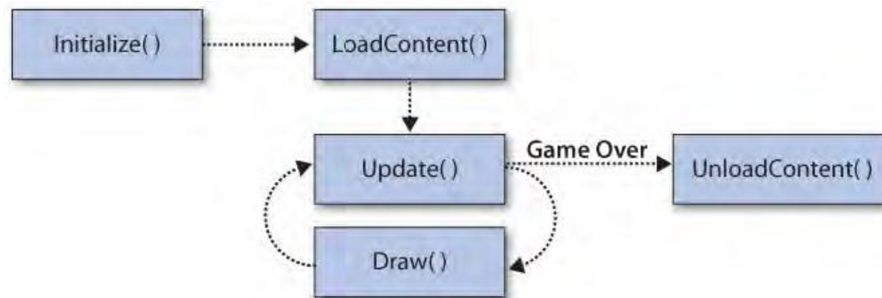
Game Loop e RequestAnimationFrame

Adesso che sappiamo come manipolare oggetti in javascript concentriamoci sullo sviluppo della struttura del gioco. Una caratteristica che accomuna tutti i giochi, è il **Game Loop**: una funzione che gestisce il ciclo del gioco attraverso due funzioni **Update,** che fa l'aggiornamento di posizione dei nostri elementi di gioco, il controllo delle collisioni ecc. e ***Draw***, ovvero il rendering degli oggetti che fanno parte del gioco (i cosiddetti ***GameObject***).



requestAnimationFrame, gestire il refresh della scena

In HTML5 è stata inserita la funzione **requestAnimationFrame** che rimpiazza il vecchio metodo di animazione, che utilizzava un timer (**setTimeout**) per renderizzare un frame ogni tot millisecondi. Con requestAnimationFrame possiamo indicare al motore JavaScript quale funzione lanciare la prossima volta che scatterà un frame. Ecco la sintassi:

```
(request ID) = requestAnimationFrame(callback)
```

Grazie a questa funzione nativa invece, il browser può **ottimizzare il rendering** nativamente mantenendo un framerate più stabile e fluido (con un massimo di 60fps),

GameLoop

Possiamo finalmente definire il **GameLoop**, ovvero quella funzione che ci permette di creare un ciclo infinito in cui gestire animazioni ed eventi frame dopo frame. Definiamo quindi la funzione GameLoop, all'interno dell'oggetto Game:

```
function Game() {
    //....tutto il codice prcedente
       this.pause = false;
       // implementazione del GameLoop
       this.GameLoop = function() {
       if(!this.paused) {

       // aggiorna tutti gli oggetti
       this.Update();
              }
       //disegna l'intera scena a schermo
       this.Draw();
       window.requestAnimationFrame(function() {
       // rilancia la funzione GameLoop ad ogni frame
       game.GameLoop();
           });
    }//fine GameLoop
    this.Draw = function() {

    }
```

```
this.Update = function(){

    }
 }//fine oggetto Game

function StartGame(){
       //crea un istanza di Game
       game = new Game();
}

window.addEventListener('load', function() {
  StartGame();
}, true);
```

Il cuore del loop sta nella chiamata a **requestAnimationFrame**, in cui chiamiamo nuovamente la funzione GameLoop dell'istanza di Game, in modo che si inneschi un meccanismo ricorsivo, in cui vengono eseguiti Update() e Draw() ad ogni frame.
La maggior parte dei **framework** per videogiochi, prevede che gli oggetti nel gioco (**GameObject**, che stiamo per vedere) siano dotati delle funzioni Update e Draw, ciò fa di esse due tasselli fondamentali dell'engine. Definiamo inoltre la variabile **paused**, che bloccherà la funzione Update quando sarà impostata a true.

Update e Draw

Il nostro game engine si basa principalmente su un loop che ad ogni frame richiama le funzioni **Update** e **Draw**. Possiamo pensare a queste funzioni come agli eventi principali del gioco. Di conseguenza tutti i **GameObject** dovranno gestire questi eventi con i relativi metodi **Update** e **Draw**.
In altre parole si tratta di funzioni comuni a tutti gli elementi del gioco, che ci permettono di gestire in modo generico tutte le istanze, nonostante esse abbiano comportamenti, variabili e funzionalità diverse.
Abbiamo organizzato la successione degli eventi nel **GameLoop**, in modo tale che **Update** sia sempre eseguito prima del **Draw**. Il motivo è semplice: Update ha il compito di modificare e spostare gli oggetti sulla scena, verificare le collisioni, gestire i frame dell'animazione, etc.
In questo modo, prima dell'esecuzione di Draw, tutte le istanze sono aggiornate, e possono essere disegnate alla posizione corretta, col giusto frame dell'animazione. Abbiamo agiunto un gestore dell'evento load alla finestra con il quale facciamo partire il gioco facendo una istanza di Game.

