OOP e VideoGame

Arrivati a questo punto dello sviluppo del gioco dovremmo aggiungere allo scenario altri personaggi come ad esempio nemici e uno scopo del gioco come raccogliere oggetti per accumulare punti ecc. e soprattutto un punto finale con il quale si passa al livello successivo. Se vogliamo aggiungere ad esempio un nemico ci accorgiamo subito che questo nuovo oggetto avrebbe molte proprietà e funzionalità simili al player ad esempio lo spostamento l'animazione dei frame e la gestione delle collisioni. Il motivo per cui ha avuto così tanto successo la programmazione orientata agli oggetti consiste proprio in questo, e cioè la possibilità di raccogliere tutte le caratteristiche comuni in un oggetto padre e grazie all'ereditarietà derivare da esso oggetti figli che ereditano da esso tutte le proprietà e i metodi in comune; risparmiando così la riscrittura di codice ed evitando la proliferazione di variabili con nomi improbabili. Abbiamo visto come sfruttare l'ereditarietà di primo livello in javascript tramite **protoype**, cerchiamo quindi di sfruttare l'ereditarietà per eliminare il codice ripetuto e includere il *codice riusabile* in un unico oggetto **GameObj**.

Per prima cosa creiamo un nuovo file di nome GameObjects.js nel quale andremo ad inserire il codice per creare un gameobject padre che racchiude tutte le caratteristiche comuni e i vari oggetti del nostro oggetto. Cominciamo quindi con l'oggetto GameObject:

```
function GameObj(x, y) {
    // variabili di uso comune inizializzate dal costruttore
    this.x = x;
    this.y = y;
    this.animSpeed = 1;
    this.curFrame = 0;
    this.xOffset = 0;
    this.yOffset = 0;

    /* questo metodo imposta l'offset di  rendering al centro dello sprite*/
    this.OffsetCenter = function () {
        this.xOffset = this.sprite.w / 2;
        this.yOffset = this.sprite.height / 2;
    }

    /*questo metodo  cicla una lista di gameobject ritorna l'id di un oggetto con cui
collide il bounding box*/
    this.GetCollision = function (listblocks, x, y) {
        for (var i = 0; i < listblocks.length; i++) {

            if (this.bbox.CollidesAt(listblocks[i].bbox, x, y)) {
                return game.blocks[i];
            }
        }
        return null;
    }

    /* questo metodo realizza il draw per GameObject semplici*/
    this.__DrawSimple = function () {
        game.ctx.drawImage(this.sprite, this.x - game.viewX - this.xOffset, this.y -
game.viewY - this.yOffset);
    }

    /*questo metodo realizza il draw per GameObject animati*/
    this.__DrawAnimated = function () {
        var ox = Math.floor(this.curFrame) * this.sprite.w;
        game.ctx.drawImage(this.sprite, ox, 0, this.sprite.w, this.sprite.height, this.x -
game.viewX - this.xOffset, this.y - game.viewY - this.yOffset, this.sprite.w,
this.sprite.height);
    }

    /*questo metodo imposta lo sprite e seleziona l'evento draw corretto*/
    this.SetSprite = function (sprite) {
        this.sprite = sprite;
```

```
        if (sprite.frames > 0) {
            // imposta l'evento draw animato
            this.Draw = this.__DrawAnimated;
        } else {
            //imposta l'evento draw semplice e cancella l'evento animazione
            this.Draw = this.__DrawSimple;
            this.UpdateAnimation = function () { };
        }
    }

    /*questo metodo aggiorna i frames dell'animazione in base a animSpeed, che può essere
sia positiva che negativa.*/
    this.UpdateAnimation = function () {
        this.curFrame += this.animSpeed;
        if (this.animSpeed > 0) {
            var diff = this.curFrame - this.sprite.frames;
            if (diff >= 0) {
                this.curFrame = diff;
                // al termine dell'animazione, chiama questa funzione
                this.OnAnimationEnd();
            }
        }
        else if (this.curFrame < 0) {
            this.curFrame = (this.sprite.frames + this.curFrame) - 0.0000001;
            // al termine dell'animazione, chiama questa funzione
            this.OnAnimationEnd();
        }
    }

    /*questo metodo distrugge l'istanza corrente eliminandola dalla lista dei gamobjects e
attiva l'evento OnDestroy*/
    this.Destroy = function () {
        this.OnDestroy();
        game.gameobjects.splice(game.gameobjects.indexOf(this), 1);
    }
    /* inizializza le variabili degli eventi non generici con funzioni vuote esse saranno
specializzate dai singoli oggetti*/
    this.Draw = function () { }
    this.Update = function () { }
    this.OnAnimationEnd = function () { }
    this.OnDestroy = function () { }

    /*aggiunge questo gameobject alla lista di gameobjects*/
    game.gameobjects.push(this);
}
```

