Player principale e collisioni

sviluppiamo il gioco vero e proprio e iniziamo creando il personaggio principale (il player). Iniziamo col **creare l'animazione** , servendoci di una strip con tutti i fotogrammi (frame) affiancati.

Carichiamo quindi le immagini relative al personaggio e un'immagine per il background all'interno della funzione *Init*, nel file *main.js*, come visto nelle lezioni precedenti:

```
this.sprPlayerIdle     = this.gr.LoadSprite("immagini/playerIdle.png",1);
this.sprPlayerIdleShot = this.gr.LoadSprite("immagini/playerShot.png",1);
this.sprPlayerRun      = this.gr.LoadSprite("immagini/playerRun.png",6);
this.sprPlayerJump     = this.gr.LoadSprite("immagini/playerJump.png",1);
this.sprPlayerJumpShot = this.gr.LoadSprite("immagini/playerJumpShot.png",1);
this.sprPlayerFall     = this.gr.LoadSprite("immagini/playerFall.png",1);
this.sprPlayerFallShot = this.gr.LoadSprite("immagini/playerFallShot.png",1);
this.background1       = this.gr.LoadSprite("immagini/sky.png", 1);
```

Definiamo quindi l'oggetto player all'interno di un nuovo file *player.js*

```
//oggetto player
    function Player() {
        this.sprite = game.sprPlayerRun;
        this.curFrame = 0;
        this.flip = 1;
        this.animSpeed = 0.2;
        this.width = this.sprite.w;
        this.height = this.sprite.height;
        this.xStart = game.canvas.width / 2;
        this.yStart = game.canvas.height / 2 - 60;
        this.x = this.xStart;
        this.y = this.yStart;
        this.xOffset = Math.floor(this.width / 2);
        this.yOffset = 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);
            game.ctx.restore();
        }
    }
```

Definiamo, fin da subito, gran parte delle variabili locali che ci serviranno:

Variabile	Descrizione
sprite	Immagine contenente tutti i frames dell'animazione corrente
curFrame	Il frame corrente
width	Larghezza del singolo frame (non bisogna utilizzare sprite.width perchè questa variabile corrisponde alla larghezza totale dell'immagine e non del singolo frame)
height	Altezza del singolo frame
xStart, yStart	Coordinate di partenza

x, y	Coordinate attuali del personaggio
xOffset, yOffset	Offset con cui disegneremo l'immagine

Definiamo anche la funzione **Draw** che svolgerà le seguenti azioni:

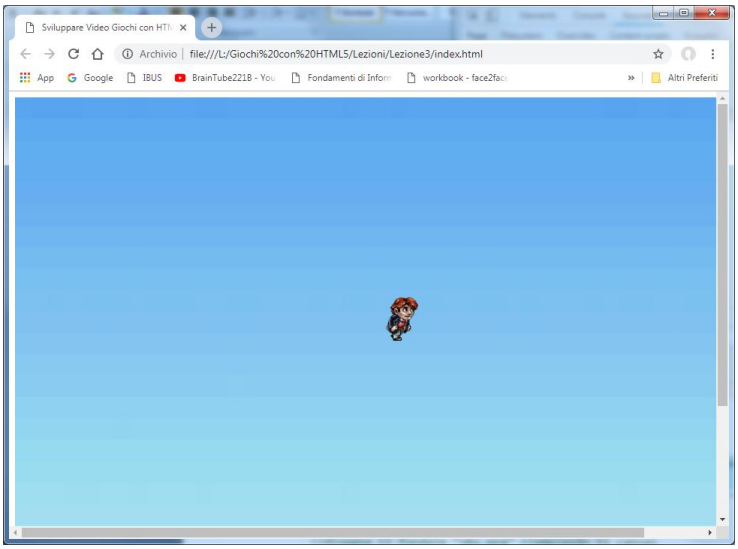
- Salva l'impostazione del context attuale;
- Trasla il context in base alle coordinate del personaggio, meno la X della view (che serve a simulare una telecamera: nel nostro caso seguirà il personaggio e influirà sulla traslazione dell'intera scena);
- Flips (ribalta) il context, in base all'orientamento del personaggio;
- Calcola la posizione X del frame corrente nell'intera strip di animazione.
- Disegna solamente la porzione di immagine relativa al frame corrente utilizzando la funzione DrawImage, nella sua forma estesa (maggiori info su W3C)
- Ripristina le impostazioni del context