Gestione e caricamento delle risorse di gioco

vediamo come aggiungere al game engine un **gestore di risorse**, che ci permetterà di caricare immagini, suoni ed effettuare il check di eventuali errori. Definiamo l'oggetto (la funzione) GestioneRisorse all'interno di un nuovo file risorse.js:
Aggiungiamo poi a GestioneRisorse una funzione metodo per **caricare le immagini e gli sprite**:

```
function GestioneRisorse() {
      this.errori=0;
       //carica un immagine e ritorna un id
       this.LoadSprite = function(url, frames, funct){
              var img = new Image();
```

```
            img.src = url;
            img.rh = this;
            img.frames = frames;
            this.w = this.width / this.frames;
            img.onload = function(){
                   if(funct != undefined){
                          funct();
                   }
                   this.w = this.width/this.frames;
                   };
            img.addEventListener("error", function(e){
                   this.errori++;
            alert("errore nel caricare: "+url+"  "+this.errori);

            });
     return img;
     }
```

In **LoadSprite** creiamo un'immagine, la assegniamo ad una variabile temporanea img e impostiamo la sorgente dell'immagine (img.src) in base all'argomento url.

Fatto questo aggiungiamo all'oggetto img la proprietà **rh** (ResourceHandler), che utilizziamo per mantenere all'interno dello scope di img un riferimento all'istanza attuale dell'oggetto contenitore **GestioneRisorse**. Ci servirà per modificare le variabili di **GestioneRisorse** in una funzione globale che utilizzeremo successivamente. Salviamo il numero di frames in una variabile locale dell'immagine.

Calcoliamo la *larghezza del singolo frame* (per gli sprite delle animazioni utilizzeremo strip affiancate, come nella prossima immagine) e la salviamo nella variabile locale img.w (non possiamo utilizzare img.width, dato che è già una variabile interna, corrispondente alla lunghezza dell'intera immagine).

Gestiamo ora l'evento onload di img, che viene scatenato quando il caricamento dell'immagine è completo: eseguiamo (se esiste) la funzione passata come parametro, infine controlliamo se il processo di caricamento è terminato, tramite la funzione CheckLoaded del GestioneRisorse che definiremo tra poco.

Infine gestiamo l'evento error nel caso il file non esistesse e in caso di errore incrementiamo la variabile errori e segnaliamo l'errore. La variabile errori ci servirà all'avvio del gioco facendo partire il gioco solo se non ci sono stati errori nel caricamento delle risorse.

Nota: *Anche se il riferimento all'immagine si ottiene istantaneamente, questa viene caricata in modo asincrono, è quindi necessario attendere una risposta dall'evento load o error prima di utilizzarla.*

Aggiungiamo un metodo per caricare I suoni definendo la funzione **LoadSounds**:

```
     //carica un suono
     this.LoadSound = function(url){
            var sound = new Audio();
            sound.src = url;
            sound.addEventListener("fileload", function(e){
            sound.formatIndex = 0;
            sound.volume = 0.05;
            });
```

```
            sound.addEventListener("error", function(e){
            alert("errore nel caricare: "+url);
            this.errori++;
            });
            return sound;
        }
}
```

Questa funzione crea un elemento **audio**, lo salva in una variabile sound e successivamente imposta il source e il volume. Come per le immagini, si controlla attraverso un listener eventuali errori.

Gestire L'imput

Un altro aspetto ovviamente importante è la gestione dell'input dell'utente. Creiamo quindi un nuovo file JavaScript "inputs.js" e definiamo delle costanti che ci serviranno successivamente

```
/* inputs.js
*/
// costanti riguardanti i bottoni del mouse e alcuni tasti della tasteira
MOUSE_LEFT = 1;
MOUSE_MIDDLE = 2;
MOUSE_RIGHT = 3;
KEY_LEFT = 37;
KEY_RIGHT = 39;
KEY_UP = 38;
KEY_DOWN = 40;
KEY_ENTER = 13;
KEY_ESC = 27;
KEY_CTRL = 17;
KEY_SPACE = 32;
```

Poiché le API JavaScript per gli eventi da tastiera associano a ciascun tasto un **keycode** numerico, grazie a queste "costanti" possiamo evitare di ricordare il numero corrispondente ad ogni tasto. (In realtà si tratta di variabili: le **costanti** infatti non sono ancora supportate da tutti i browser).

Definiamo quindi il nostro oggetto Inputs e inizializziamo alcune variabili/proprietà:

```
// oggetto che gestisce gli input

Inputs = function () { }
Inputs.mouseX = 0;
Inputs.mouseY = 0;
Inputs.mouseLeft = false;
Inputs.mouseLeftPress = false;
Inputs.mouseLeftRel = false;
Inputs.mouseRight = false;
Inputs.mouseRightPress = false;
Inputs.mouseRightRel = false;
Inputs.key = [];
Inputs.keyPress = [];
Inputs.keyRel = [];
```

Proprietà	Descrizione
mouseX *mouseY*	indicano la posizione del puntatore, relativamente al game Canvas.
mouseLeft *mouseRight*	indicano se il tasto sinistro/destro sono in pressione (l'utente ha premuto il tasto e non l'ha ancora rilasciato).
mouseLeftPress	indicano se in questo momento si è premuto il

Per la tastiera, definiamo invece 3 array vuoti (key, keyPress, keyRel) , che indicano rispettivamente i tasti in-pressione, premuti e rilasciati.

Gli event listener

Per rilevare l'input dell'utente, utilizzeremo il classico **element.addElemetListener** per associare le funzioni da noi definite a ciascun evento, come la pressione o il rilascio di un tasto.

Quindi aggiungiamo al file inputs.js il seguente codice:

```
window.addEventListener("keydown", function (e) {
    if (!Inputs.key[e.keyCode]) {
        Inputs.keyPress[e.keyCode] = true;
        Inputs.key[e.keyCode] = true;
    }
}, false);
window.addEventListener("keyup", function (e) {
    Inputs.keyRel[e.keyCode] = true;
    Inputs.key[e.keyCode] = false;
}, false);
```

Abbiamo agganciato così due event listener all'oggetto window, che si occuperanno di intercettare gli eventi della tastiera e inserire i relativi codici nei nostri array.

L'evento **keydown**, cui abbiamo associato la prima funzione, viene scatenato quando l'utente preme un tasto della tastiera e viene ripetuto ogni tot millisecondi, finché il tasto rimane premuto. Perciò evitiamo di inserire il valore nell'array **keyPress** se il **keyCode** è già nell'array **key**: il tasto è già stato premuto ed è in pressione e non c'è bisogno di inserirlo nuovamente in **keyPress**.

L' evento **keyup**, cui abbiamo associato la seconda callback, viene scatenato quando il tasto viene rilasciato, perciò in questo caso aggiorniamo gli array **keyRel** e **key**, anche qui usando come indice il **keyCode**.

Facciamo qualcosa di simile anche per gli eventi del Mouse:

```
window.addEventListener("mousedown", function (e) {
    switch (e.which) {
        case 1:
            Inputs.mouseLeft = true;
            Inputs.mouseLeftPress = true;
            break;
        case 2:
            Inputs.mouseMiddle = true;
            Inputs.mouseMiddlePress = true;
            break;
        case 3:
            Inputs.mouseRight = true;
            Inputs.mouseRightPress = true;
            break;
    }
}, false);
window.addEventListener("mouseup", function (e) {
    switch (e.which) {
        case 1:
            Inputs.mouseLeft = false;
            Inputs.mouseLeftRel = true;
            break;
        case 2:
            Inputs.mouseMiddle = false;
            Inputs.mouseMiddleRel = true;
            break;
        case 3:
            Inputs.mouseRight = false;
            Inputs.mouseRightRel = true;
            break;
    }
}, false);
```

Gli eventi per i tasti del mouse ritornano un numero da 1 a 3, che indica quale sia il tasto premuto/rilasciato (1=sinistro, 2=centrale [o rotellina], 3=destro).

Aggiorniamo quindi le variabili per la posizione:

```
window.addEventListener("mousemove", function (e) {
    Inputs.mouseX = Math.round(e.pageX - game.canvas.offsetLeft);
    Inputs.mouseY = Math.round(e.pageY - game.canvas.offsetTop);
    Inputs.mouseMoved = true;
}, false);
```

Tramite l'evento **mousemove**, aggiorniamo le variabili mouseX emouseY relativamente al canvas (l'evento ritorna la posizione del mouse rispetto all'intera finestra del browser, quindi sottraiamo la posizione del canvas, così da ottenere coordinate relative).

Infine impostiamo a true la variabile mouseMoved, che potrebbe tornare utile per controllare se il mouse è stato spostato.

Ora definiamo la funzione *Clear*, che si occuperà di resettare la pressione/rilascio dei tasti.

```
Inputs.Clear = function () {
    Inputs.mouseLeftPress = false;
    Inputs.mouseLeftRel = false;
    Inputs.mouseMiddlePress = false;
    Inputs.mouseMiddleRel = false;
    Inputs.mouseRightPress = false;
    Inputs.mouseRightRel = false;
    Inputs.mouseMoved = false;
    Inputs.keyPress = [];
    Inputs.keyRel = [];
}
```

Qui non facciamo altro che impostare tutte le variabili del mouse a false e svuotare gli array della tastiera.

Dovremo chiamare questa funzione per ogni frame al termine del **gameloop**, dopo aver eseguito tutti gli eventi degli altri oggetti. Come si vede non cancelliamo, e non dobbiamo farlo, le variabili e gli array relativi agli input "in pressione" (mouseLeft, mouseMiddle, mouseRight, key), ma solo quelli del press/release che durano solo il tempo di un frame inseriamo la chiamata a questa funzione nel metodo Game di main.js:.

```
/* main.js
*/
function Game() {
    //...
    this.GameLoop = function () {
        //...
        Inputs.Clear();
    }
}
```

Dopo aver descritto le funzionalità interne, passiamo a definire le funzioni che utilizzeremo durante la programmazione del nostro gioco.

```
Inputs.GetKeyDown = function (k) {
    if (typeof (k) == "string") {
        k = k.charCodeAt(0);
    }
    return (Inputs.key[k] == true);
}
```

```
Inputs.GetKeyPress = function (k) {
    if (typeof (k) == "string") {
        k = k.charCodeAt(0);
    }
    return (Inputs.keyPress[k] == true);
}

Inputs.GetKeyRelease = function (k) {
    if (typeof (k) == "string") {
        k = k.charCodeAt(0);
    }
    return (Inputs.keyRel[k] == true);
}
```

Queste tre funzioni, prendono come parametro un valore che può essere un numero intero (**keyCode**) o una stringa (ad esempio Inputs.GetKeyDown("R") convertirà la stringa 'R' nel keyCode corrispondente 82) e verificano se l'elemento dell'array, avente come indice il keyCode, esiste ed è true.

Creiamo delle funzioni simili per gestire Down/Press/Release anche per i tre tasti del mouse

```
Inputs.GetMouseDown = function (b) {
    if (b == 1) return Inputs.mouseLeft;
    if (b == 2) return Inputs.mouseMiddle;
    if (b == 3) return Inputs.mouseRight;
}
Inputs.GetMousePress = function (b) {
    if (b == 1) return Inputs.mouseLeftPress;
    if (b == 2) return Inputs.mouseMiddlePress;
    if (b == 3) return Inputs.mouseRightPress;
}
Inputs.GetMouseRelease = function (b) {
    if (b == 1) return Inputs.mouseLeftRel;
    if (b == 2) return Inputs.mouseMiddleRel;
    if (b == 3) return Inputs.mouseRightRel;
}
```

Infine aggiungiamo una funzioni per controllare se il mouse sia all'interno di un rettangolo

```
Inputs.MouseInsideRect = function(x,y,w,h) {
      return (Inputs.mouseX >= x && Inputs.mouseY >= y && Inputs.mouseX <= x+w &&
Inputs.mouseY <= y+h);
}
```

Inseriamo nel file index.html i tags <script>…………..</script> per il caricamento dei file javascript precedentemente descritti.

Bene siamo pronti per iniziare a fare qualcosa di concreto. Iniziamo con il costruirci un menù splash screen che verrà visualizzato all'avvio del gioco

Il nostro menu, sarà composto da:

- Elementi testuali cliccabili
- Uno sfondo
- Un logo col titolo del gioco + immagine laterale del personaggio

Eventualmente un footer coi crediti.