Utilizzeremo questo oggetto come base per tutti gli oggetti animati e non del gioco quindi con questo nuovo approccio possiamo generalizzare le funzioni di aggiornamento e ridisegno di tutti gli oggetti nel gameloop.

Iniziamo con il Definire l'**array gameobject** in *ResetLevel inserendo il codice:*

```
this.ResetLevel = function () {
    //...
    this.gameobjects = [];
}
```

E quindi generalizziamo le funzioni **Update**, **EndLoop** e **Draw** di Game:

```
// aggiorna tutti i gameobjects
this.Update = function () {
    if (this.level > 0) {
        for (var i = 0; i < this.gameobjects.length; i++) {
            this.gameobjects[i].Update();
        }
    }
}
// aggiorna tutte le animazioni
this.EndLoop = function () {
    if (this.level > 0) {
        for (var i = 0; i < this.gameobjects.length; i++) {
```

```
            this.gameobjects[i].UpdateAnimation();
        }
    }
}

// disegna le tiles e i gameobjects
this.Draw = function () {
    //...
    // al posto di player.Draw()
    for (var i = 0; i < this.gameobjects.length; i++) {
        this.gameobjects[i].Draw();
    }
    //...
}
```

In questo modo generalizziamo gli aggiornamenti, ad ogni loop, gestendo il tutto con ciò che viene dichiarato nei rispettivi metodi di tutti i game object.

Ereditarietà dei game object

Definiamo una nuova funzione, che chiamiamo Eredita, che ci consente di creare nuovi oggetti eredi di GameObj e la inseriamo alla fine del file Gameobjects.js :

```
function Eredita(obj, parent) {
    // impostiamo GameObj come parent predefinito
    if (parent == undefined)
        parent = GameObj;
    //imposta il prototype per ereditare dal parent
    obj.prototype = Object.create(parent.prototype);
    //imposta il costruttore per obj
    obj.prototype.constructor = obj;
}
```