Ricordiamoci di inizializzare le seguenti variabili nell'oggetto ***Game***:

```
this.viewX = 0;
this.viewY = 0;
```

Quindi inseriamo nella function **Draw** dell'oggetto **Game**, il **draw** del background e la funzione **Draw** del **player**. Poi per testare, creiamo un istanza dell'oggetto player e vediamo se viene renderizzata a schermo se tutto va bene dovremmo vedere l'immagine qui sotto:

```
this.ResetLevel = function () {
            //...
            this.player = null;
        }
this.LoadLevel = function (lev) {
    //...
    if (lev == 0) {
                //...
    }
    else {
        //carica un livello di gioco
        this.player = new Player();
    }
}
this.Draw = function () {
    //...
    if (lev == 0) {
        //...
    } else {
        //disegna il fondale "sky.png" riempiendo il canvas
        this.ctx.drawImage(this.background1, 0, 0, this.canvas.width, this.canvas.height);
              //livello di gioco
              this.player.Draw();
           }
}
```



Ovviamente prima di provare importiamo il file player.js in index.html

I Movimenti

Il passo successivo è quello di dare vita al nostro personaggio, facendo in modo che risponda agli input da tastiera.

Aggiungiamo qualche variabile al nostro oggetto Player:

```
this.maxSpeed = 10;
this.hSpeed = 0;
this.vSpeed = 0;
```

Quindi aggiungiamo la funzione **Update:**

```
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 {
                this.hSpeed /= 1.1;
                if (Math.abs(this.hSpeed) < 1) {
                    this.hSpeed = 0;
                }
            }
            if (this.hSpeed != 0) {
                // sposto il player
                this.x += this.hSpeed;
            }
        }
```

Il primo if, verifica che il tasto KEY_RIGHT (Freccia a destra) della tastiera sia premuto. Quindi controlla che la velocità orizzontale sia minore di 0 (in tal caso il personaggio si stava muovendo a sinistra):

```
if (this.hSpeed < 0) this.hSpeed = 0;
```

se true: imposta la sua velocità a 0, ovvero ferma il personaggio per fare in modo che inizi a muoversi verso destra. Se la velocità orizzontale è entro i limiti della velocità massima:

```
if (this.hSpeed < this.maxSpeed) this.hSpeed += 0.8;
```

incrementa la velocità orizzontale (che assumerà valori positivi sempre maggiori, quindi è una velocità orizzontale verso destra).
Lo stesso discorso si applica alla pressione del tasto KEY_LEFT, ma con segni opposti, dato che si tratta di una velocità negativa sull'asse x.
Se nessuno dei due tasti è premuto, la velocità orizzontale viene ridotta dividendola per un valore (nel nostro caso 1.1) e se il suo valore assoluto è minore di 1 (utilizziamo Math.abs in modo che il segno +/- non influisca sul controllo della variabile), imposta a 0 la velocità del player.
Infine, se la velocità orizzontale è diversa da 0, spostiamo la X del personaggio di un numero di pixel, pari alla sua velocità.
L'ultimo passo, è quello di aggiungere player.Update all'interno dell'evento Update dell'oggetto Game:

```
//aggiorna tutto
this.Update = function () {
    if (this.level > 0) {
        this.player.Update();
    }
}
```

Se avviamo il gioco, dovremmo avere la possibilità di spostare il personaggio sull'asse X premendo i tasti della tastiera.
Poiché un personaggio paralizzato non è tanto bello da vedere, vediamo subito come applicare le animazioni e definiamo la funzione UpdateAnimation, che gestirà il flusso dei frames dell'animazione, all'oggetto Player.

```
this.UpdateAnimation = function () {
    this.curFrame += this.animSpeed;
    if (this.animSpeed > 0) {
        var diff = this.curFrame - this.sprite.frames;
        if (diff >= 0) {
            this.curFrame = diff;
        }
    }
    else if (this.curFrame < 0) {
        this.curFrame = (this.sprite.frames + this.curFrame);
    }
}
```

Aggiungiamo all'oggetto Game, una funzione *EndLoop*, che verrà chiamata nella funzione *GameLoop*, subito dopo il *draw*. All'interno di *EndLoop*, eseguiamo *UpdateAnimation* del personaggio

```
//aggiorna animazioni
this.EndLoop = function () {
    if (this.level > 0) {
        this.player.UpdateAnimation();
    }
}
this.GameLoop = function () {
    //..
    this.Draw();
    this.EndLoop();
    //..
}
```

Se proviamo ad avviare il gioco vedremo che il personaggio esegue continuamente l'animazione di corsa. Dobbiamo perciò impostare lo sprite di Idle (personaggio fermo), quando non sta effettivamente correndo e l'animazione di corsa quando hSpeed è diversa da 0. Aggiungiamo alcune righe di codice alla funzione Update del player:

```
this.Update = function() {
       if(Inputs.GetKeyDown(KEY_RIGHT)) {       //...  }
       else if(Inputs.GetKeyDown(KEY_LEFT)) { //...    }
       else{//x...
              if(Math.abs(this.hSpeed) < 1) {
                     // ...
                     //imposto lo sprite del personaggio fermo
                     this.sprite = game.sprPlayerIdle;
                     this.curFrame = 0;}
           }
          //...
       if(this.hSpeed != 0) {
              // sposto il player
              this.x += hSpeed;
              // 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;
              }
```

```
        }
}
```

Quando impostiamo l'animazione della corsa, dobbiamo anche impostare il flip orizzontale (variabile this.flip) in base alla velcocità orizzontale.

Collisioni

Perché il personaggio interagisca con il mondo circostante, abbiamo bisogno di introdurre il meccanismo delle **collisioni**. Esistono diverse tecniche per farlo, più o meno elaborate e realistiche. In questa guida utilizzeremo per semplicità il metodo del **BoundingBox**. Con questo sistema, si dovrà approssimare ogni forma ad un rettangolo, in modo da poter verificare in modo semplice l'intersezione tra due forme.

Creiamo un nuovo file bbox.js e all'interno inseriamo il seguente codice (BoundingBox nuovo oggetto)

```
function BoundingBox(x, y, w, h) {
    this.x = x;
    this.y = y;
    this.width = w;
    this.height = h;
    }
```

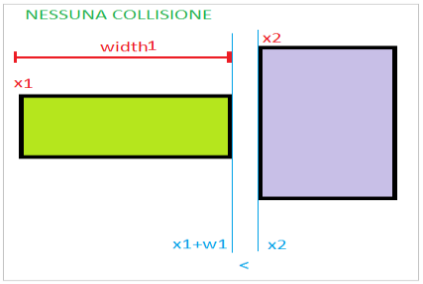
Dichiariamo quindi 2 variabili per indicare la posizione (x, y) e due per le dimensioni (width, height) del nostro rettangolo. Per **verificare la collisione** con un altro bounding box, possiamo inserire la seguente funzione:

```
this.Collide = function (b) {
    return !(this.x + this.width < b.x || b.x + b.width < this.x ||
  this.y + this.height < b.y || b.y + b.height < this.y);
}
```

L'argomento *b* della funzione è un altra istanza di BoundingBox.

Questa funzione utilizza le seguenti condizione per verificare se i rettangoli *NON* si intersecano

no_collisione = (x1+w1<x2 or x2+w2<x1 or y1+h1<Y2 or y2+h2<y1)



Ecco un immagine per spiegare la prima condizione X1+W1<X2 (le altre si possono immaginare di conseguenza)

Per provare quanto abbiamo definito fin'ora aggiungiamo alla nostra schermato una serie di rettangoli di dimensioni 64x64 pixels. al solo scopo di provare le collisioni. Quindi per prima cosa definiamo nell'oggetto Game una variabile cellSize che ci servirà anche in seguito quando importeremo una tileMap per disegnare lo scenario del gioco.

```
function Game(...){
	// ...
	this.cellSize = 64;
	// ...
}
```

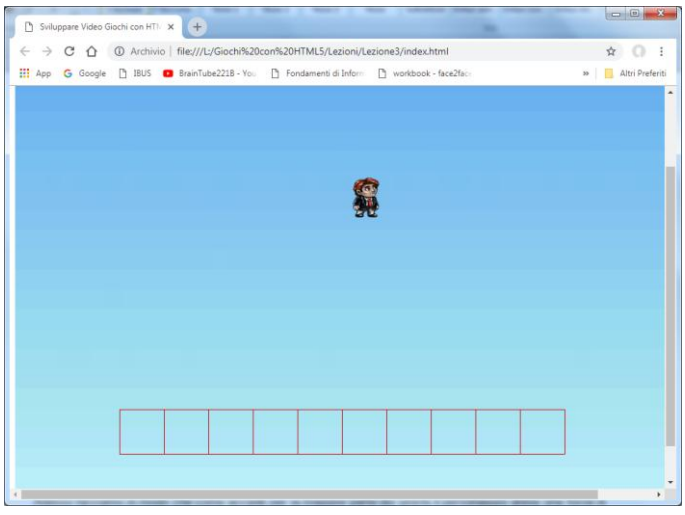
E aggiungiamo nel file main.js la seguente funzione per creare dei semplici blocchi rettangolari:

```
function Block(x, y) {
	this.x = x;
	this.y = y;
	this.width = game.cellSize;
	this.height = game.cellSize;
	this.bbox = new BoundingBox(x, y, this.width, this.height);
}
```

Creiamo quindi un array di blocchi in ResetLevel sempre dentro Game, creiamo qualche istanza appena sotto il player e aggiungiamo in Draw il rendering del contorno dei blocchetti, al solo scopo di Debug:

```
this.ResetLevel = function() {
	// ...
	this.blocks = [];
}
this.LoadLevel = function(lev) {
	// ...
	if(lev == 0) {
		// ...
	}
	else {
		// ...
		for(i=0; i<10; i++){
			this.blocks.push(new Block(150 + i*64, 600));
		}
	}
}
this.Draw = function() {
	// ...
	if(this.level == 0) {
		// ...
	}
	else {
		// ...
		// debug dei blocchi (cambio colore e spessore)
		this.ctx.strokeStyle = "#c00";
		this.ctx.lineWidth = 1;
		for(var i = 0; i < this.blocks.length; i ++) {
			this.ctx.strokeRect(this.blocks[i].x-game.viewX+0.5, this.blocks[i].y-
game.viewY+0.5,this.blocks[i].width,this.blocks[i].height);
		}
	}
	//
...}
```

Di fianco il risultato:



Adesso facciamo in modo che come accade per la maggior parte dei giochi il personaggio abbia una forza di gravità che lo attira verso il basso e sfruttando le collisioni si fermerà sui blocchi che abbiamo disegnato. Aggiungiamo quindi la variabile *gravity* al nostro oggetto *Player* e facciamo in modo che questa influenzi la sua velocità verticale (*vSpeed*):

```
....
this.gravity = 0.4;
...
this.Update = function () {
    // ...
    this.vSpeed += this.gravity;
    this.y += this.vSpeed;
}
```

Se avviamo il gioco adesso, il personaggio cadrà nel vuoto, perciò dobbiamo aggiungere un check per le collisioni con i blocchi sottostanti. Aggiungiamo al player la seguente riga per definire il suo BoundingBox

```
……….
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 ()
              {
……….
```

Aggiungiamo all'oggetto BoundingBox nel file bbox.js la funzione:

```
this.Move = function (x, y) {
        this.x = x;
        this.y = y;
    }
```

Questa funzione servirà a spostare il rettangolo di collisione nella nuova posizione del player dopo ogni spostamento. Aggiungiamo quindi nel player la chiamata a questa funzione alla fine del metodo update del player:

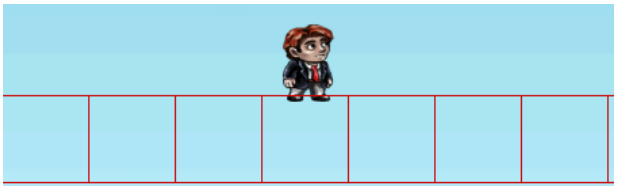
```
this.bbox.Move(this.x - this.xOffset,this.y-this.yOffset);
```

Aggiungiamo una funzione *GetCollision* al nostro player, che ritornerà l'istanza con cui collide il nostro Player, oppure null.

```
this.GetCollision = function (gameObjList) {
    for (var i = 0; i < gameObjList.length; i++) {
        if (this.bbox.Collide(gameObjList[i].bbox)) {
            return gameObjList[i];
        }
    }
    return null;
}
```

Questa funzione prende come parametri una lista di GameObjects, verso cui controllare la collisione. Più avanti provvederemo a integrare le funzioni di collisione in un unico oggetto generico, sfruttando al meglio l'ereditarietà. Sfruttiamo la funzione appena creata per controllare la collisione col terreno. All'interno di Update del player aggiungiamo:

```
this.Update = function () {
    // ...
    this.vSpeed += this.gravity;
    if (this.GetCollision(game.blocks)) {
        this.vSpeed = 0;
    }
    this.y += this.vSpeed;
}
```



Se facciamo partire il gioco avremmo un effetto non molto soddisfacente perché, verificando se il personaggio, spostato di una quantità di pixel pari a vSpeed collide con un blocco, non teniamo conto di un piccolo errore dovuto all'incremento effettuato. Questo si nota particolarmente se il framerate è basso o gravità maggiori di 1. Il Player, come si vede nell'immagine sotto, effettua un controllo spostando il bounding box di tot unità sull'asse verticale, e questo lo farà fermare quando già e entrato nel terreno. Per risolvere questo inconveniente senza molti problemi con un effetto abbastanza soddisfacente, per prima cosa aggiungiamo una nuova funzione all'oggetto BoundingBox:

```
//dx: piccol spostamento in orizzontale ,  dy: piccolo spostamento in verticale
this.CollidesAt = function(b, dx, dy){
       return !(this.x + dx + this.width < b.x || b.x + b.width < this.x + dx || this.y +
this.height + dy < b.y || b.y + b.height < this.y + dy);
}
```

Al posto della funzione precedentemente inserita nel player :

```
this.GetCollision = function (listblocks) {
    for (var i = 0; i < listblocks.length; i++) {

        if (this.bbox.Collide(listblocks[i].bbox)) {
            return game.blocks[i];
        }
    }
    return null;
}
```

Scriviamo una nuova funzione che considera questi avvicinamenti successivi chiamando CollidesAt appena creata al posto di Collide e basta.

```
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;
}
```

E al posto del semplice controllo di collisione:

```
this.vSpeed += this.gravity;
    if (this.GetCollision(game.blocks)) {
        this.vSpeed = 0;
    }
    this.y += this.vSpeed;
```

Scriviamo un controllo più raffinato come segue:

```
this.vSpeed += this.gravity;
collides = false;
for (var a = Math.abs(this.vSpeed); a > 0; a--) {
    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;
}
```

e subito dopo questa istruzione aggiungiamo le istruzioni per modificare lo sprite relative al salto e alla caduta:

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

Se facciamo ripartire vediamo che il player è ben saldo sul terreno;

Aggiungiamo due blocchi laterali e uno in alto per provare anche le collisioni sugli spostamenti orizzontali e nei salti quindi in main.s modifichiamo il codice di creazione dei blocchi come segue:

```
//inizializzazione dele vettore con 10 blocchi
    for(i=0; i<10; i++){
        this.blocks.push(new Block(150 + i*64, 600));
    }
    this.blocks.push(new Block(150 , 536));
    this.blocks.push(new Block(576+150 , 536));
    this.blocks.push(new Block(350 , 350));
```

aggiungiamo anche la possibilità di far saltare il player con il tasto freccia su: aggiungendo a player.js nella function update subito dopo l'else relativo al tasto KEY_LEFT il codice:

```
else if (Inputs.GetKeyDown(KEY_UP)) {
        if (this.vSpeed == 0) this.vSpeed = -18;
       }
```

modifichiamo anche il codice relativo ad hSpeed che precedentemente avevamo scritto:

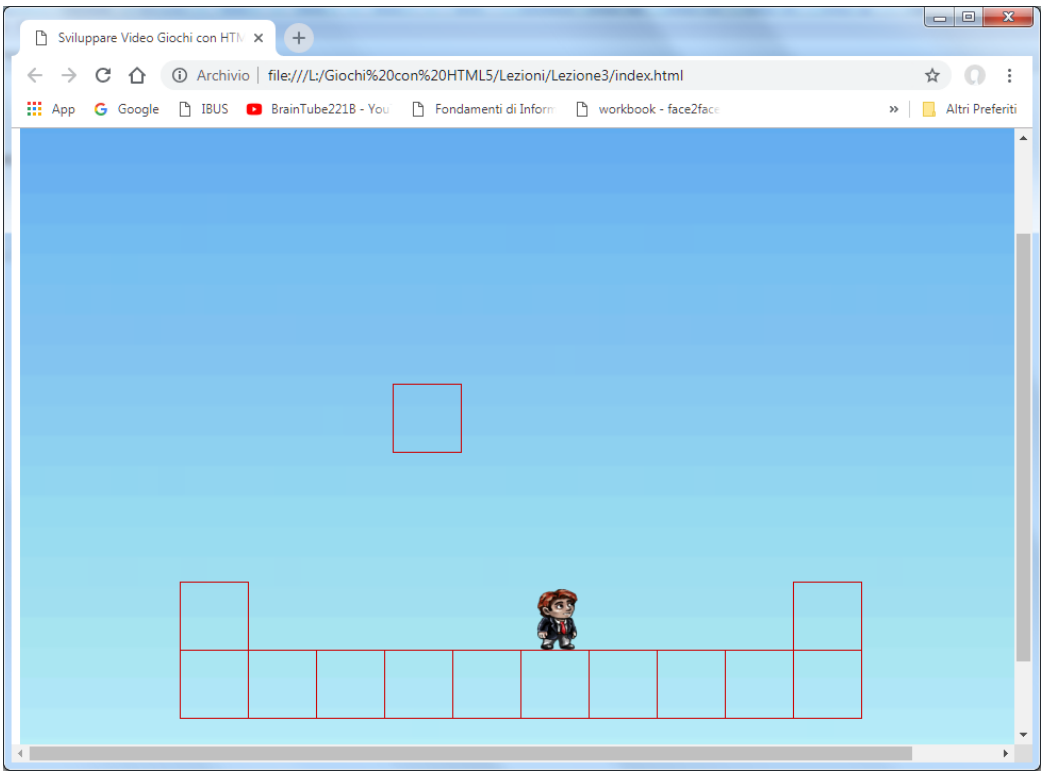
```
if (this.hSpeed != 0) {
    // sposto il player
    this.x += this.hSpeed;
    this.flip = (this.hSpeed < 0) ? -1 : 1;
    if (this.sprite != game.sprPlayerRun) {
        this.sprite = game.sprPlayerRun;
        this.curFrame = 0;
    }

}
```

con il codice seguente:

```
if (this.hSpeed != 0) {
                //sposto il player per piccolo incrementi
     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;
}
```

se facciamo ripartire il gioco dovremmo avere questo risultato:



E spostandoci e saltando le collisioni dovrebbero essere rilevate correttamente.