Per prima cosa aggiungiamo all'oggetto game nel file main.js il metodo init che caricherà le risorse che ci servono il metodo ovviamente andremo ad inserirlo prima del gameloop e per gestire il caricamento delle risorse faremo una istanza del GestoreRisorse precedentemente creato.

```
-----------------
```

```
this.pause=false;
	//istanza del gestore delle risorse
	this.gr = new GestioneRisorse();
	//inizializzazione delle risorse
	this.init=function(){
			//caricamento delle risorse grafiche
			this.sprLogo = this.gr.LoadSprite("immagini/logo1.png",1);
			this.sprSplashLogo  = this.gr.LoadSprite("immagini/splashLogo.png",1);

			this.backgroundMenu = this.gr.LoadSprite("immagini/sfondomenu.jpg", 1,
			function() {  game.patternMenu =
			game.ctx.createPattern(game.backgroundMenu,"repeat");
		});

		//caricamento suoni
		this.sndMusic = this.gr.LoadSound("audio/datagrove.mp3");

	}

	// implementazione del GameLoop
	this.GameLoop = function() {
	if(!this.paused) {

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

	//disegna l'intera scena a schermo
	this.Draw();}
	Inputs.Clear();
     window.requestAnimationFrame(function() {
	// rilancia la funzione GameLoop ad ogni frame
	game.GameLoop();});
}
```

IL nostro gioco avrà dei livelli che andranno da 1 a n. pertanto al livello 0 faremo corrispondere il menù iniziale del gioco e a tale proposito aggiungiamo all'oggetto game dopo GameLoop i seguenti metodi:

```
this.ResetLevel = function() {
	this.mainMenu = null;
	this.levelCompleted = null;
	this.score = 0;
}
this.LoadLevel = function(lev) {
	this.level = lev;
	this.ResetLevel();
	if(lev == 0) {
		this.mainMenu = new MainMenu();
	}
	else {
		//carica un livello di gioco
	}
}
```

Funzione	Descrizione
ResetLevel	Si occupa di azzerare tutte le variabili al cambio di ogni livello.
LoadLevel	Crea un'istanza di MainMenu se carichiamo il livello 0, altrimenti carica un livello di gioco.
Draw	Esegue clearRect per pulire la schermata del canvas dal precedente draw ed effettua il rendering del menu o del livello corrente.

Per visualizzare il nostro menu incominciamo ad implementare il metodo draw come segue:

```
this.Draw = function() {
	this.ctx.clearRect(0, 0, this.canvas.width, this.canvas.height);
	if(this.level == 0) {
		//menu principale
		this.mainMenu.Draw();
	}
	else {//disegna il livello di gioco}}
```

e modifichiamo il metodo StartGame facendo in modo che se non ci sono stati errori nel caricamento delle risorse chiamiamo il metodo di caricamento di un livello con il parametro 0 e avviamo il gameLoop.

```
function StartGame(){
	//crea un istanza di Game
```

```
        game = new Game();
    game.init();
    if(game.gr.errori==0){
                game.LoadLevel(0);
                game.GameLoop();
          }
}
```

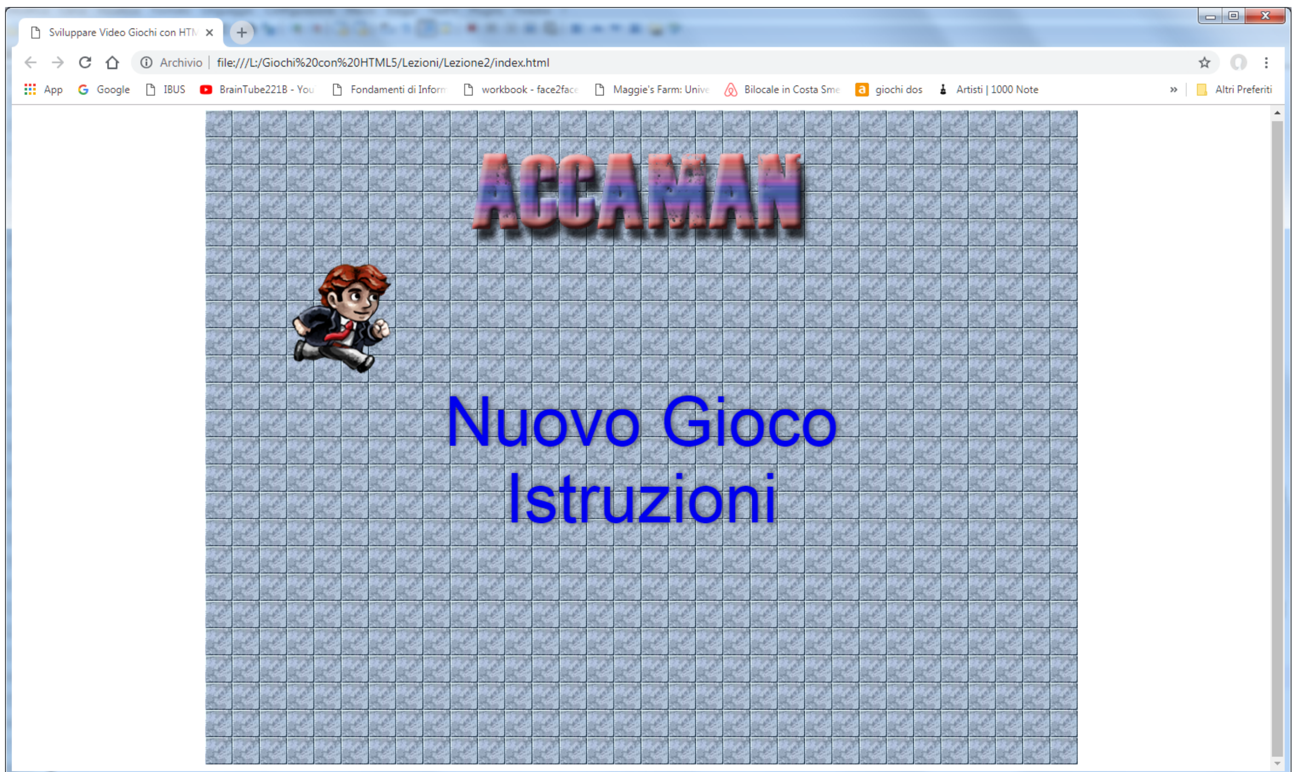
Adesso non ci resta che crearci un nuovo file javascript che chiameremo display.js nel quale creiamo l'oggetto mainMenu come segue:

```
function MainMenu() {
        game.sndMusic.loop = true;
        game.sndMusic.play();

        this.Draw = function() {
            // disegna lo sfondo
               game.ctx.save();
               game.ctx.fillStyle = game.patternMenu;
               game.ctx.fillRect(0, 0, game.canvas.width, game.canvas.height);
               game.ctx.restore();
               // mostra logo e personaggio
               game.ctx.drawImage(game.sprLogo, game.canvas.width/2 - game.sprLogo.width/2 ,
               50);
               game.ctx.drawImage(game.sprSplashLogo, 30 , 180);
               game.ctx.shadowColor = "#555";
               game.ctx.shadowOffsetX = 1;
               game.ctx.shadowBlur = 10;

               // imposta il font
               game.ctx.font = "60pt 'Arial'"
               game.ctx.textAlign = "center";
               // centro del canvas
               var cx = game.canvas.width/2;
               var cy = game.canvas.height/2;
               // disegna il menu e rileva le azioni dell'utente
               if(Inputs.MouseInsideText("New Game",cx, cy+10,"#00e", "#ea4") &&
Inputs.GetMousePress(MOUSE_LEFT)) {
                      //carica il livello 1
                      game.sndMusic.pause();
                      game.LoadLevel(1);
               }
               if(Inputs.MouseInsideText("Istruzioni",cx, cy+100,"#00e", "#ea4") &&
Inputs.GetMousePress(MOUSE_LEFT)) {
                      window.location.href = "istruzioni.html";
               }
               game.ctx.shadowOffsetX = 0;
               game.ctx.shadowBlur = 0;
        }
}
```

Aggiungiamo al file input.js la funzione:

```
Inputs.MouseInsideText = function(str, x, y, col1, col2) {
        var w = game.ctx.measureText(str).width;
        var h = 30;
        var inside = (Inputs.mouseX > x - w/2 && Inputs.mouseY > y - h && Inputs.mouseX < x +
w/2 && Inputs.mouseY < y+4 );
        if(inside) game.ctx.fillStyle = col2;
        else game.ctx.fillStyle = col1;
        game.ctx.fillText(str, x, y);
        return inside;
};
```

Se tutto è andato bene dovremmo avere come risultato :



le immagini sono relative alla mia cartella immagini volendo potete scaricare le risorse a questo indirizzo: http://prof.accarino.altervista.org/risorse.zip