
                //se premo il bottone di pausa
                if (Inputs.MouseInsideRect(buttonX, buttonY, buttonW, buttonH)) {
                    game.paused = !game.paused;

            }
        }
        //Disegna il punteggio e il livello corrente
        game.ctx.fillStyle = "#fff";
        game.ctx.shadowColor = "#000";
        game.ctx.shadowOffsetX = 1;
```

```
        game.ctx.font = "24pt 'PixelFont'"
        game.ctx.shadowBlur = 3;
        game.ctx.fillText("Punti: " + game.score, 500, game.canvas.height - 20);
        game.ctx.fillText("Livello: " + game.level, 670, game.canvas.height - 20);
        game.ctx.shadowOffsetX = 0;
        game.ctx.shadowBlur = 0;
    }
}
```

In ***ResetLevel*** aggiungiamo:

```
this.display = null;
this.paused = false;
```

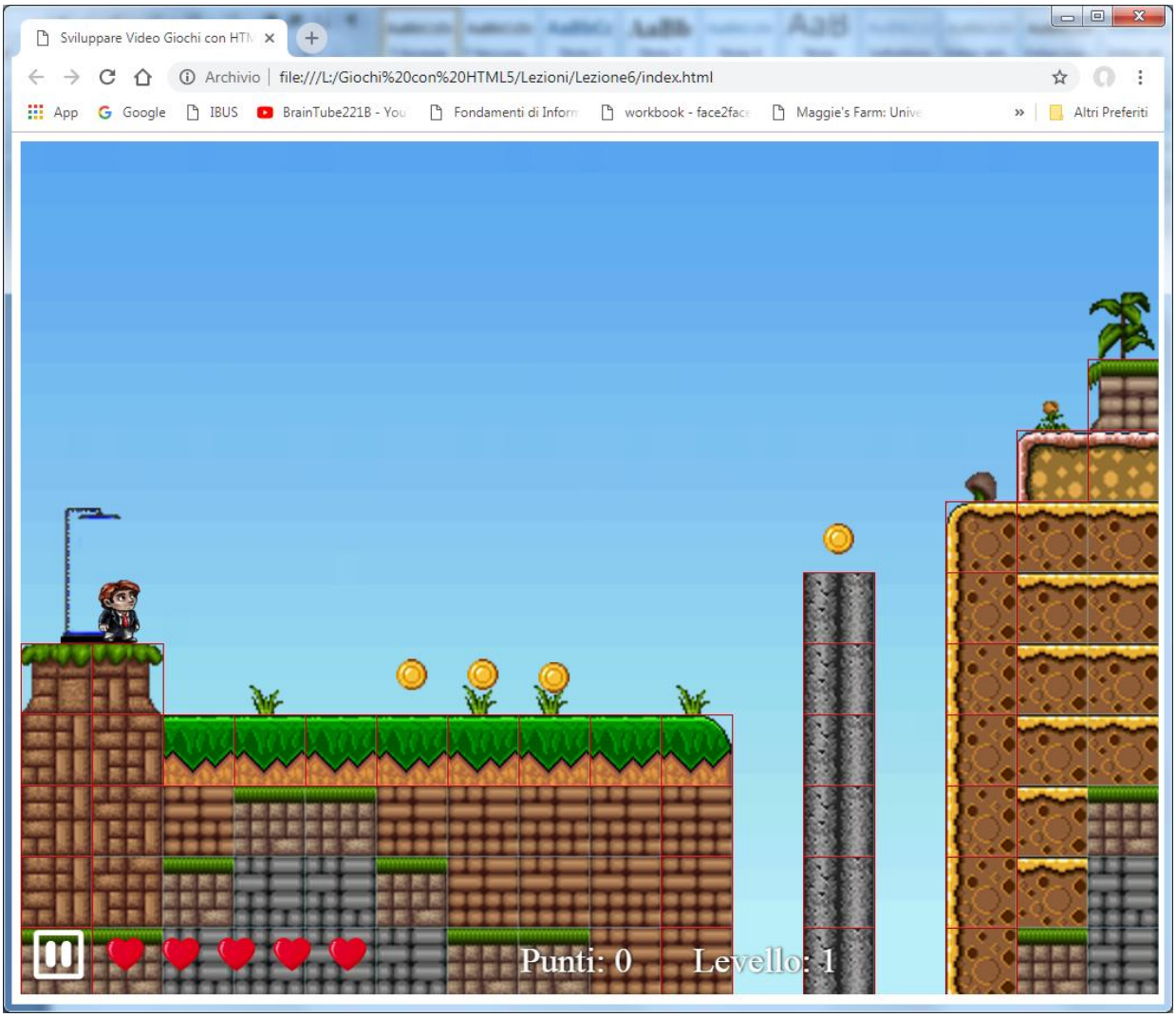
In LoadLevel, dopo la creazione del player, istanziamo l'oggetto Display:

```
this.display = new Display();
```

In Draw dell'oggetto Game, dopo il rendering dei GameObjects e il debug dei blocchi, aggiungiamo:

```
this.display.Draw();
```

proviamo a mandare in esecuzione il gioco, dovremmo ottenere la schermata seguente:



Per migliorare la giocabilità possiamo anche modificare il frame rate e cioè il tempo di ridisegno della scena.

Come abbiamo visto nelle prime lezioni la funzione:

```
window.requestAnimationFrame(function () {
    // rilancia la funzione GameLoop ad ogni frame
    game.GameLoop();

});
```

rilancia il gameloop 60 volte al secondo possiamo ridurre il tempo di aggiornamento inserendo all'inizio della funzione game:

```
function Game() {
    var tempoAttuale;
    var tempoIniziale = Date.now();
    tempoIniziale=(tempoIniziale/1000)*60; //tempo inziale in sessantesimi di secondo
```

e modificando il metodo GameLoop come segue:

```
/*implementazione del GameLoop*/
this.GameLoop = function () {

    tempoAttuale = Date.now();
    tempoAttuale = (tempoAttuale / 1000) * 60
    if (tempoAttuale - tempoIniziale > 0) {
        if (!this.paused) {

            // aggiorna tutti gli oggetti
            this.Update();

        }

        tempoIniziale = tempoAttuale;
    }
    //disegna l'intera scena a schermo
    this.Draw();
    //resetta lo stato degli inputs
    this.EndLoop();
    Inputs.Clear();

    window.requestAnimationFrame(function () {
        // rilancia la funzione GameLoop ad ogni frame
        game.GameLoop();

    });
}
/*fine GameLoop */
```

In questo modo l'aggiornamento sarà fatto 30 volte al secondo. Arrivati a questo punto possiamo eliminare il rendering dei blocchi di collisione dalla funzione draw nel file main. Js ottenendo la schermata: