

Premessa

“Game Salad tutorials – Volume 1: le basi” è un pdf contenente le versioni testuali di 10 videotutorial sulle **basi di Game Salad** pubblicati gratuitamente da Francesco Milanese sul suo canale Youtube e sul sito www.francescomilanese.com .

L'ebook, che consta di 65 pagine in formato A4, ha quindi un prezzo volutamente basso perché è rivolto a chi ha seguito i videotutorial gratuiti, li ha trovati interessanti e desidera avere un testo in pdf (con screenshots dei videotutorial) da poter consultare facilmente, mediante le funzioni di ricerca o stampandolo... oppure desidera ringraziare in maniera concreta l'autore dei videotutorial, ottenendo comunque una piccola “guida – reference” su Game Salad.

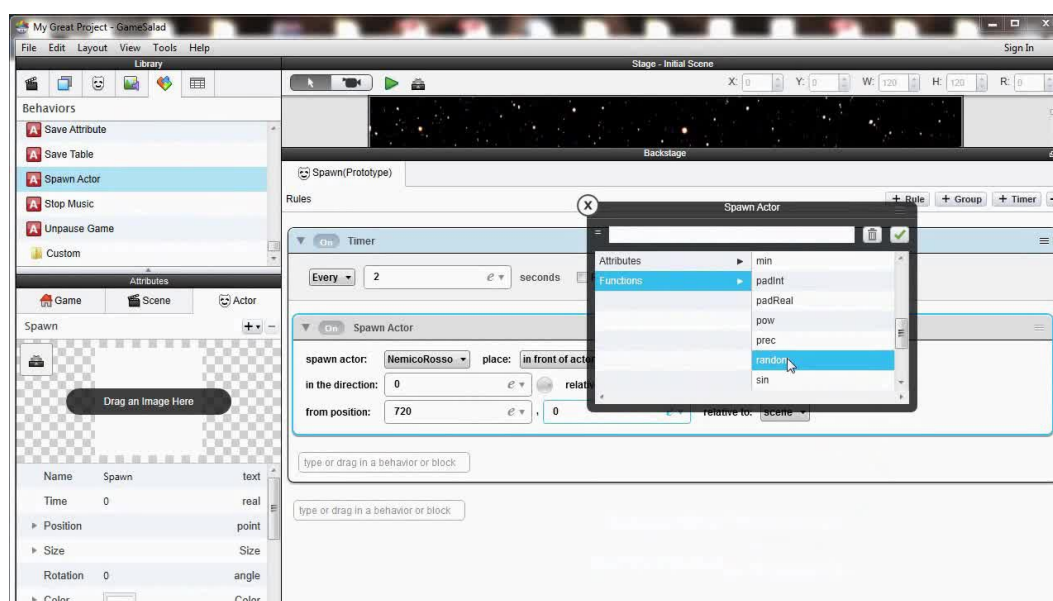
SOMMARIO

1. Introduzione	1
2. Panoramica dell'Editor di Game Salad	4
3. Le Scene: i “livelli” (o “schermate di gioco”)	12
4. Gli Actors (attori, elementi del progetto). Prototipi e istanze	19
5. I Behaviors	25
6. Assets: immagini, suoni e altri oggetti di gioco	30
7. Muovere gli oggetti (Actors)	34
8. Rilevare le collisioni tra oggetti di gioco	38
9. Compilazione e pubblicazione di un Progetto	42
10. Realizzazione pratica di un semplice livello di gioco	48

Introduzione a Game Salad - Tutorial 1

Introduzione

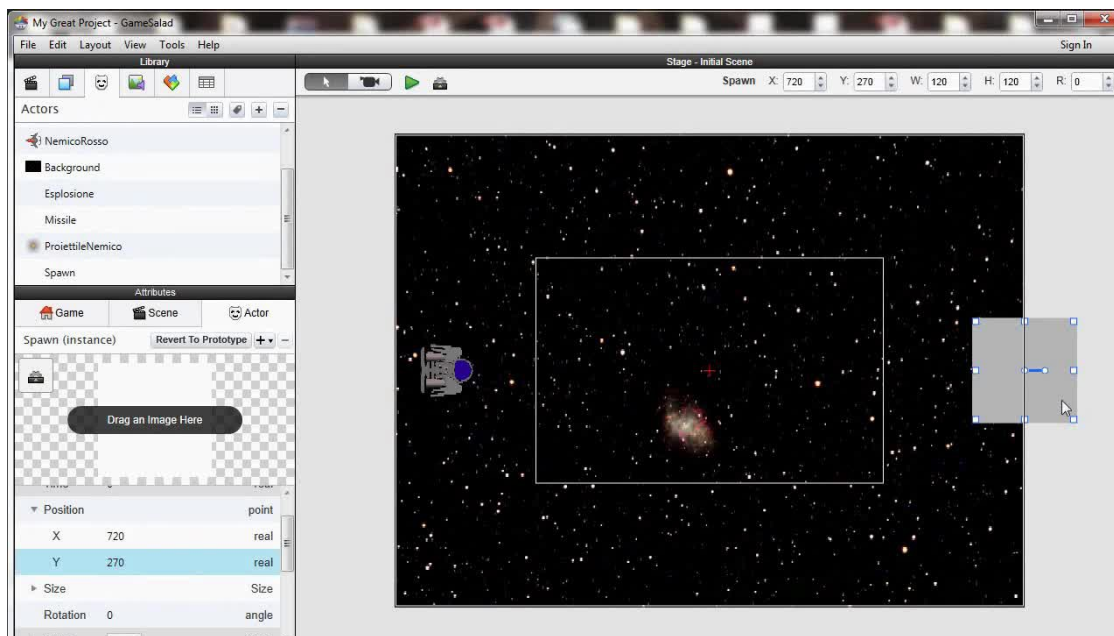
GameSalad è un programma che consente di creare **videogiochi 2D**, di tipo “**Arcade**”, per dispositivi **mobile** (iOS e Android), **html5** e **Mac OS**, il tutto mediante un unico ambiente di sviluppo (l'editor di **GameSalad**) e compilando poi i progetti per i vari dispositivi di destinazione.



Il software è **scaricabile gratuitamente**, in versione base, dal sito ufficiale; per pubblicare i propri progetti è necessario **registrarsi** al sito, cosa che può avvenire gratuitamente con la licenza base o a pagamento con la **licenza Pro**, che mette a disposizione funzionalità extra.

Per sviluppare giochi interessanti, comunque, la **licenza base** è più che sufficiente.

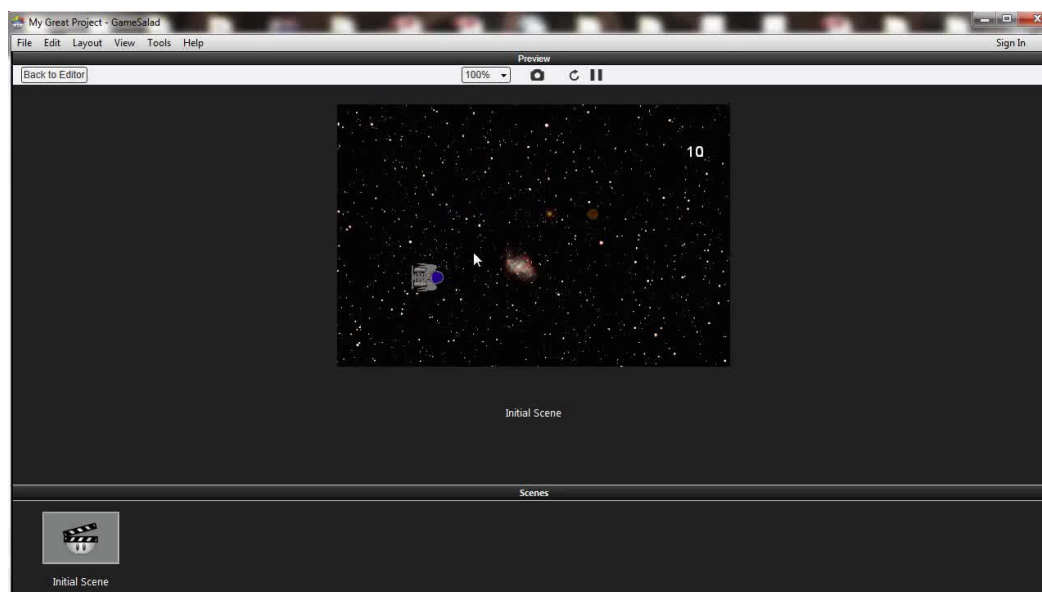
L'approccio utilizzato da **GameSalad** consiste nella definizione di “**blocchi logici**”, ossia della combinazione di oggetti, comportamenti, scene, azioni e conseguenze per realizzare gli scenari e le dinamiche di gioco.



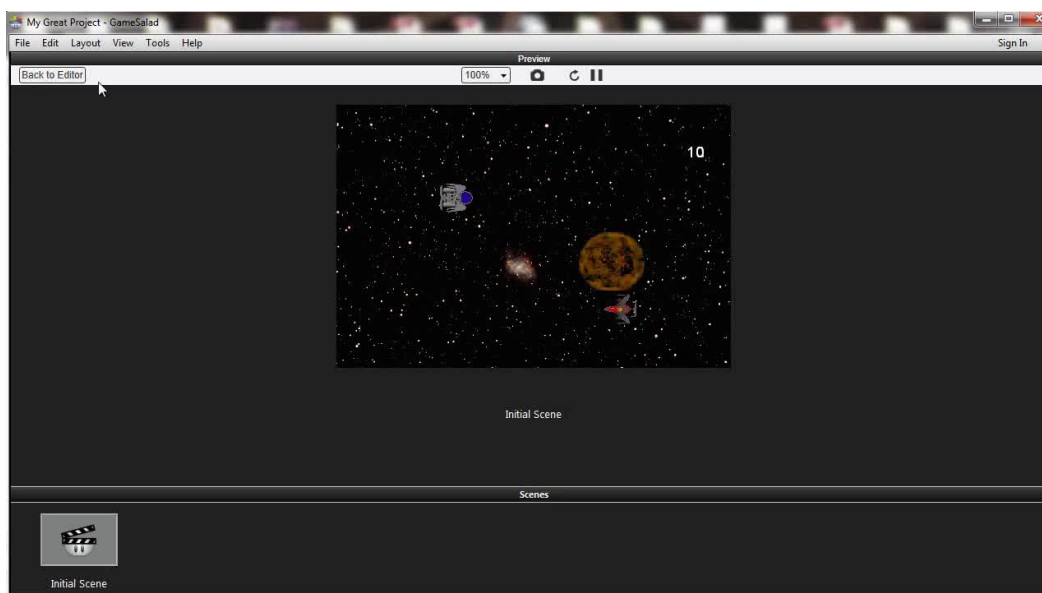
Non è necessario (e, anzi, nemmeno possibile) scrivere **nemmeno una riga di codice**, cosa che per certi versi è una limitazione notevole, tuttavia per lo sviluppo di giochi **Arcade**, come vedremo, anche con la sola combinazione di **blocchi logici** – che sono tanti – è possibile ottenere risultati notevoli.

I **giochi** realizzati mediante **GameSalad** sono tantissimi e vi basterà fare una semplice ricerca online a riguardo per convincervi che le potenzialità offerte da questo editor sono davvero enormi.

A parte l'approccio a “**mattoncini di logica**” (*logic bricks*), un'altra caratteristica notevole dell'Editor è la **modalità drag-and-drop**, che consente di definire, posizionare e collegare gli elementi, sia visivi (di gioco) che logici, semplicemente mediante click e trascinamento, in maniera visuale.



Attenzione, una premessa importante: i tutorial di questa guida introduttiva saranno, inizialmente, piuttosto teorici, nel senso che vedremo delle panoramiche dei vari elementi presenti nel programma, concentrandoci più che altro sulla sua “filosofia” di esecuzione, per cui questa serie va utilizzata come introduzione / riferimento per altri tutorial e videocorsi pratici che trattano casi concreti senza riprendere le basi, che esaminiamo qui; ad ogni modo, per non rendere questa guida puramente teorica – e per consentirvi di sviluppare un semplice livello di gioco funzionante – nell'ultimo capitolo realizzeremo un piccolo progetto, i cui screenshot sono visibili in questa pagina.



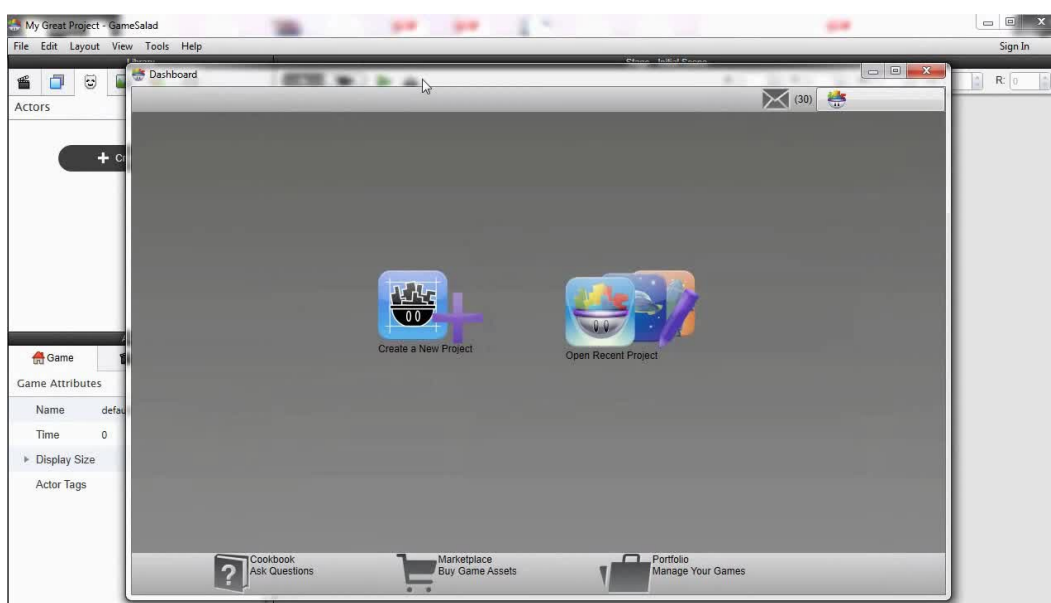
* * *

Introduzione a Game Salad - Tutorial 2

Panoramica dell'Editor di Game Salad

In questo secondo capitolo della guida introduttiva a **GameSalad** vedremo una breve panoramica dell'**Editor** del programma.

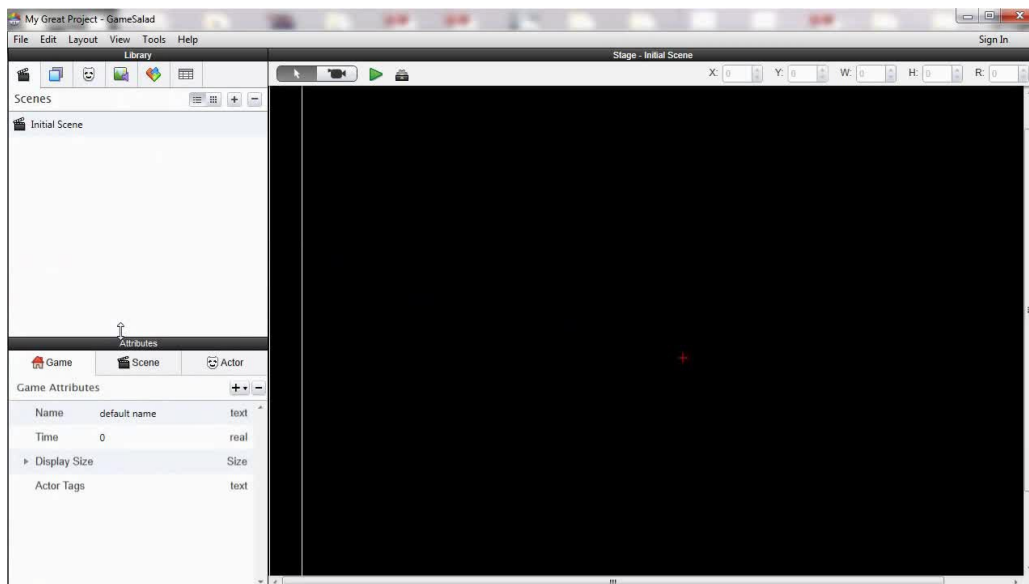
La prima cosa che vi consiglio di esaminare, almeno al primo avvio di **Game Salad**, è la **Dashboard**, richiamabile dal menù **File**.



Qui è possibile creare un **nuovo progetto** o aprirne uno **esistente**; sono presenti, inoltre, dei link al **Cookbook** (pagine di manuale, in Inglese), all'**Asset Store** (per acquistare elementi per i giochi dallo *Store di GameSalad*) e un link al “**Portfolio**”, per gestire i propri progetti.

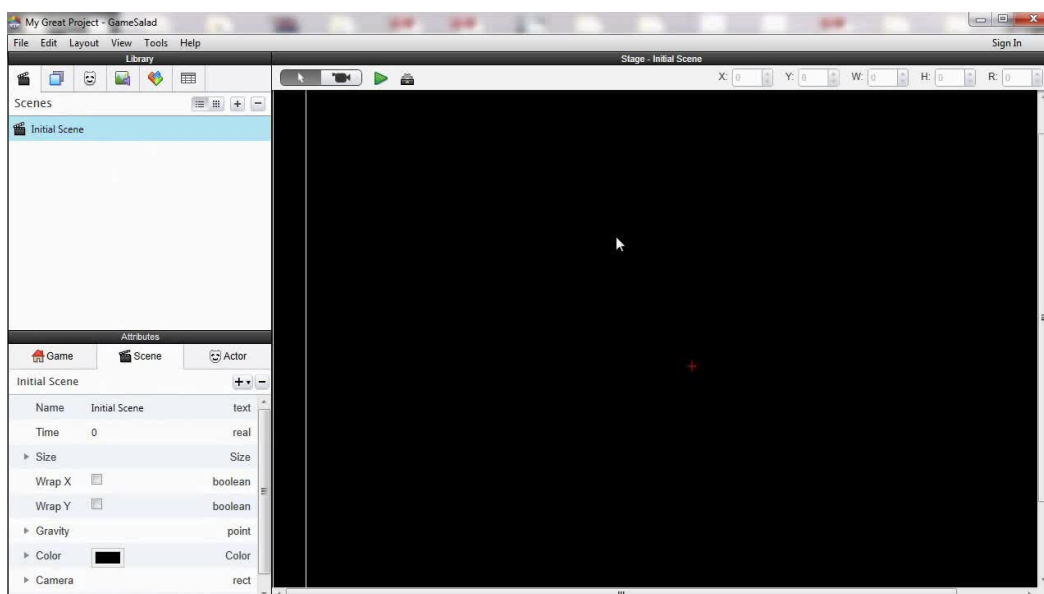
Creiamo quindi **un nuovo progetto**, ossia il gioco da realizzare.

Chiudiamo la **Dashboard** e torniamo all'**Editor**, per continuare la panoramica sull'interfaccia utente del programma.



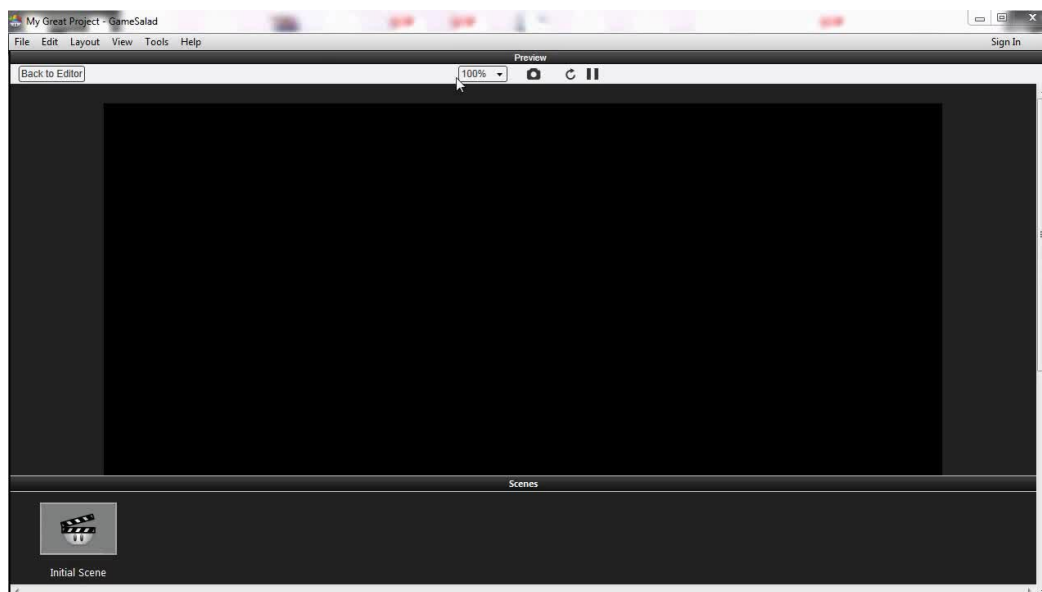
A sinistra troviamo, in colonna, le schede **Library** (la libreria degli elementi di gioco da utilizzare nel **Progetto**) e **Attributes** (che a sua volta contiene delle schede per specificare gli attributi, ovvero i **parametri e le impostazioni**, dell'intero progetto (a partire dal titolo del gioco e dalla sua icona), della scena o livello corrente e dell'elemento (**Actor**, attore attualmente selezionato)).

Un doppio click su una **scena** aprirà l'**Editor degli attributi** della stessa; idem per **attori** e altri elementi.

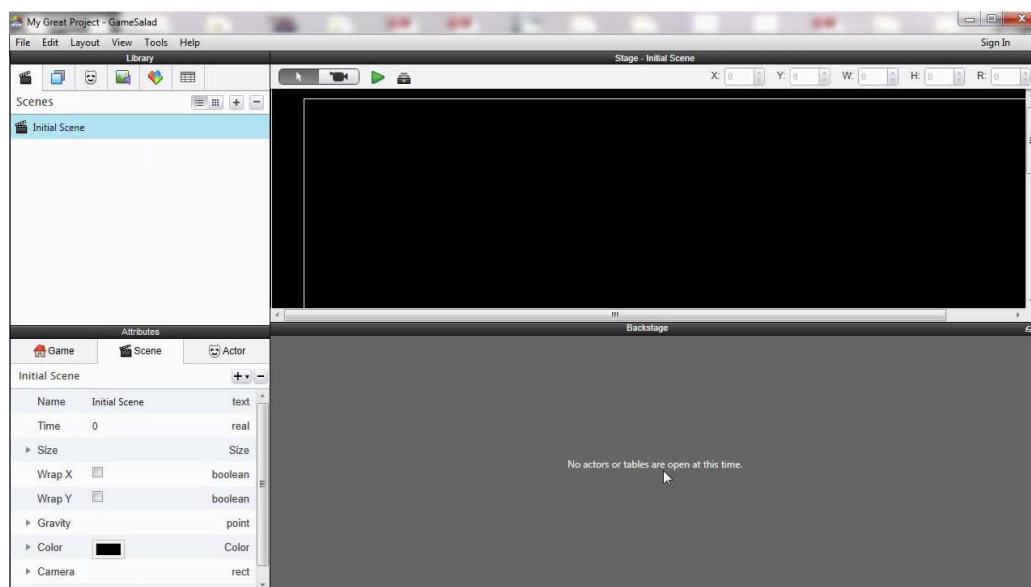


A destra troviamo lo *“stage”*, il “palcoscenico” nel quale andremo a definire i vari livelli di gioco. Qui posizioneremo gli attori e vedremo il gioco in esecuzione (in effetti, il rettangolo nero al centro rappresenta proprio **la schermata di gioco**, quando questo verrà eseguito sui vari dispositivi di destinazione).

Il tasto **Play** in alto consente di avviare la modalità gioco che, in questa fase, mostrerà solo una schermata nera vuota, come nello screenshot qui sotto.

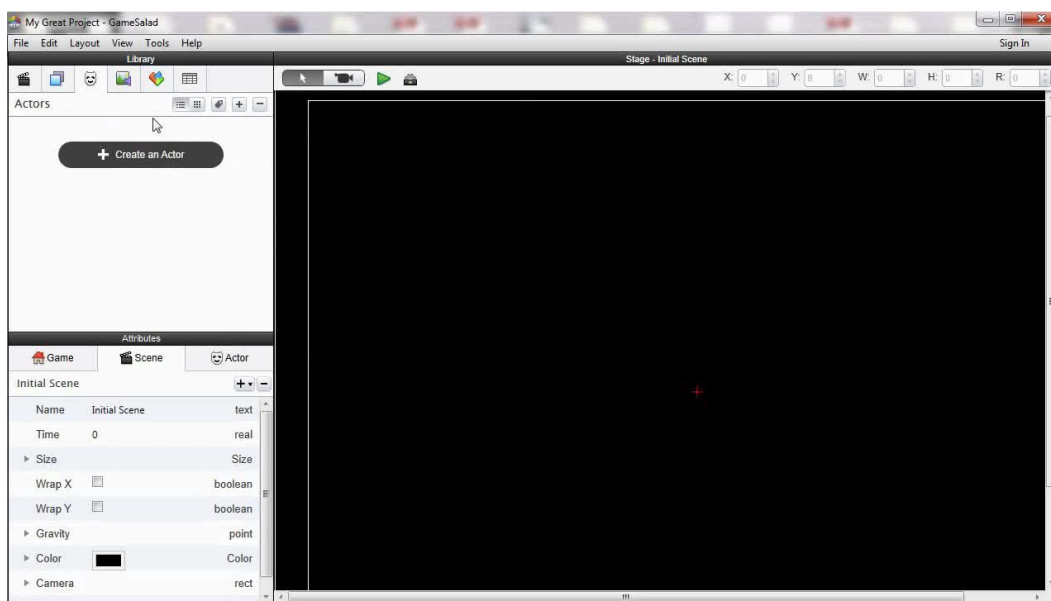


Quando si clicca su **Play** viene aperto anche il *“backstage”* (il “dietro le quinte”), per tenere traccia degli elementi presenti nel livello; il *backstage* può essere richiamato o nascosto con un click sul simbolo dell'ingranaggio, accanto al **Play**, in alto.



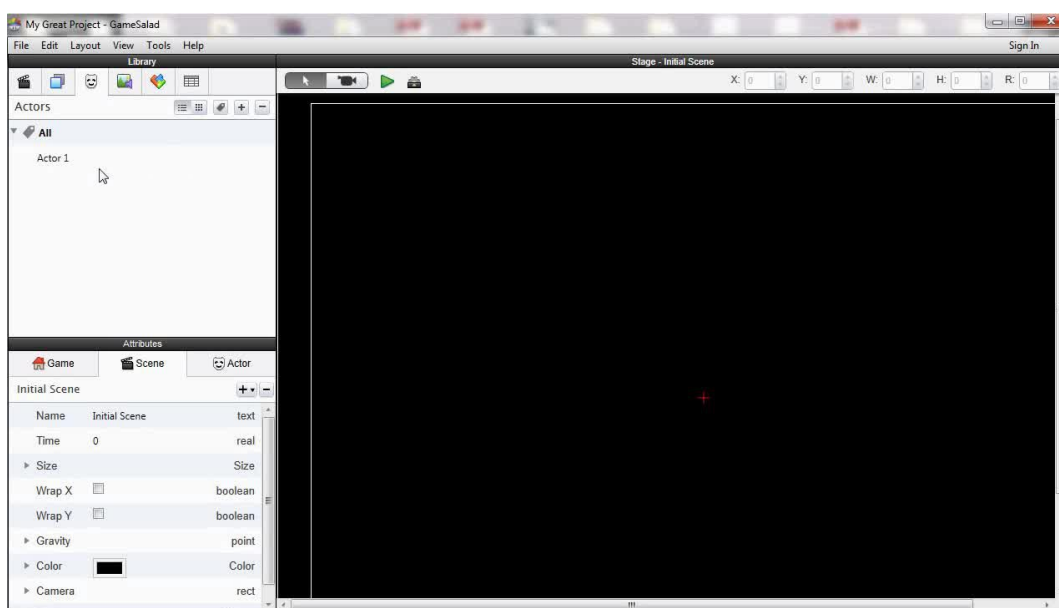
In alto a destra troviamo, infine, il tasto **“Sign In”**, per collegarsi online al proprio **account** sul sito **GameSalad**, necessario per poter pubblicare il proprio progetto, come vedremo in futuro.

A parte queste finestre, c'è quella di definizione degli **attori**, dove è possibile – dopo aver definito un **Actor**, ossia un **elemento di gioco** – specificare i vari componenti e mattoncini logici che ci consentiranno di realizzare azioni, reazioni e comportamenti dei vari oggetti.



Senza andare troppo in dettaglio in questa prima fase, diamo un'occhiata a questa schermata.

Creiamo un primo **attore** con il + (o la scritta **“Create New Actor”** o simili) nella **Library**, che rappresenta la collezione di tutti gli elementi presenti nel nostro progetto.



Quello appena creato è un **PROTOTIPO di attore**.

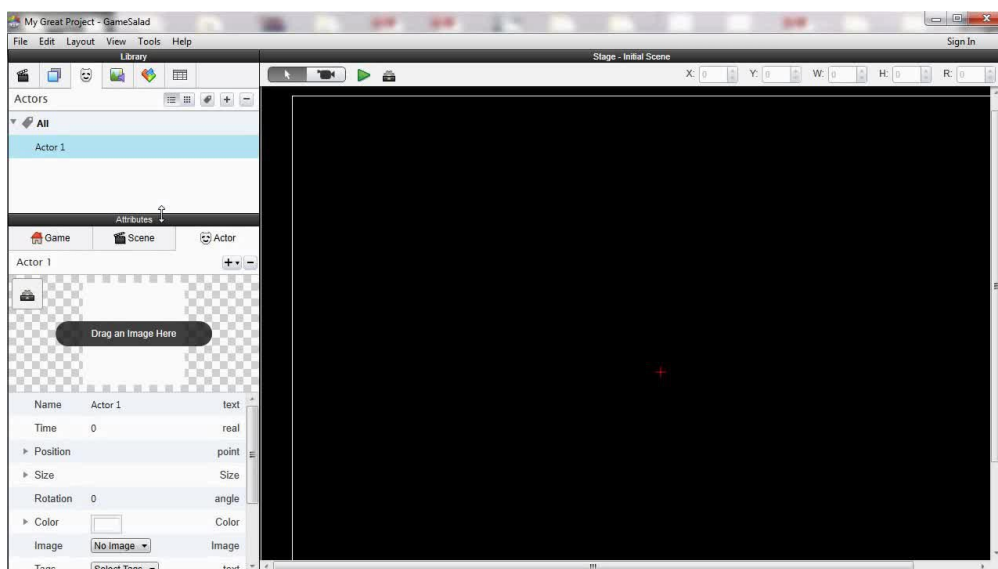
Il concetto di **prototipo** e **istanza** è comune in Informatica, Computer Grafica e nei motori di gioco: il prototipo è, appunto, un modello di riferimento, in un certo senso **uno “stampo”**, dal quale si ricavano poi le copie concrete, dette “**istanze**”.

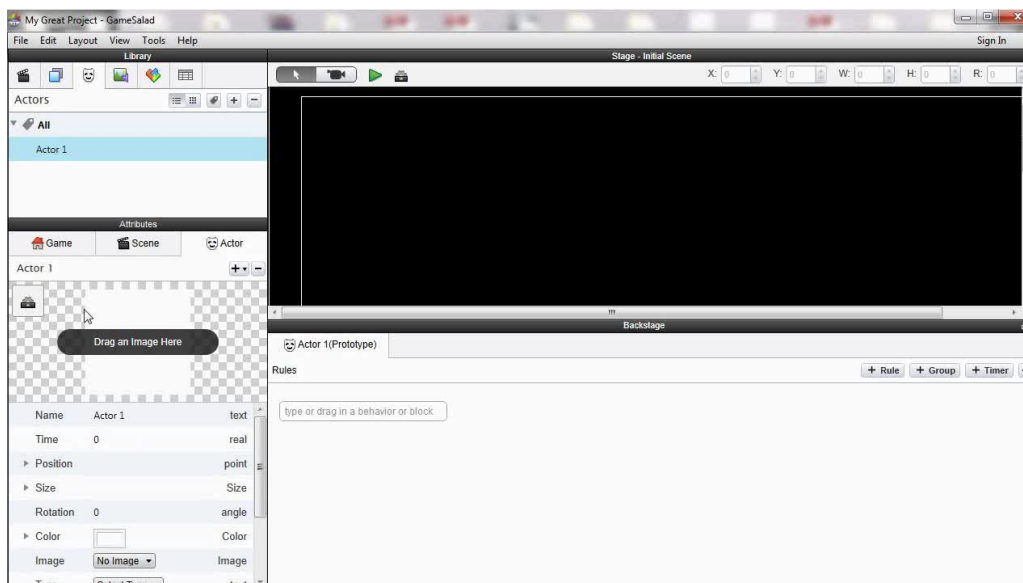
Le istanze sono collegate al prototipo, nel senso che modificando il prototipo – o definendo per tale elemento comportamenti, attributi, eccetera – ritroveremo le modifiche anche nelle istanze, il che rappresenterà, come vedremo, un notevole risparmio di tempo per apportare delle modifiche agli elementi di gioco.

Sui prototipi e le istanze torneremo più volte ma diciamo subito che, anche se **un'istanza** avrà le proprietà del suo prototipo, **può anche avere altre regole**, proprietà, eventi e reazioni proprie, diciamo “personali”, senza condividerle con altre, per cui le istanze sono “copie collegate” di un prototipo, sì, ma con un certo grado di autonomia (e in ogni caso non dobbiamo dimenticare che **le istanze sono elementi concreti**, inseriti nei livelli, mentre i prototipi sono appunto degli “stampi” generici, non attivi in un livello al momento della creazione).

Abbiamo quindi creato un prototipo di **Attore**.

Apriamo la scheda **Attributes – Actor**: notiamo alcuni campi e impostazioni intuitivi, come nome, posizione ed altri parametri iniziali; notiamo, inoltre, l'icona di un ingranaggio: cliccandoci sopra, si aprirà a destra il **Backstage**, relativo questa volta al **Prototipo**, dove potremo inserire i “**Behaviors**” (letteralmente: comportamenti), ossia **eventi, azioni ed operazioni**.





Questi eventi sono detti “**Rules**”, regole: in sostanza, una regola definisce **una serie di eventi** in base ai quali si deve svolgere **un'azione** (es.: il giocatore ha premuto la barra spazio) e le azioni da effettuare (es.: il personaggio salta).

Tutto ciò rappresenta, in pratica, quanto detto nel capitolo introduttivo, ossia: la definizione di un **progetto** (il gioco da realizzare), composto da varie **scene** (i livelli) e, al loro interno, la definizione della partita mediante **attori, comportamenti, azioni e reazioni**, che vengono messi in relazione tra loro specificando delle **Rules** che combinano dei mattoncini logici (gli stessi attori e i **Behaviors**, i comportamenti e le azioni da eseguire) per dare vita al gioco.

Questo capitolo riguarda una prima panoramica dell'interfaccia, per cui ci fermiamo qui; nel prossimo parleremo un po' più in dettaglio delle **Scene** (i livelli di gioco) in **Game Salad**... prima di chiudere, però, due parole sull'**Aspect Ratio**.

L'**Aspect Ratio** è un rapporto tra larghezza e altezza della visualizzazione delle **Scene**.

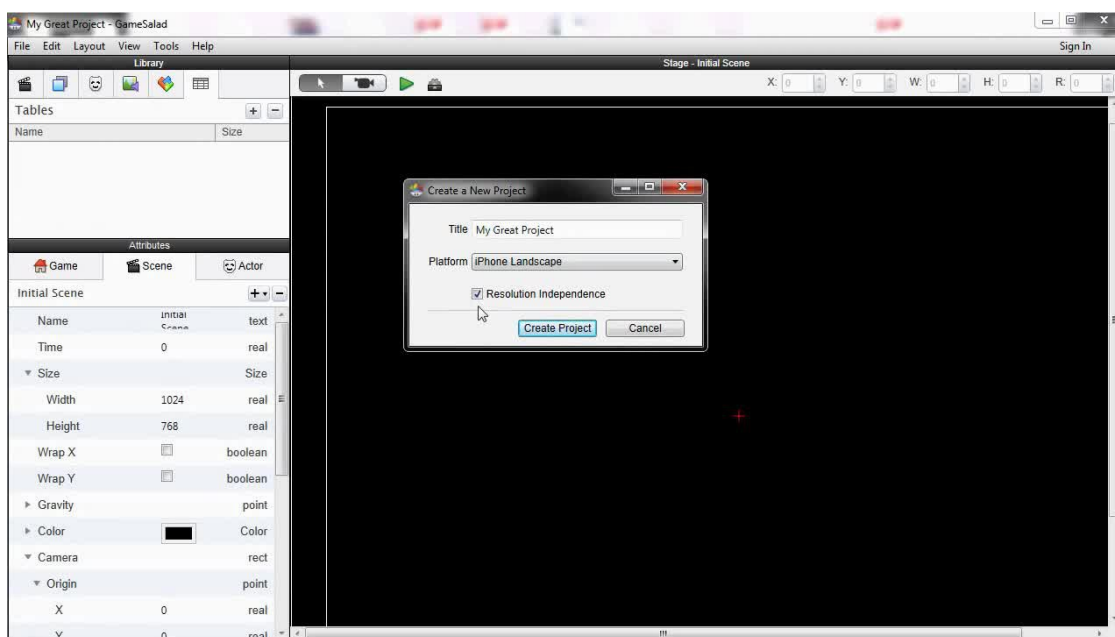
Per via della gran quantità di dispositivi mobile esistenti, ciascuno con le proprie dimensioni per i display, è necessario prevedere delle modifiche nella visualizzazione dell'area della scena, per realizzare un'esperienza di gioco quanto più uniforme possibile anche su **dispositivi molto diversi tra loro**.

In **GameSalad** è presente un parametro, *Aspect Ratio*, che mette a disposizione due opzioni per gestire formati di visualizzazione differenti: *Overscan* e *Letterbox*.

Overscan adatterà la scena al dispositivo allargandola al massimo ed eventualmente tagliando via (cropping) parti di scena qualora una delle due dimensioni dovesse eccedere l'area di visualizzazione; in sostanza, il centro della scena sarà uguale per tutti, ma a seconda dei dispositivi potrebbero esserci aree “tagliate fuori”, letteralmente, sopra e sotto oppure a sinistra e destra (ossia, nel senso verticale o in quello orizzontale).

Il metodo *Letterbox*, al contrario, restringe la scena per farla entrare in entrambe le direzioni nell'area di visualizzazione del display e riempie le eventuali parti non coperte (sopra e sotto oppure a destra e sinistra) con il colore di sfondo (*Background*) della Scena.

Va detto comunque che non sempre si deve arrivare a queste due soluzioni: **quando si crea il progetto**, infatti, si indica il dispositivo (*device*) di destinazione (ad esempio, Kindle o iPhone5); in questo caso, **GameSalad** adatterà l'area di gioco per farla coincidere con quella del dispositivo selezionato, per cui non vi sarà né **Letterbox** né **Cropping**. Queste soluzioni servono infatti solo quando si deve portare il gioco su un device che non è quello specificato nel **Project** (e che, quindi, avrà verosimilmente **un altro Aspect Ratio**). In casi estremi, la soluzione migliore è preparare un nuovo progetto proprio per il nuovo tipo di dispositivo.



Attenzione, una nota finale per questo capitolo: l'area di visualizzazione della **Telecamera** di gioco non deve coincidere per forza con tutto il **Display Size**, tuttavia questa è la soluzione adottata, in genere, per gli Arcade, anche per evitare ulteriori problemi di **Aspect Ratio** e di confinamento dell'area inquadrata dalla Telecamera nel Display... è, quindi, un argomento che non tratteremo in questa guida base, ma è bene conoscere **la distinzione tra Main Camera Area e Display Area**.

* * *

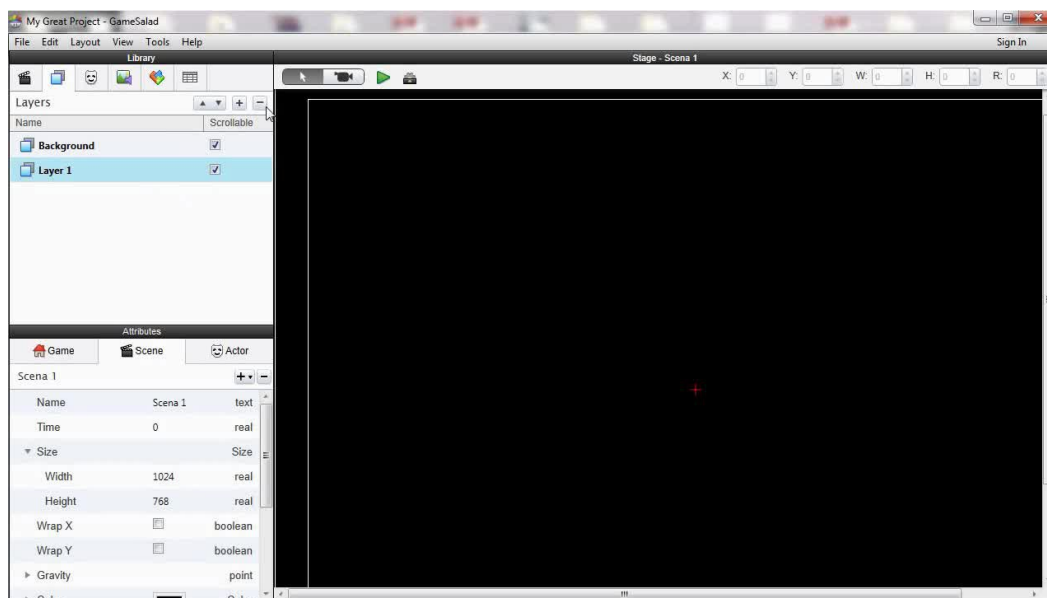
Introduzione a Game Salad - Tutorial 3

Le Scene: i “livelli” (o “schermate di gioco”)

In questo terzo capitolo sulle basi di **GameSalad** parleremo delle **Scene**, che in un progetto rappresentano i “livelli” o le “schermate” del gioco.

Le **Scene** sono, sostanzialmente, “schermate”, nel senso che anche la pagina iniziale, il menù, le opzioni, la classifica e i credits vengono realizzati mediante Scenes.

Le **Scene** consentono quindi di suddividere il progetto (il gioco) in pezzi, raggruppando tra l'altro gli elementi presenti nelle varie schermate (gli oggetti di gioco, detti “**attori**”).

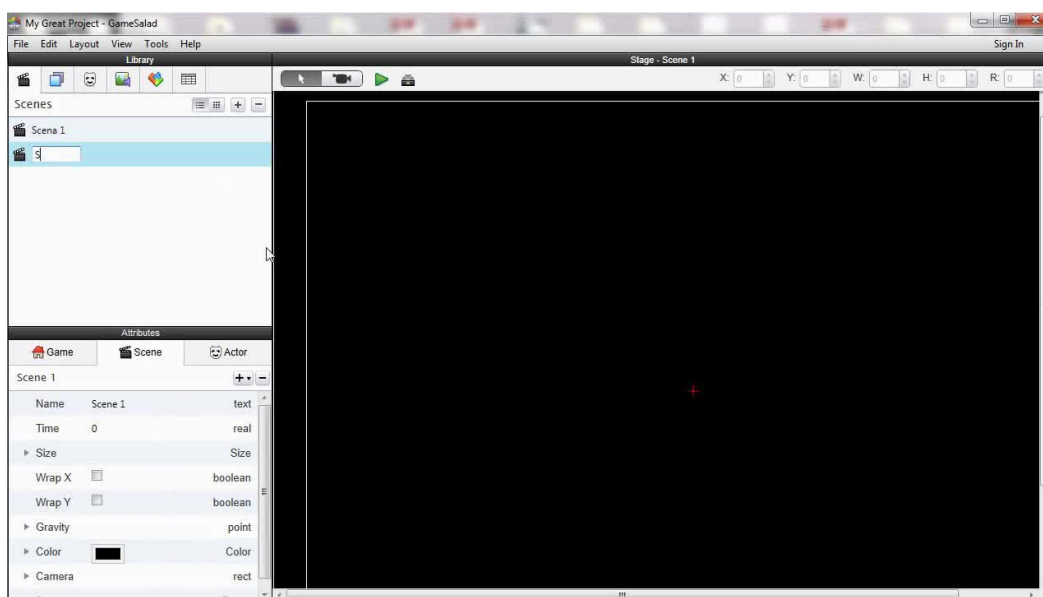


Non solo: all'interno di una scena è possibile effettuare una suddivisione in **Layers**, “livelli”, come quelli di Photoshop o altri programmi; pensate quindi a dei fogli semitrasparenti sovrapposti,

ciascuno con un elemento di gioco, per cui è possibile “mettere dietro” o “in primo piano” certi elementi piuttosto che altri, cosa utile ad esempio **per separare** lo sfondo dai protagonisti, dalle scritte o icone dell'interfaccia in sovrimpressione, eccetera.

L'elenco delle scene di un progetto si trova nella scheda “**Scenes**” della **Library**; di base, ce n'è almeno una, vuota.

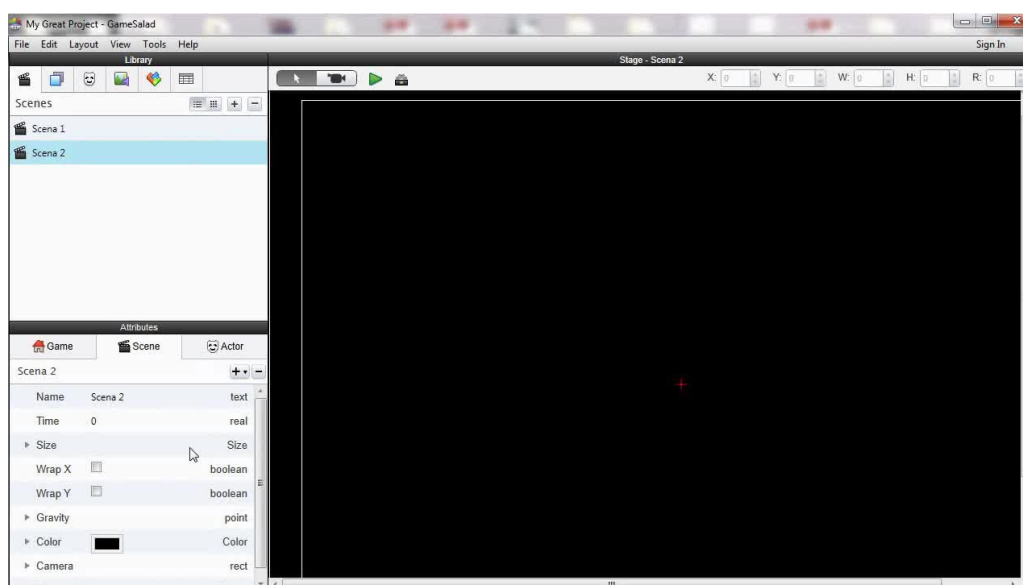
Per aggiungere una scena, cliccare sul + della scheda **Scenes**, in **Library**; per rimuoverla, selezionarla dall'elenco e cliccare sul – o premere CANC; per duplicarne una, selezionarla e cliccare sul + mentre si tiene premuto ALT.



Nella sezione **Attributes** troviamo vari parametri per la **Scena** attualmente selezionata; esaminarli qui uno per uno, senza esempi pratici, sarebbe alquanto noioso e probabilmente inutile, per cui vediamo solo le voci principali (tratteremo le altre nei prossimi capitoli, con esempi concreti):

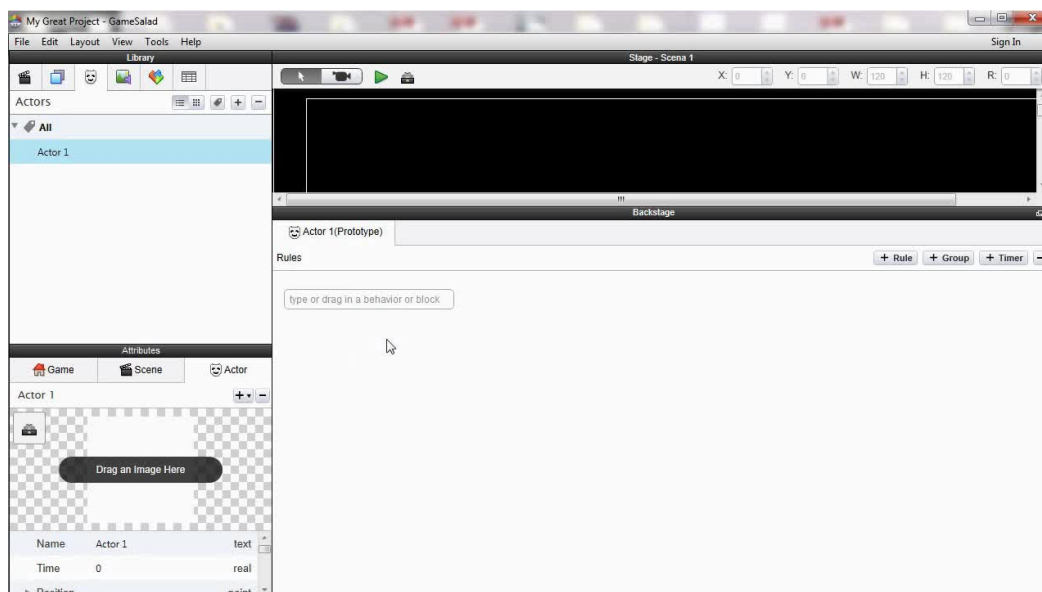
- **name**: il nome della scena, utile soprattutto in via di sviluppo per dare nomi significativi agli ambienti (es.: schermata iniziale, high scores, livello 1, ecc..);
- **time**: indica il tempo trascorso dall'apertura o attivazione della scena; è un parametro che può essere letto da altri elementi del gioco (ad esempio, per far partire un timer o lanciare un evento ad un dato istante), ma non può essere modificato o sovrascritto;
- **size**: la dimensione, in pixel, della scena (larghezza per altezza);

- **wrap** (*x* e *y*): se attivati, quando un Actor uscirà da un lato della schermata riapparirà dall'altro (*wrap* sta per “avvolgi”: in pratica, si collegano il lato destro al sinistro e quello superiore a quello inferiore, altrimenti gli Actor usciranno a volontà dall'area di gioco);
- **gravity**: il valore di gravità della scena, per attirare gli Actors (soggetti a gravità) verso il basso, impostando un valore per *y* (verticale) oppure per dar loro una spinta, quindi un moto con accelerazione, in altre direzioni, visto che è possibile specificare valori anche negativi per *x* (orizzontale); il valore di default è 0;
- **colour**: il colore di sfondo, in mancanza di immagini;
- **auto-rotate**: rotazione automatica della scena se cambia l'orientamento del dispositivo.

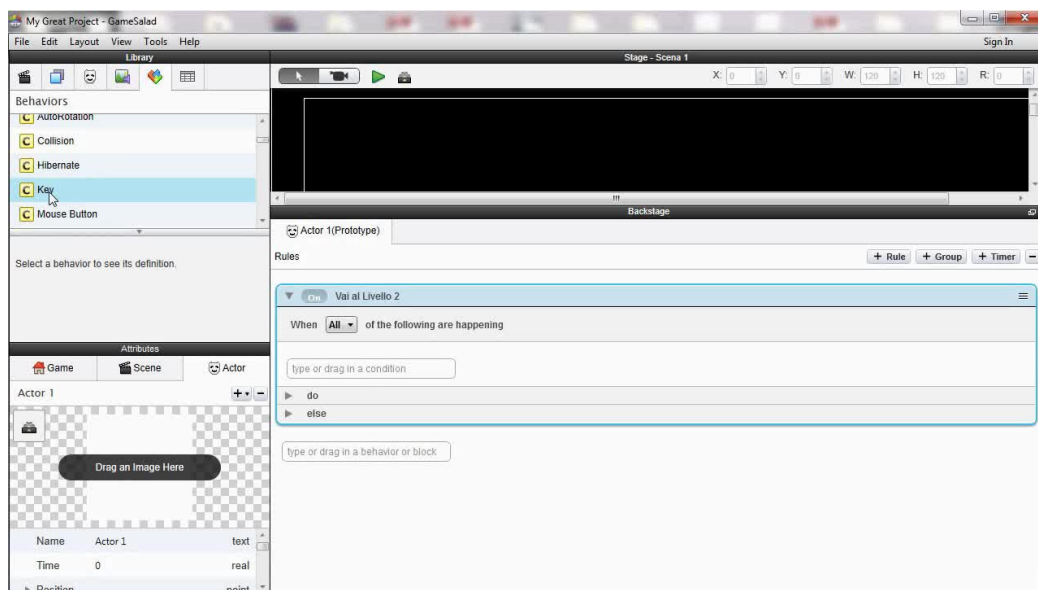


Ci sono diversi **Behaviors** legati alle **Scenes**; per mostrarne uno – e per mostrare, nel mentre, un esempio di **creazione di Rules**, le regole, specificando eventi ed azioni – procediamo in questo modo:

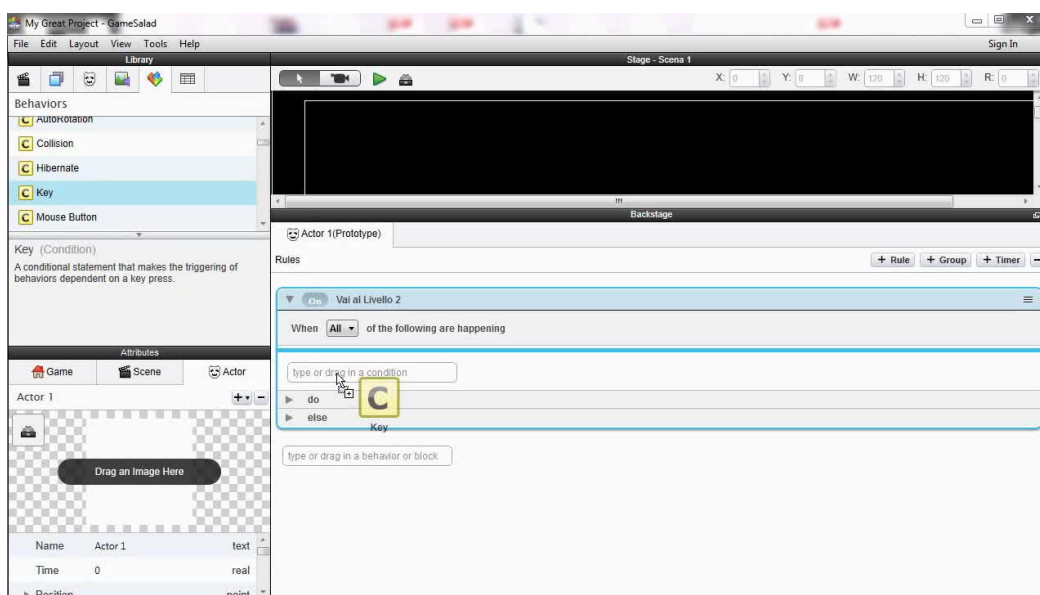
1. nel pannello **Scene**, creiamo una seconda Scena (ad es., “Scena 2”), quindi facciamo di nuovo doppio click su *Scene 1* per tornare lì (il nome della scena corrente si trova in alto al centro nella finestra di *Preview* del livello);
2. creiamo un Actor vuoto dalla sezione **Actors** della *Library*;
3. nella sezione **Attributes** dell'Actor, clicchiamo sul simbolo dell'ingranaggio per aprire il *backstage* del prototipo (immagine nella pagina successiva);



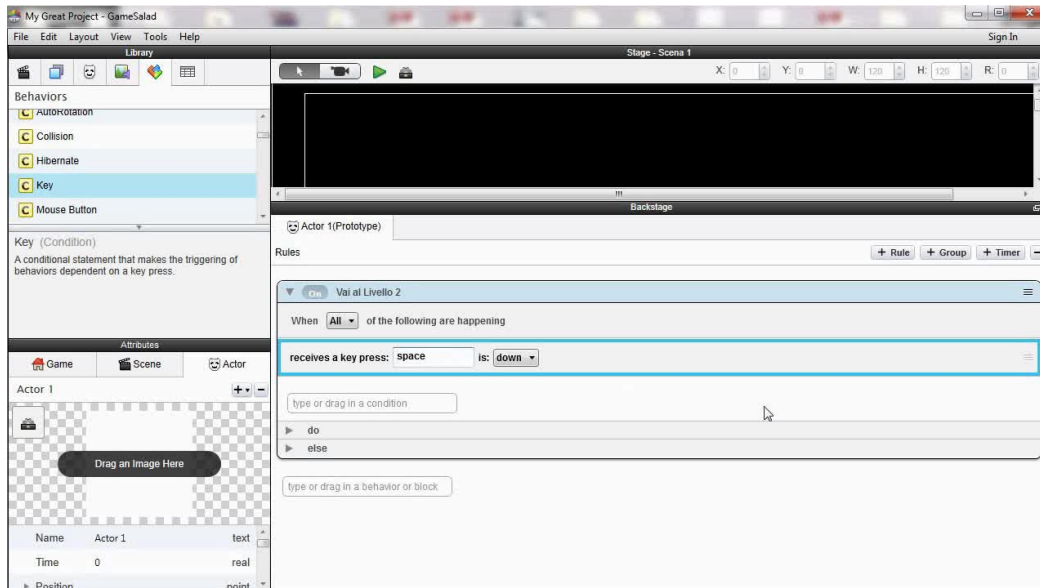
4. nel **backstage**, clicchiamo su “+ Rule” per aggiungere una Regola, ovvero una serie di cause ed effetti: avremo una regola vuota, che possiamo rinominare ad esempio in “vai al livello 2” con doppio click sul titolo e scrivendo quanto detto;



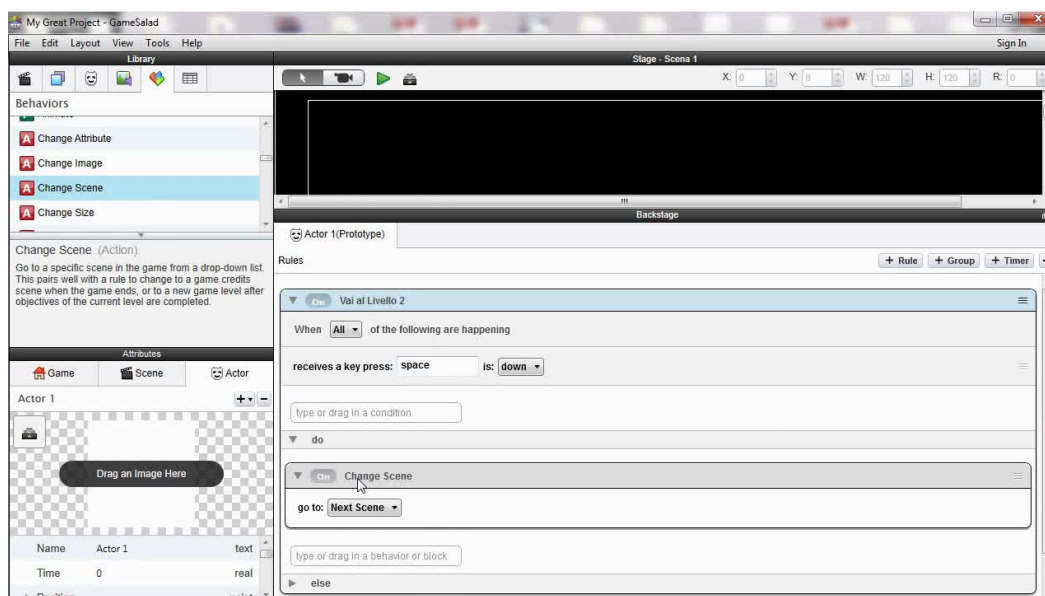
5. nella parte alta della regola, notiamo il menù “**When All**”, ovvero: esegui quello che segue quando si verificano le condizioni specificate qui sotto; dalla scheda *Library - Behaviors*, quindi, cerchiamo la sezione *C (Conditions, Condizioni)*, selezioniamo “**Key**” (tasto) e trasciniamola nella sezione **When** della **Rule**...



6. ... e notiamo che ora abbiamo *“receives a key press”*, un campo vuoto e un menù impostato su *“Down”* (tasto premuto); facciamo click sul campo vuoto e premiamo la barra spazio, ottenendo *“space”*, ossia: *“alla pressione della barra spazio”*;



7. adesso apriamo la scheda dei *Behaviors*, nella *Library*, e trasciniamo un *Behavior “Change Scene”* (Cambia Scena) nella sezione *“Do”* (“fai”) della regola, impostando come scena di destinazione la *Scena 2*.



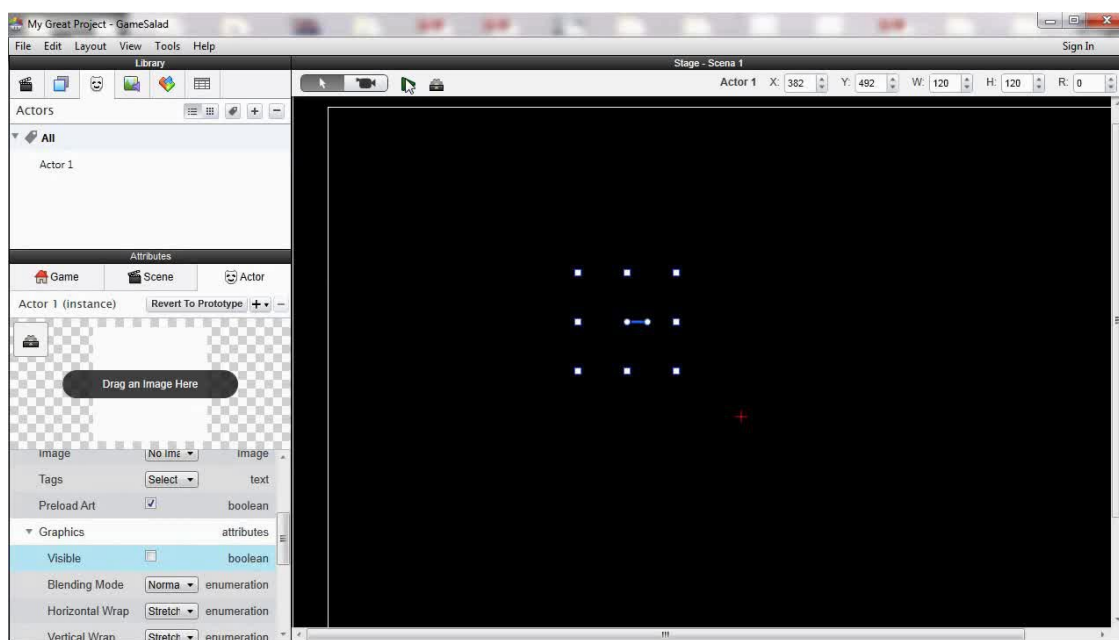
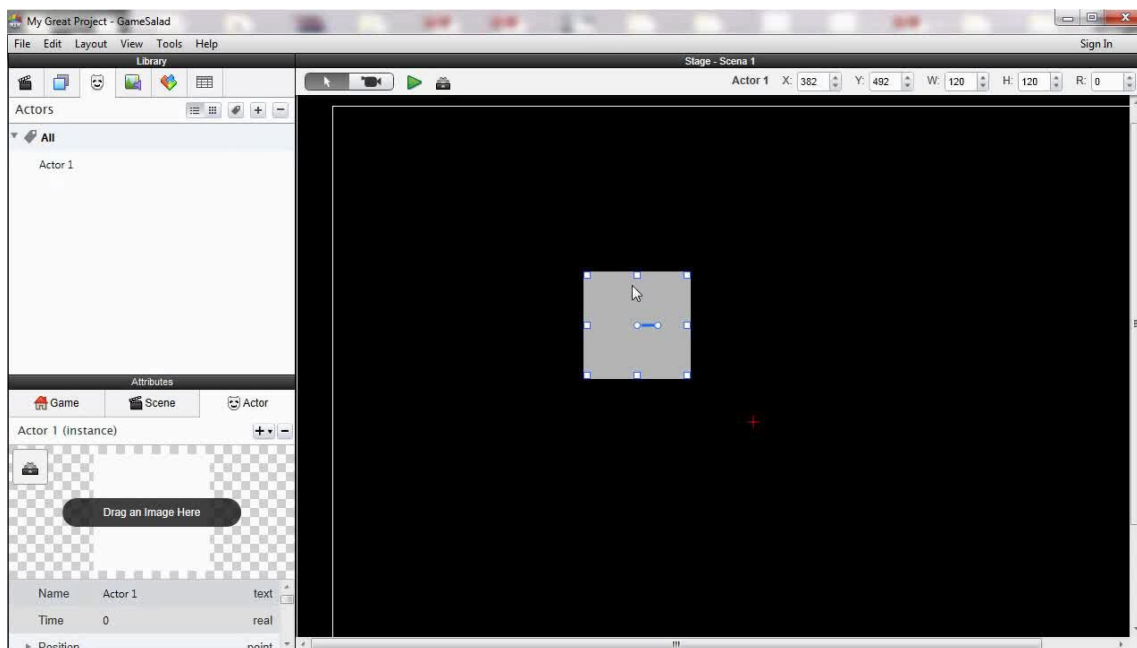
Ricapitolando: l'attore è dotato di una **regola** che intercetta la pressione della barra spazio (**evento**: “*When*”) e come **azione** (“*Do*”) cambia la scena.

Questa è, in generale, la “filosofia” di **Game Salad**, descritta già diverse volte – anche se, finora, in maniera alquanto teorica: **attori**, **regole con eventi** di attivazione, **azioni** (i *behaviors*).

Attenzione, però: **il prototipo dell'Actor** c'è, ma non una sua **istanza**, ovvero: non c'è, nella scena, **un oggetto concreto** in grado di intercettare la pressione della barra spazio e di effettuare l'azione collegata.

Prima di premere **Play**, quindi, selezioniamo l'*Actor* dalla *Library* e trasciniamolo nella *Scena*; a questo punto, nella scheda *Attributes* apriamo la sezione “**Graphics**” e deseleggiamo “**Visible**”: l'istanza è invisibile, non è raffigurata da immagini o altro, ma c'è, per cui ora possiamo premere **Play** e, subito dopo, la barra spazio.

Nei due screenshot riportati nella prossima pagina vengono mostrate le due operazioni appena descritte, ovvero: **istanziamento di un Actor** (prima immagine) e passaggio alla modalità “invisibile” (deselezionando **Visible**) per tale oggetto all'interno della scena.



Sotto l'area della schermata noteremo la scritta “*Scene2*”: siamo passati, in effetti, alla seconda scena... beh, visivamente il risultato non è entusiasmante (una schermata nera), ma quello che ci interessa – la logica – c'è e implementarla è semplicissimo!

* * *

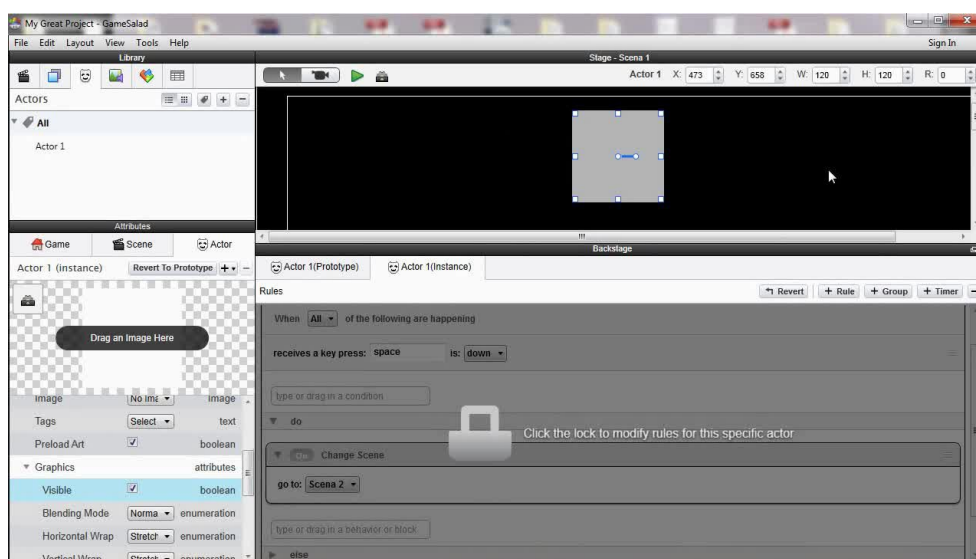
Introduzione a Game Salad - Tutorial 4

Gli Actors (attori, elementi del progetto). Prototipi e istanze

Gli **Attori** costituiscono tutto ciò che c'è all'interno di una scena o livello di gioco.

Come visto nel capitolo sulle **Scene**, un attore può essere anche **invisibile** e contribuire all'esecuzione del gioco **intercettando** ad esempio gli **eventi** da tastiera o touch, oppure **gestendo** un Timer, il punto di rinascita dei nemici ed altro ancora.

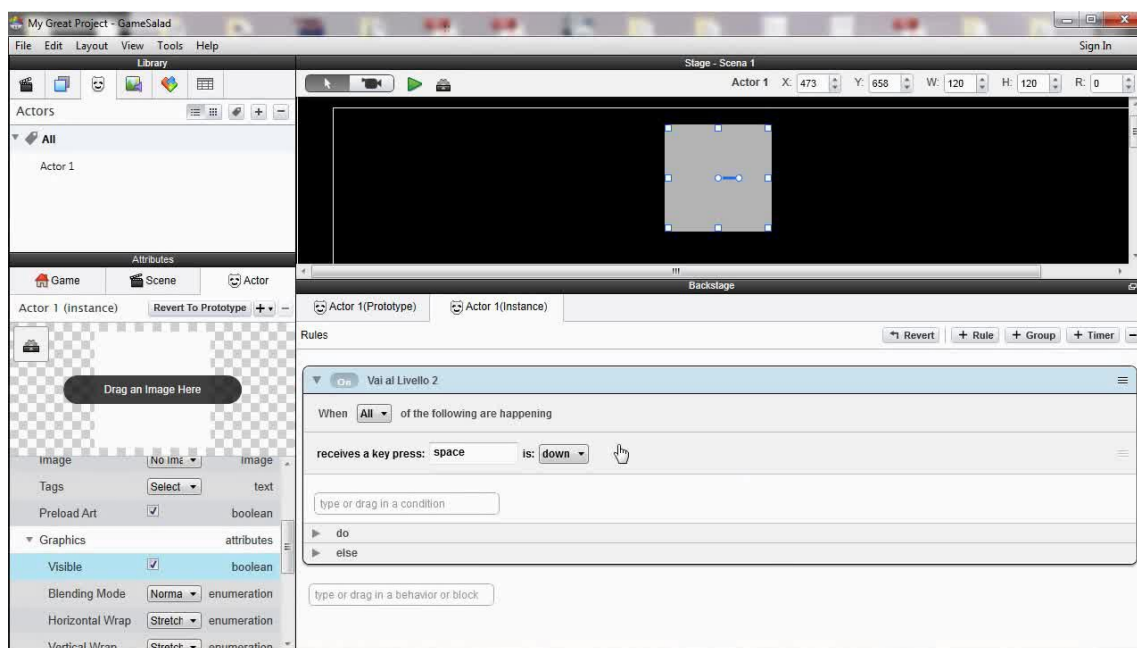
Un altro concetto già espresso, ma che riprendiamo, è la distinzione tra **Prototipo** e **Istanza**: un *Actor* definito nella *Library* è un *Prototipo* di attore, con caratteristiche, regole e parametri propri; quando poi si inserisce un *Actor* nella scena, sia mediante trascinamento che dinamicamente, cioè creandolo durante il gioco (pensate, ad esempio, a proiettili, oggetti da raccogliere, nemici, eccetera), in quel caso si ha *l'istanza di un Actor*, che ha le stesse proprietà, regole e caratteristiche dello “stampo” originale.



Un Actor inserito nella scena

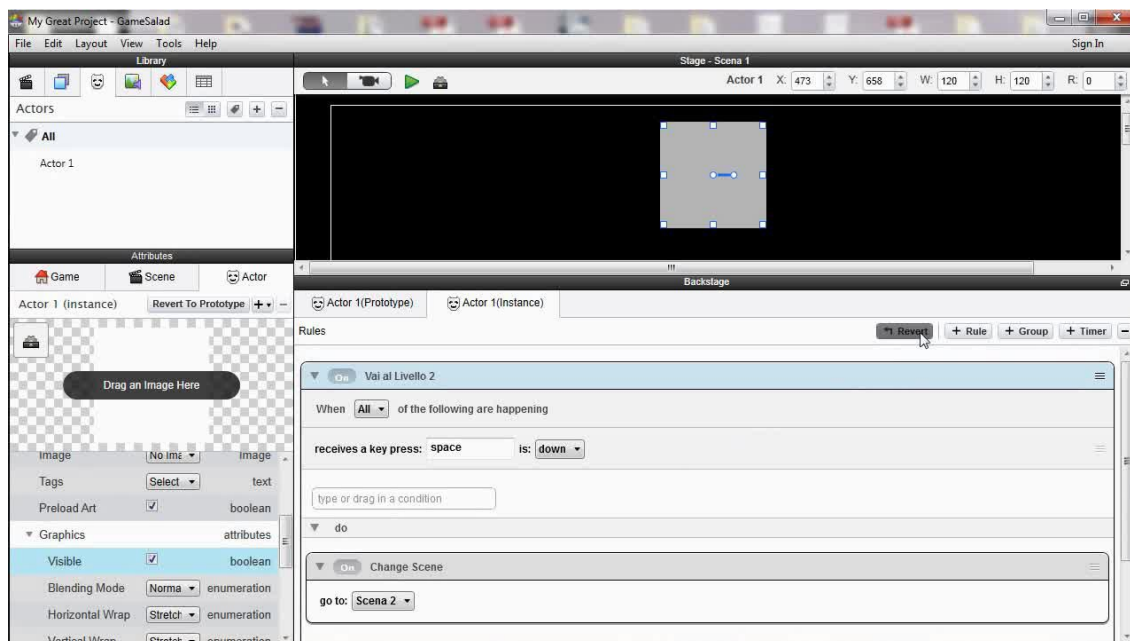
Quando si seleziona un *Actor* e si clicca, nella sezione **Attributes**, sull'icona dell'ingranaggio, nel *backstage* appariranno le **Rules** e le proprietà del prototipo, mentre se si inserisce un prototipo nella scena, creando un'istanza concreta, e si passa al *backstage*, notiamo **la presenza di un lucchetto** (come nell'immagine nella pagina precedente) e di una scritta che ci informa che quell'attore deriva da un prototipo, quindi eredita le caratteristiche del prototipo, e che, se si desidera fare delle **personalizzazioni**, si dovrà annullare il collegamento col prototipo (*“aprendo il lucchetto”*), nel senso che eventuali modifiche fatte al prototipo non verranno più applicate anche all'Actor concreto, che adesso non sarà più un'istanza di prototipo ma **un elemento di gioco a sé stante**.

L'immagine seguente mostra il *backstage* di un'istanza per la quale è stato “aperto il lucchetto”, rimuovendo il collegamento delle proprietà dell'istanza con quelle del suo prototipo.

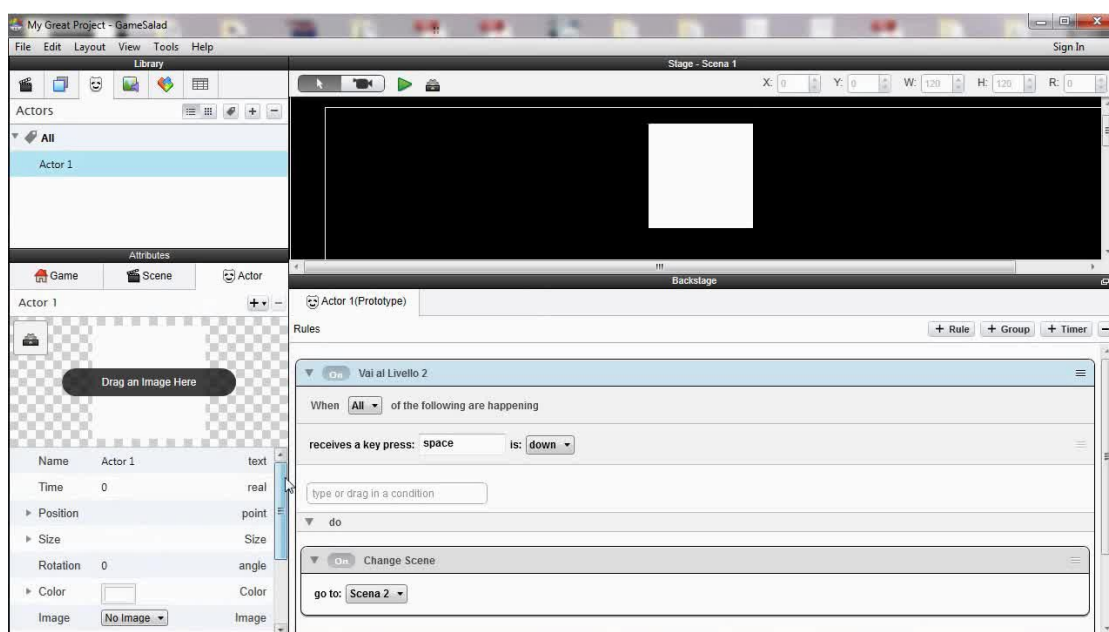


Le modifiche fatte su un prototipo hanno effetti sulle sue istanze, ma non vale il viceversa.

È possibile, comunque, **ripristinare il collegamento** di un oggetto con un Prototipo, per farlo tornare un'istanza collegata, cliccando su **Revert to Prototype** (il pulsante si trova in alto a destra nel *backstage* di un oggetto, come mostrato nell'immagine nella pagina seguente).

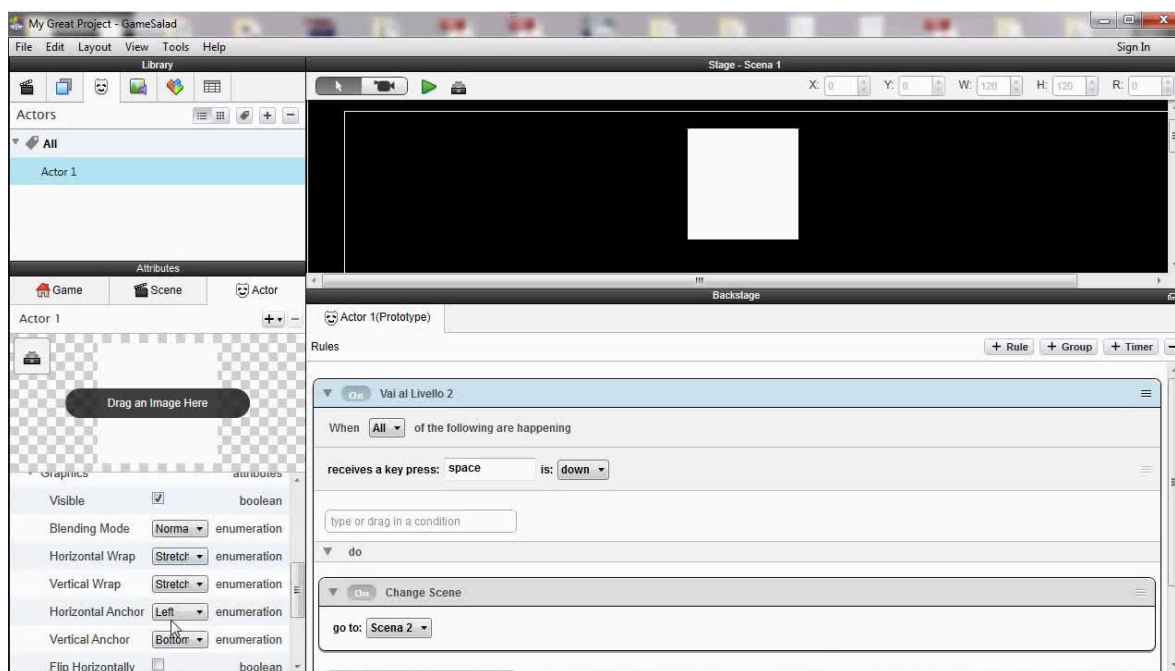


Per ogni **Actor** – sia prototipo che istanza – sono disponibili diversi campi e parametri nella scheda **Attributes**; in tutti i casi, i nomi lasciano intuire le funzioni: nome, posizione, dimensioni, rotazione, tempo (ovvero il tempo trascorso dall'apparizione dell'oggetto nella scena; utile ad esempio per l'autodistruzione dopo n secondi, con una semplice **Rule**), immagine da utilizzare, eccetera... chiaramente, in seguito opereremo su questi campi con casi concreti, tra l'altro mediante **Rules**, per cui per ora ci limitiamo a una panoramica veloce.



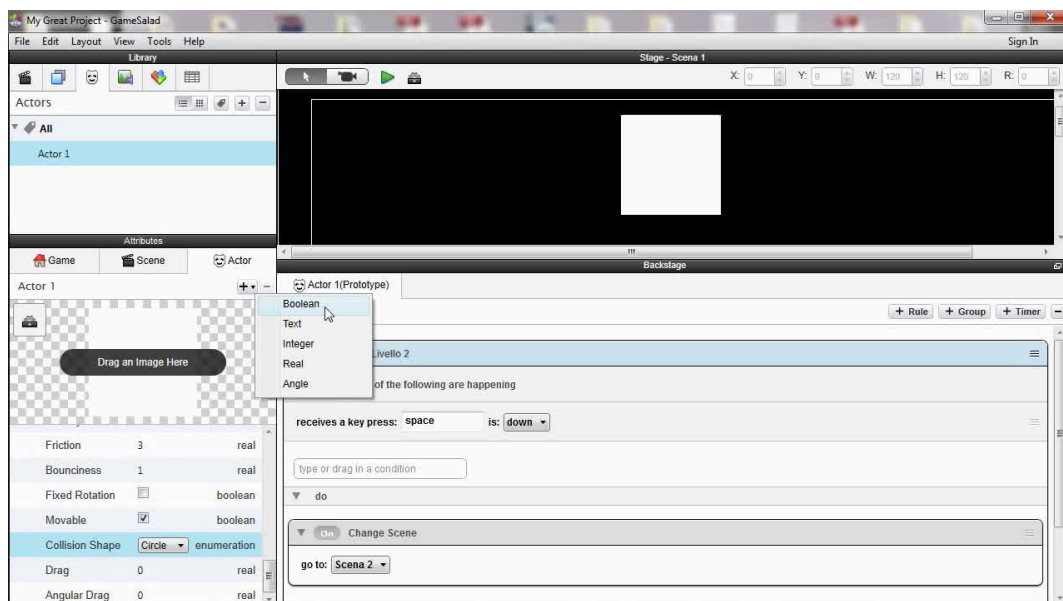
Sempre in **Attributes**, abbiamo tre raggruppamenti di parametri:

- **Graphics**, relativo all'aspetto visivo dell'elemento;
- **Motion**, contenente le informazioni su velocità lineare, velocità angolare e velocità massima;
- **Physics**, con le informazioni relative a peso, attrito, rimbalzo, forma delle collisioni, eccetera.

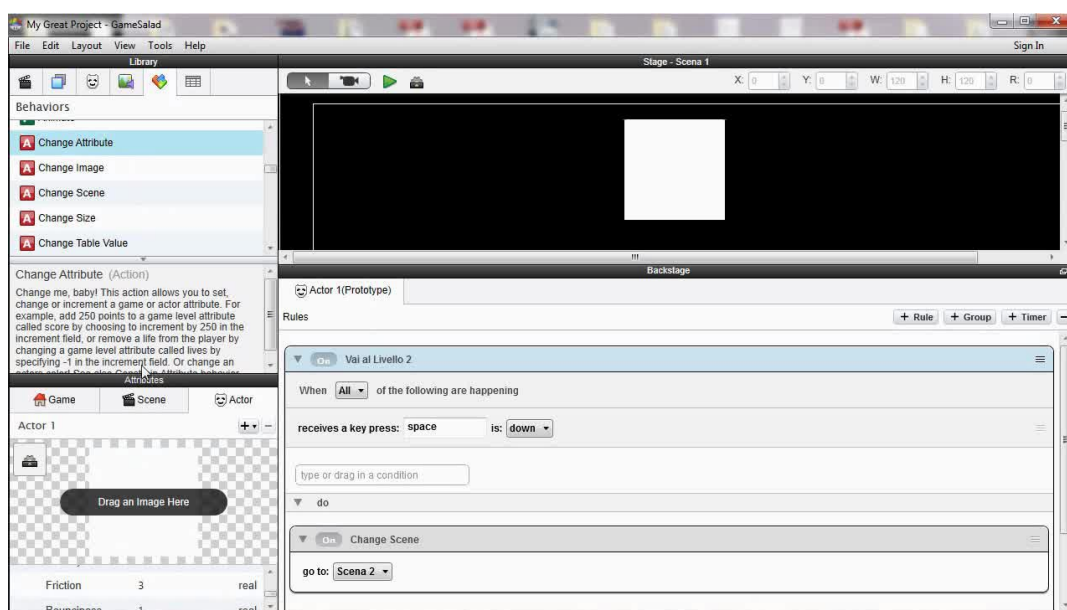


Ad ogni modo, questi **non sono gli unici attributi disponibili**: succede spesso, infatti, di dover definire “**flag**” o campi personalizzati, ad esempio il livello di energia del giocatore, il numero di proiettili a disposizione, il punteggio raggiunto, eccetera... tutti questi attributi sono, sostanzialmente, **variabili**, di tipo booleano (sì/no), stringhe, numeri interi, numeri con virgola o angoli.

È possibile **definire un attributo personalizzato** cliccando sul + nella scheda “**Attributes**” e selezionando, nel menù che apparirà, **il tipo di dato** che vogliamo.

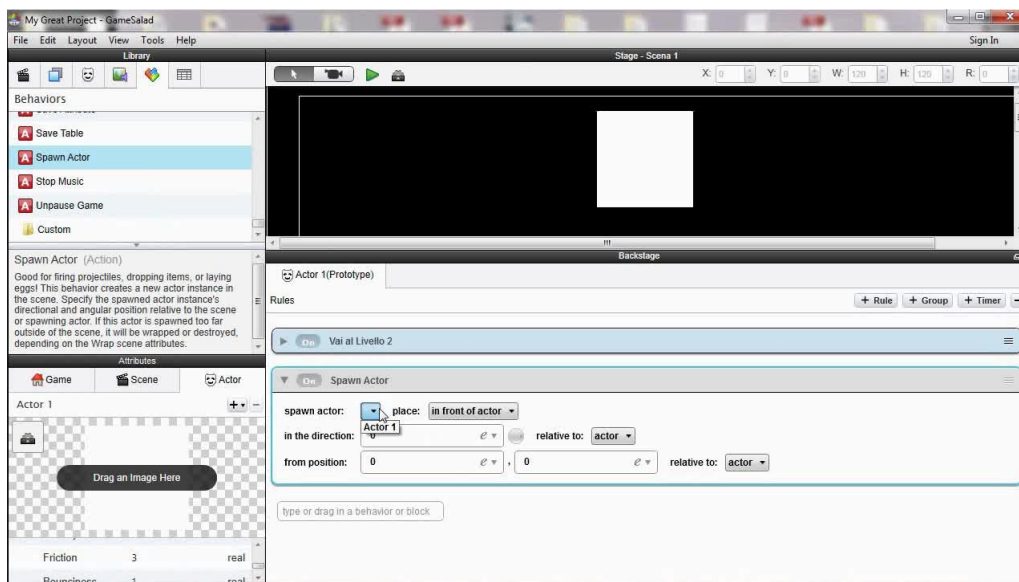


Chiaramente, questi attributi saranno accessibili dalle **Rules**, anche da altri oggetti di gioco, e modificabili mediante il **Behavior Change Attribute**.



Abbiamo detto che un attore può essere aggiunto alla scena mediante trascinamento, ma questa è una cosa che avviene, ovviamente, solo via *Editor*, quando si progetta il gioco; **per aggiungere un oggetto “al volo” (“a runtime”)**, in base ad un **evento** (definito, come sappiamo, con una **Rule**) si deve utilizzare invece il **Behavior “Spawn Actor”**, dal funzionamento abbastanza intuitivo: è necessario specificare quale *Actor* far apparire, dove (con le coordinate in riferimento o all'oggetto che lo genera o alla scena), con che direzione e velocità iniziale.

L'immagine seguente mostra l'inserimento di un **Behavior** “*Spawn Actor*” per un attore, all'interno del *backstage* di quest'ultimo; non è presente, ancora, la regola (**Rule**) che serve a definire l'evento in seguito al quale procedere con la creazione dell'attore.



Vedremo come **importare gli Assets** (file immagini e altri “media” in generale) e come collegarli agli *Actors* in un altro capitolo, comunque come esercizio potete esplorare la sezione **Behaviors** e creare delle **Rules** per intercettare eventi e generare semplici azioni.

* * *

Introduzione a Game Salad - Tutorial 5

I Behaviors

In questo quinto capitolo sulle basi di **Game Salad** ci occuperemo dei **Behaviors**.

A questo punto è chiaro che i *Behaviors* sono, in generale, **azioni** associabili agli *Actors* del gioco **per realizzare delle operazioni** in base a eventi o condizioni.

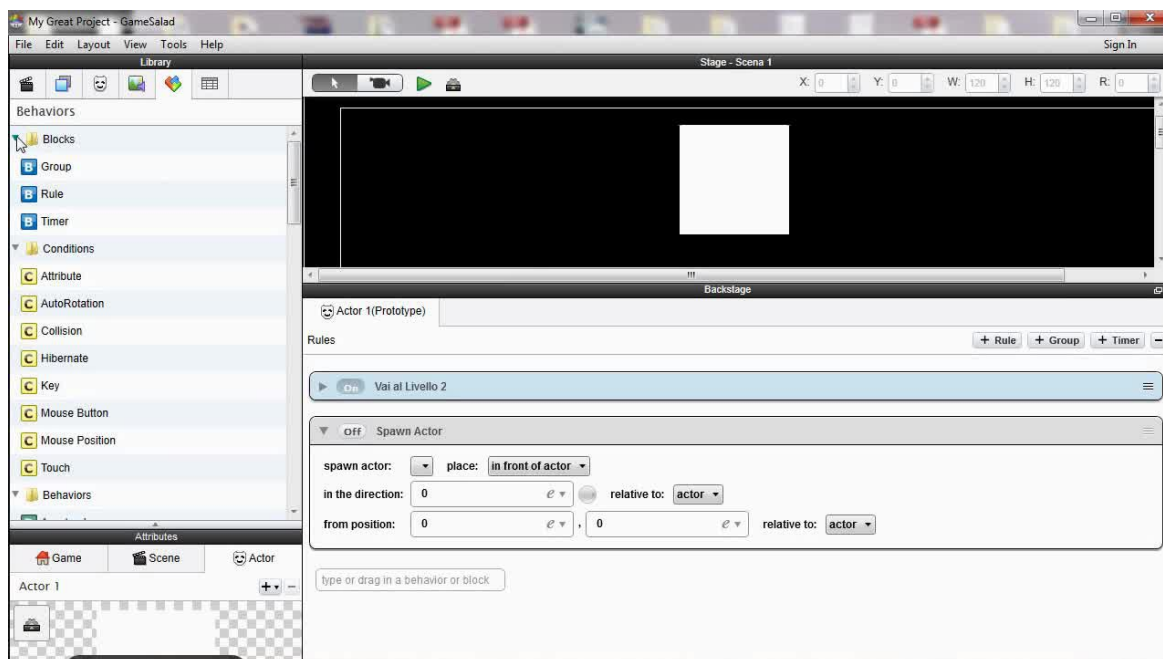
I **Behaviors**, nella versione 0.10 beta di **Game Salad**, sono 36.

In generale (ma non sempre), i **Behaviors** vanno inseriti all'interno di **Rules**, come abbiamo visto con degli esempi in precedenza, ed effettuano delle operazioni se si verificano certe condizioni (il “*When*” di una **Rule**).

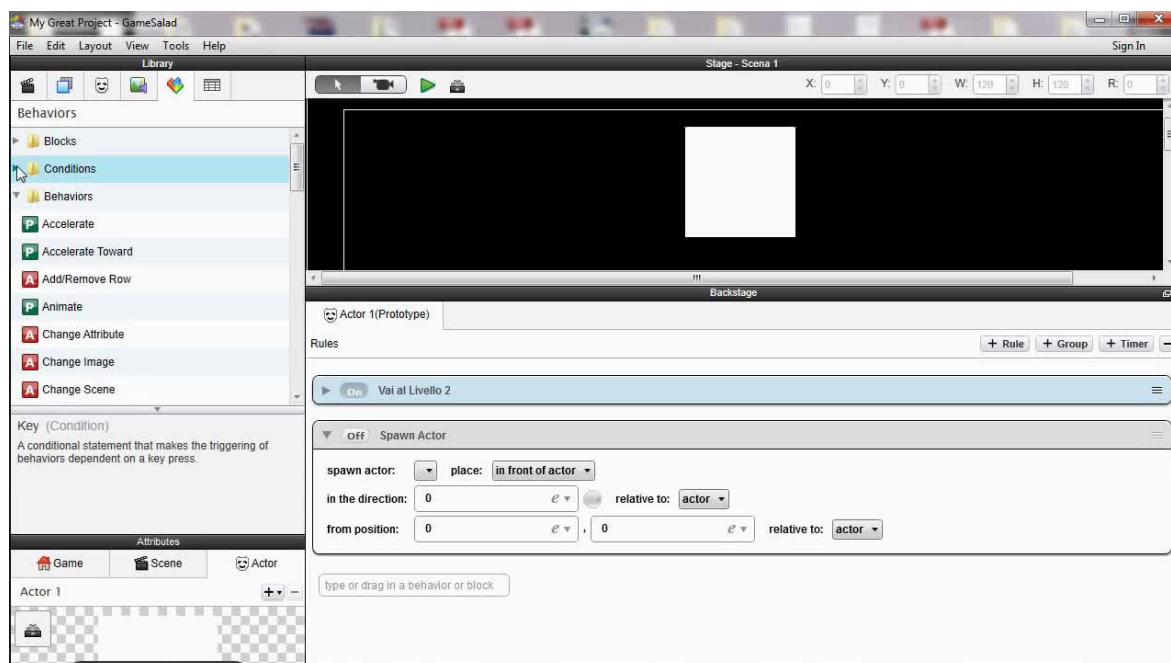
I **Behaviors** possono essere, sostanzialmente, di **tre tipi**:

- **condizioni**: indicati con la **C**, rilevano degli eventi e servono per attivare altri Behavior o modificare proprietà;
- **persistenti**: indicati con la **P**, vengono eseguiti in continuazione, frame dopo frame, a meno di non inserirli in delle **Rules** o di terminarli esplicitamente mediante altri **Behaviors**;
- **di azione** (*Action Behavior*): indicati con la lettera **A** accanto al nome: vengono eseguiti una volta e possono essere eseguiti più volte se posti in una **Rule** che passa da valida a non valida, poi di nuovo valida e così via.

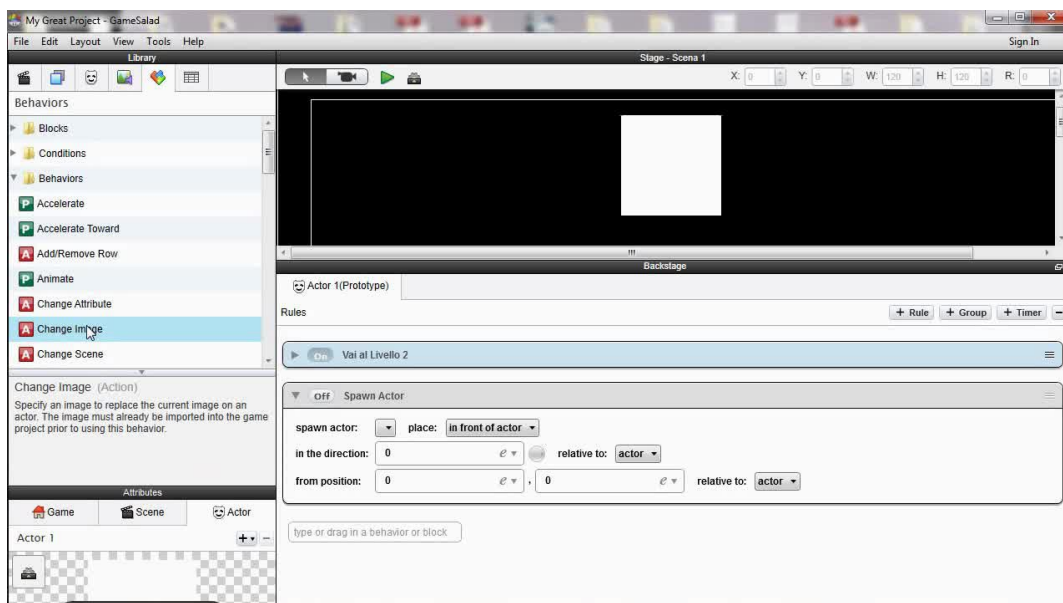
Esistono poi tre *Behavior* di tipo **B** (**blocco**, nel senso di “blocco di eventi / azioni”), che servono appunto a identificare dei **gruppi**, delle **regole** (le **Rules**) o degli **eventi temporali** (*Timer*; es.: ogni n secondi, dopo t secondi, ...), per gestire altri *Behaviors*.



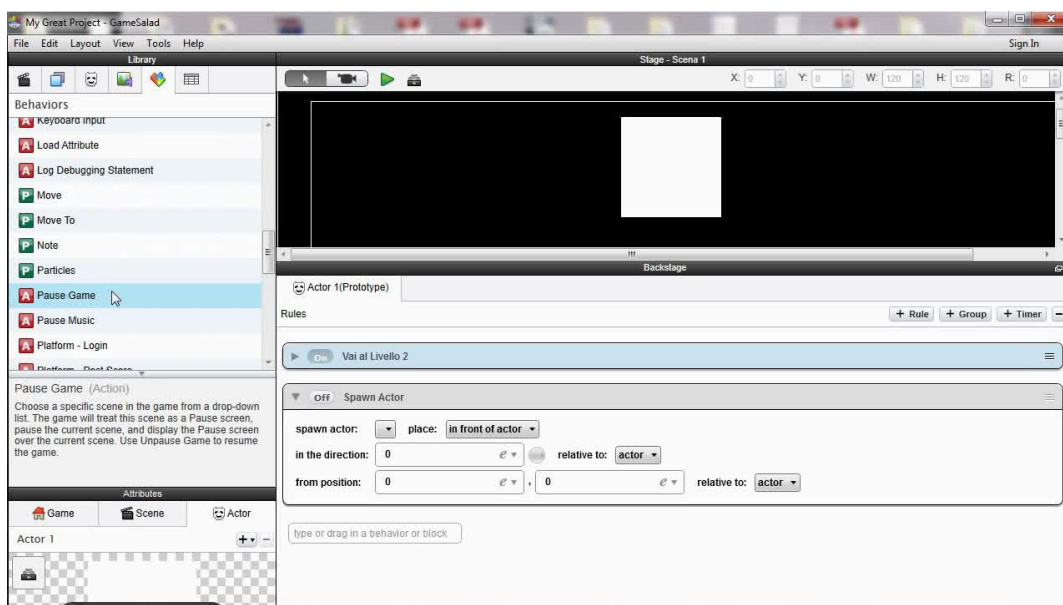
Nella sezione **Library – Behaviors**, cliccando su un *Behavior* apparirà, nella parte inferiore della scheda, una **breve descrizione** delle operazioni effettuabili con tale oggetto.



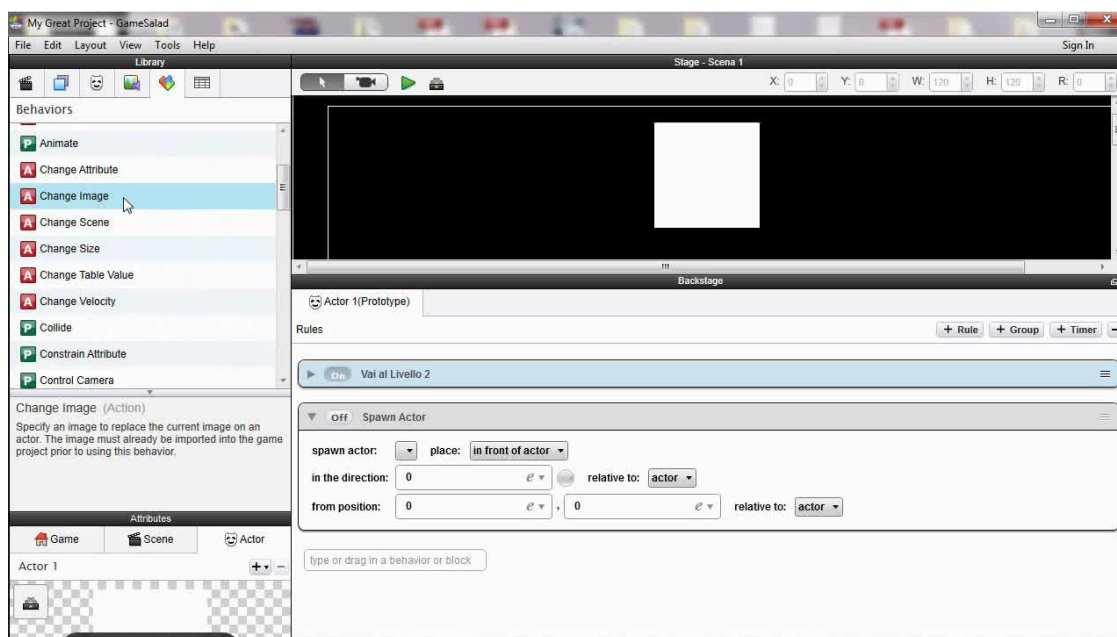
Abbiamo, ad esempio, **Behavior “Change”** che consentono di **modificare** il valore di un *Attribute*, anche personalizzato, oppure di cambiare scena, velocità o immagine di un *Actor*.