Per verificare le nuove funzionalità che abbiamo definito iniziamo ad apportare le modifiche necessarie all'oggetto Player, per ereditare da *GameObj il nuovo codice è tutto riportato di seguito :*

```
//oggetto player
    function Player() {
        // imposto le coordinate di partenza
        this.xStart = 60;
        this.yStart = 2;
        this.x = this.xStart;
        this.y = this.yStart;
        // chiamo il costruttore del parent
        GameObj.call(this, this.x, this.y);
        //imposto lo sprite e il metodo Draw
        this.SetSprite(game.sprPlayerRun, true);
        this.curFrame = 0;
        this.flip=1;
        this.animSpeed = 0.2;
        this.width = this.sprite.w;
        this.height = this.sprite.height;
        this.xOffset = Math.floor(this.width / 2);
        this.yOffset = this.height;
        this.maxSpeed = 5;
        this.hSpeed = 0;
        this.vSpeed = 0;
        this.gravity = 0.8;
        //creazione del rettangoo di collisione
        this.bbox = new BoundingBox(this.x - this.width/2, this.y, this.width, this.height);
        this.Draw = function () {
            game.ctx.save();
            game.ctx.translate(this.x - game.viewX, this.y - game.viewY);
            game.ctx.scale(this.flip, 1);
            var ox = Math.floor(this.curFrame) * this.width;
```

```
game.ctx.drawImage(this.sprite, ox, 0, this.sprite.w, this.sprite.height,
                  -this.xOffset, -this.yOffset, this.sprite.w, this.sprite.height);//le
coordinate negative servono per flippare l'immagine


            game.ctx.restore();

        }
        this.Update = function () {
                     if (Inputs.GetKeyDown(KEY_RIGHT)) {
                if (this.hSpeed < 0) this.hSpeed = 0;
                if (this.hSpeed < this.maxSpeed) this.hSpeed += 0.8;



            }
            else if (Inputs.GetKeyDown(KEY_LEFT)) {
                if (this.hSpeed > 0) this.hSpeed = 0;
                if (this.hSpeed > -this.maxSpeed) this.hSpeed -= 0.8;
            }
            else if (Inputs.GetKeyDown(KEY_UP) && this.GetCollision(game.blocks, 0, 1)) {
                if (this.vSpeed == 0) this.vSpeed = -18;

            }

                    else {
                this.hSpeed /= 1.1;
                if (Math.abs(this.hSpeed) < 1) {
                    this.hSpeed = 0;
                                //imposto lo sprite del personaggio fermo
                                this.sprite = game.sprPlayerIdle;
                                this.curFrame = 0;
                            }
            }
            if (this.hSpeed != 0) {
                // sposto il player

                            var collides = false;
for(var a = Math.abs(this.hSpeed); a > 0; a--) {
       if(this.hSpeed > 0) {
              if( !this.GetCollision(game.blocks, a , 0)) {
                     this.x += a;
                     break;
              } else
                     collides = true;
       }
       else {
              if( !this.GetCollision(game.blocks, - a , 0)) {
                     this.x -= a;
                     break;
              } else
                     collides = true;
       }
}
if(collides) {
       this.hSpeed = 0;
}


                            // orientamento orizzontale dello sprite
                            this.flip = (this.hSpeed < 0) ? -1 : 1;
                            //cambio sprite
                            if(this.sprite != game.sprPlayerRun) {
                                   this.sprite = game.sprPlayerRun;
                                   this.curFrame = 0;
                            }
                     }
                     this.vSpeed += this.gravity;
                     collides = false;
```

```
                        for(var a = Math.abs(this.vSpeed); a > 0; a-=Math.abs(this.gravity)) {
                        if(this.vSpeed > 0) {
                        if( !this.GetCollision(game.blocks, 0, a)) {
                               this.y += a;
                               break;
                        } else {
                               collides = true;
                        }
                        }
                        else {
                        if(!this.GetCollision(game.blocks, 0 , -a)) {
                               this.y -= a;
                               break;
                        } else {
                               collides = true;
                        }
                  }
            }
            if(collides) {
                  this.vSpeed = 0;
            }

            if(this.vSpeed > 0) {
                  this.sprite = game.sprPlayerFall;
                  this.curFrame = 0;}
            else if(this.vSpeed < 0) {
                  this.sprite = game.sprPlayerJump;
                  this.curFrame = 0;
                        }

var targetX = Math.xMinMax(this.x - game.canvas.width/2, 0, game.areaW - game.canvas.width)
                  var targetY = Math.xMinMax(this.y - game.canvas.height/2, 0, game.areaH -
game.canvas.height)

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

this.bbox.Move(this.x - this.xOffset,this.y-this.yOffset);
}



}
```

Inseriamo alla fine della funzione Player, il seguente codice:

```
// Player eredita da GameObj
Eredita(Player);
```

non ci resta che importare in index.html il file GameObjects.js

```
<script language="javascript" src="main.js" ></script>
<script language="javascript" src="GestioneRisorse.js" ></script>
<script language="javascript" src="display.js" ></script>
<script language="javascript" src="inputs.js" ></script>
<script language="javascript" src="player.js" ></script>
<script language="javascript" src="bbox.js" ></script>
<script language="javascript" src="levels.js" ></script>
<script language="javascript" src="utils.js" ></script>
<script language="javascript" src="GameObjects.js"></script>
```

Se abbiamo fatto tutto giusto dovrebbe funzionare tutto come prima.

Creare un GameObj per i nemici

Iniziamo a creare qualche nemico caricando come al solito lo sprite animato nella funzione:

`this.init=function()` del file main.js :

```
this.sprUmbrella = this.gr.LoadSprite("immagini/umbrella.png", 4);
```

Definiamo poi l'oggetto Umbrella, che è un ombrellino animato che si sposta in orizzontale invertendo il senso di spostamento quando incontra un muro:

```
function Umbrella(x, y) {
    GameObj.call(this, x, y);
    // imposta lo sprite e l'evento draw
    this.SetSprite(game.sprUmbrella);
    this.hSpeed = 2;
       this.bbox = new BoundingBox(this.x, this.y, this.sprite.w, this.sprite.height);
    this.life = 5;
    this.animSpeed = 0.25;
    this.width=game.cellSize;
       this.height=game.cellSize;
       this.Update = function () {
        // se collide con un blocco, cambia direzione
        if (this.GetCollision(game.blocks,this.hSpeed/2, 0)) {
            this.hSpeed = -this.hSpeed;

        }
        this.x += this.hSpeed;
       this.bbox.Move(this.x, this.y);

             if(this.life < 0){
                   this.Destroy();
                   }

    }
    // inserisco l'istanza nell'array dei nemici
    game.nemici.push(this);
    //quando viene distrutto, cancella dall'array dei nemici
    this.OnDestroy = function () {
        game.nemici.splice(game.nemici.indexOf(this), 1);
    }

}
//eredita da GameObj
Eredita(Umbrella);
```

Salviamo tutto questo codice in un nuovo file Umbrella.js e lo importiamo in index.html. Ho scelto di avere un file separato per ogni nemico in modo da avere il codice più snello e più leggibile. Un'alternativa poteva essere inserire il codice di tutti i nemici in GameObjects.js .

Aggiungiamo un nuovo array di nome *nemici* in *ResetLevel*:

```
this.ResetLevel = function () {
    // ...
    this.nemici = [];
}
```

Nella mappa che abbiamo importato, ci sono già alcuni oggetti, tra i quali ovviamente anche l'ombrellino cha abbiamo appena creato. Possiamo caricarli aggiungendo a *LoadLevel* il seguente codice, appena sotto il caricamento delle collisioni:

```
// dati sui blocchi di collisione
// ...
// dati sugli oggetti
var objects = levels[lev][3];
```

```
for (var i = 0; i < objects.length; i++) {
    var object = null;
    var x = objects[i][0] * this.cellSize;
    var y = objects[i][1] * this.cellSize;
    switch (objects[i][2]) {
        case "umbrella":
            new Umbrella(x, y);
            break;
    }
}
```

Proviamo a riavviare il gioco, dovremmo avere nello spazio di gioco tre ombrellini che si spostano avanti e indietro. Supponiamo adesso che l'ombrellino nemico che abbiamo creato se entra in collisione con il player gli provoca un danno. Per implementare questa funzionalità dovremmo prevedere che il player abbia una certa quantità di salute (o di vite) e ogni volta che viene colpito questa si decrementa fino a morire. Quando il player muore si può pensare di terminare il gioco oppure di ripartire dal punto in cui si arrivati.

Arrivati a questo punto implementiamo anche la funzionalità relativa al caso in cui il player precipita fuori dalla mappa di gioco causandone anche in questo caso la morte. Iniziamo quindi ad aggiungere al player queste funzionalità prevedendo anche che nel caso il player sia colpito da un nemico questo venga enfatizzato da un effetto trasparenza e dia al player qualche secondo di invulnerabilità. Aggiungiamo innanzitutto al player le seguenti variabili:

```
//oggetto player
    function Player() {
      //. . . . .
      this.gravity = 0.8;

      // variabile che indica il numero di vite del personaggio
        this.lives = 5;
        // variabile che indica se il personaggio è stato colpito
        this.hit = false;
        //durata dell'invulnerabilità dopo il colpo
        this.hitTimer = 0;
        //variabili trasparenza
        this.hitAlpha = 0;
        this.hitAlphaTimer = 30;

      //. . . . .
```

Nella funzione update del player aggiungiamo il seguente codice per gestire lo stato di colpito e il precipitare:

```
// aggiorna le variabii relative allo stato di "colpito"
            if (this.hit) {
                this.hitTimer--;
                this.hitAlphaTimer--;
                if (this.hitAlphaTimer < 0) {
                    this.hitAlpha = !this.hitAlpha;
                    this.hitAlphaTimer = 10;
                }
                if (this.hitTimer <= 0) {
                    this.hit = false;
                }
            }
this.bbox.Move(this.x - this.xOffset,this.y-this.yOffset); //questa riga già esisteva
             //se il player cade sotto il livello, muore
            if (this.y > game.areaH) {
                this.Die();
            }
```

E subito dopo la fine della funzione update quindi prima della graffa che chiude il Player ggiungiamo le funzioni :

```
//imposta lo status in "colpito" per un certo tempo
        this.Hit = function () {
            this.hit = true;
            this.hitTimer = 140;
            this.hitAlpha = true;
            this.hitAlphaTimer = 10;
```

```
            // riduce le vite di 1
            this.lives--;
            if (this.lives <= 0) {
                this.Die();
            }
            //simula un contraccolpo verso l'alto
            this.vSpeed = - 10;
        }
        // riporta il personaggio all'inizio del livello
        // azzera le vite e il punteggio
        this.Die = function () {
            this.x = this.xStart;
            this.y = this.yStart;
            this.lives = 5;
            game.score = 0;
        }
```

Modifichiamo la funzione Draw del player aggiungendo il codice per la trasparenza:

```
this.Draw = function() {
    game.ctx.save();
    // se è stato colpito, utilizza la trasparenza "a intermittenza"
    if(this.hit) {
        if(this.hitAlpha)
            game.ctx.globalAlpha = 0.3;
        else
            game.ctx.globalAlpha = 0.6;
    }
//. . .. il resto è uguale
```

Aggiungiamo anche all'oggetto game di main.js la variabile punti scrivendo: `this.punti=0;` che utilizzeremo più avanti quando svilupperemo le funzionalità per permettere al player di guadagnare dei punti e quindi dare uno scopo al giocatore. Arrivati a questo punto non ci resta che fare in modo che l'ombrellino quando colpisce il player gli provoca un danno. Per comodità inserisco il codice completo dell'oggetto Umbrella:

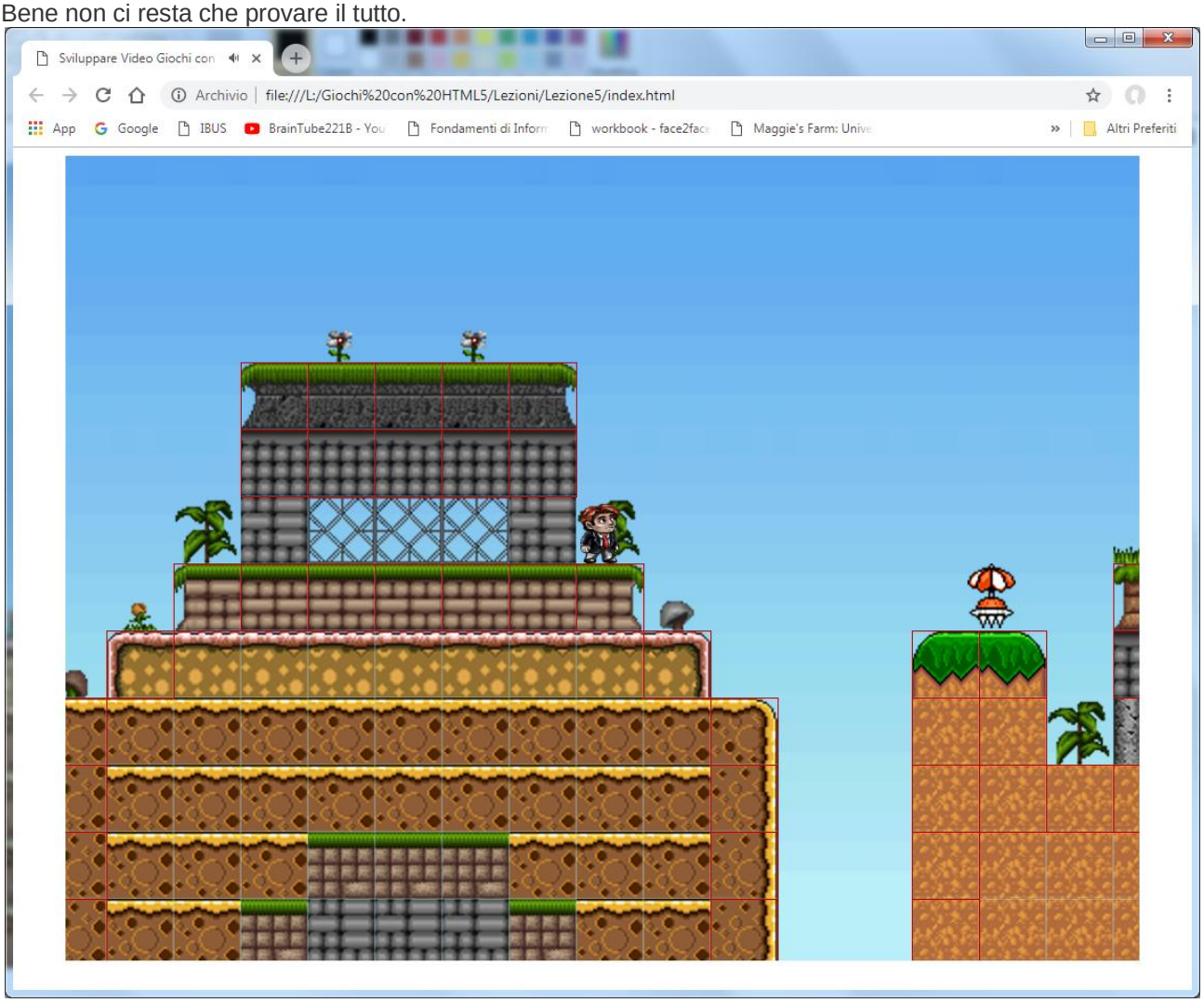
```
function Umbrella(x, y) {
    GameObj.call(this, x, y);
    // imposta lo sprite e l'evento draw
    this.SetSprite(game.sprUmbrella);
    this.hSpeed = 1;
       this.bbox = new BoundingBox(this.x, this.y, this.sprite.w, this.sprite.height);
    this.life = 5;
    this.animSpeed = 0.25;
    this.width=game.cellSize;
       this.height=game.cellSize;
       this.Update = function () {
        // se collide con un blocco, cambia direzione
        if (this.GetCollision(game.blocks,this.hSpeed/2, 0)) {
            this.hSpeed = -this.hSpeed;

        }
        this.x += this.hSpeed;
       this.bbox.Move(this.x, this.y);

             if(this.life < 0){
                    this.Destroy();
                    }
             // se il player non è in status "colpito/invulnerabile"
        if (!game.player.hit) {
            // se entra in collisione con il Player
            if (this.bbox.Collide(game.player.bbox)) {
                // colpisci il player
                game.player.Hit();
            }
        }
    }
    // inserisco l'istanza nell'array dei nemici
    game.nemici.push(this);
    //quando viene distrutto, cancella dall'array dei nemici
```

```
    this.OnDestroy = function () {
        game.nemici.splice(game.nemici.indexOf(this), 1);
    }
}
//eredita da GameObj
Eredita(Umbrella);
```

Bene non ci resta che provare il tutto.



Se tutto è andato bene vedremo gli ombrellini andare avanti e indietro