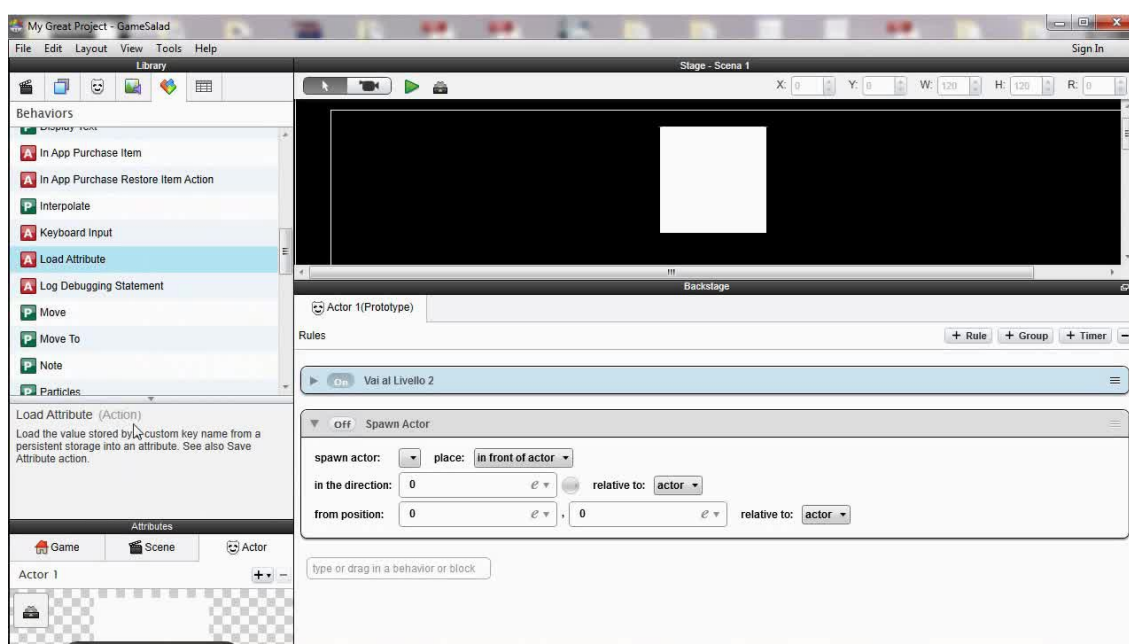
Riguardo la *Scena* e il *Progetto*, abbiamo *Behavior* in grado di **mettere in pausa un livello** o **passare ad un altro livello** (*Pause / Unpause Game, Change Scene*), **riavviare il livello** corrente o **l'intero gioco** (*Reset Scene, Reset Game*).



Buona parte dei *Behaviors* riguarda gli **Actors**: è possibile cambiare le immagini, riprodurre o fermare file audio ed effetti sonori, istanziare copie di oggetti (con *Spawn Actor*, specificando anche velocità e direzione iniziali), spostarli sia linearmente che verso destinazioni specifiche (con *Move* e *Move To*) anche in maniera accelerata (con *Accelerate* e *Accelerate To*), ruotarli e infine distruggerli (con *Destroy*).

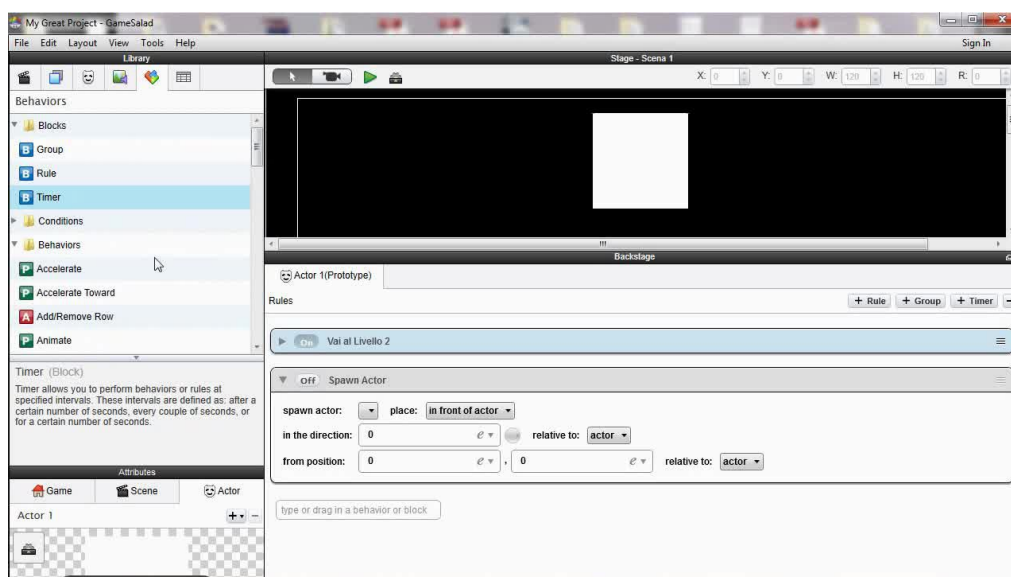
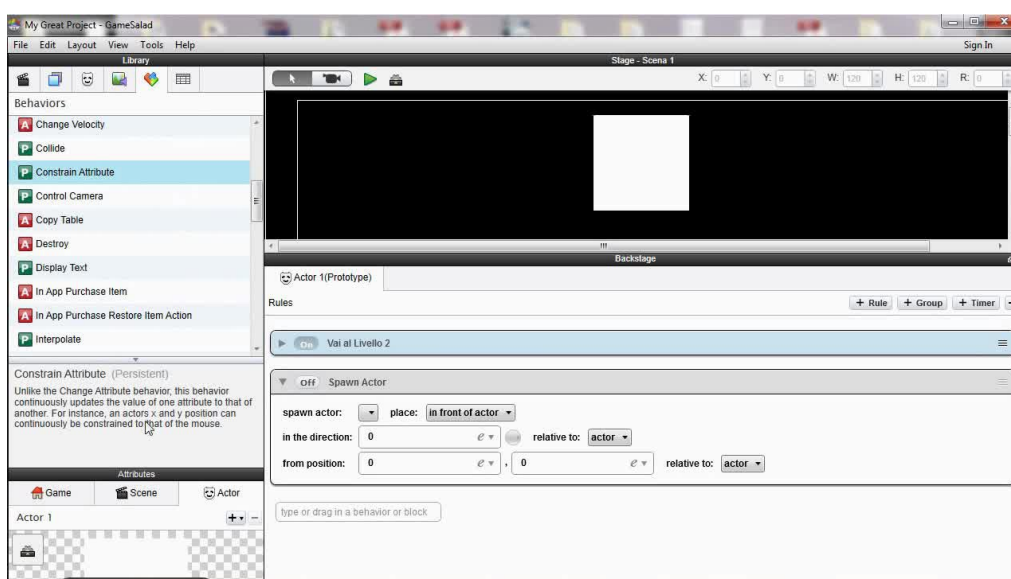


Riguardo gli attributi vanno menzionati anche *Save Attribute* e *Load Attribute* che consentono, rispettivamente, di salvare in maniera persistente, sul dispositivo, il valore di un *Attribute* di gioco e, viceversa, di caricare in un *Attribute* un valore recuperato dalla memoria permanente; si tratta, ovviamente, di operazioni utilissime **per memorizzare i dati su disco**, come i savegame, i nomi, le opzioni o la classifica dei punteggi.



Meritano poi una menzione a parte **Constraint** e **Timer**: il primo “costringe”, come suggerisce il nome, un parametro ad assumere i valori di un altro, il che è molto utile per realizzare oggetti con caratteristiche identiche o, ad esempio, movimenti e rotazioni in sincrono; il secondo è un *Timer*, un orologio, che consente di attivare degli eventi (altri *Behavior*) con 3 modalità:

- “**after**” lo avvia dopo n secondi dalla creazione del *Timer*;
- “**for**” mantiene attiva un'operazione per tutta la durata specificata in secondi;
- “**every**” attiva l'operazione ogni n secondi (per cui ad esempio l'autodistruzione di un oggetto avverrà “*after*”, dopo n secondi, mentre la generazione di alcuni nemici potrebbe avvenire “*for*”, per tutta la durata del gioco, oppure “*every*”, ogni tot secondi).



* * *

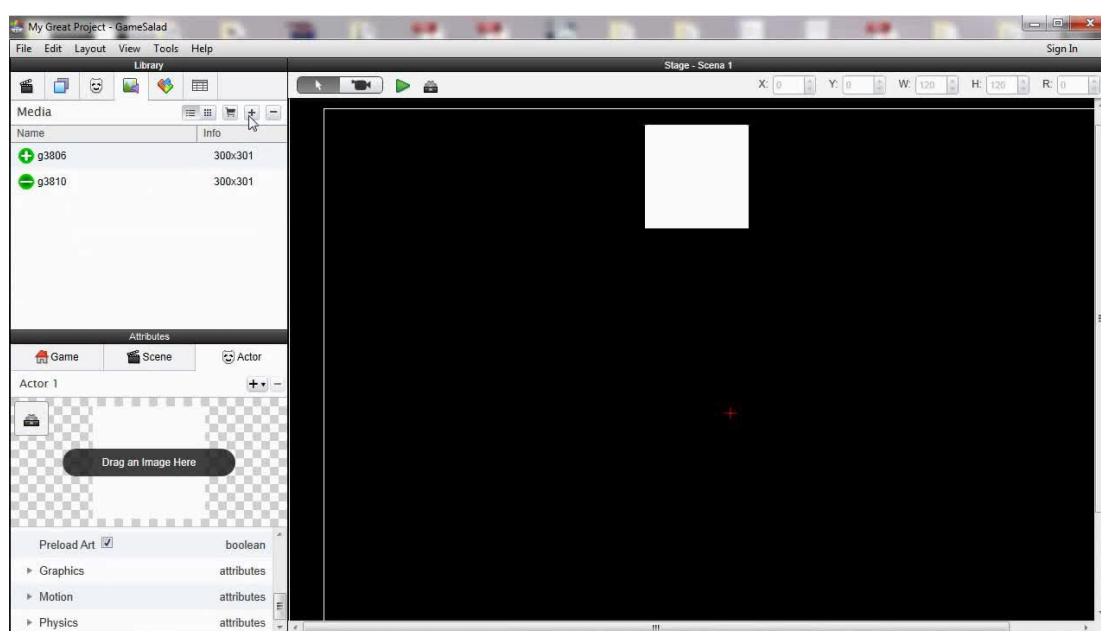
Introduzione a Game Salad - Tutorial 6

Assets: immagini, suoni e altri oggetti di gioco

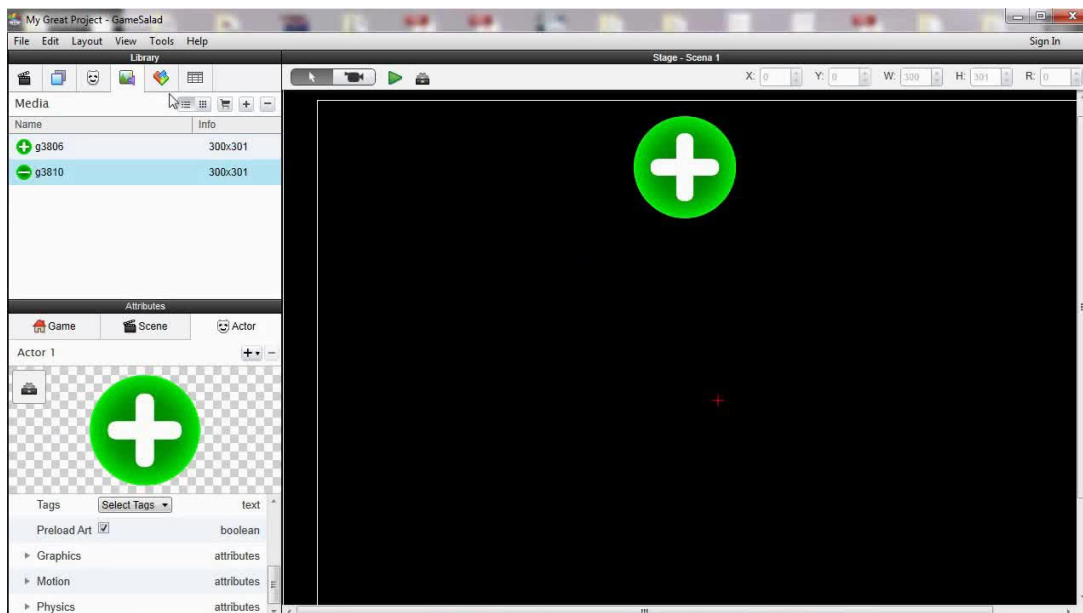
In questo sesto capitolo sulle basi di **Game Salad** parleremo degli *Assets* (immagini, suoni, animazioni e altri elementi di gioco) e della loro gestione all'interno di *Scene* e *Progetti*.

Abbiamo visto come definire e creare degli *Attori*, ma quelli trattati finora non avevano **immagini** da visualizzare, per cui ora vediamo come importare dei “*media file*” (elementi grafici, sonori, ecc...) nel nostro progetto e, quindi, come definire **l'aspetto visivo degli Actors**, anche in caso di **animazioni** (ad esempio, il ciclo di camminata di un personaggio).

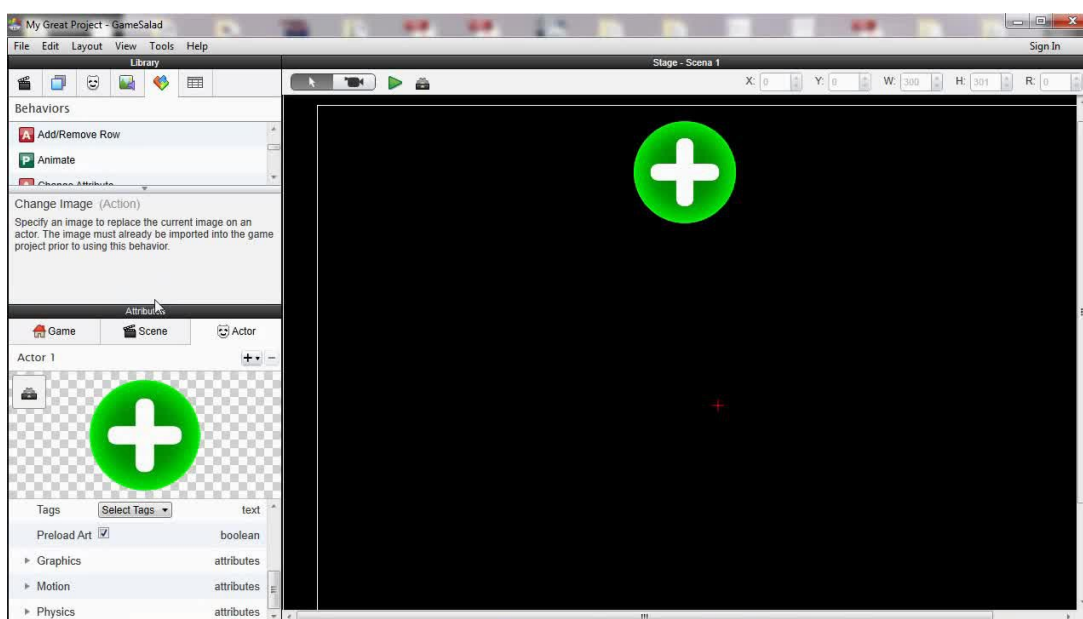
In realtà, l'unica operazione da fare è aprire la **scheda “Media”** della **Library** e... aggiungere un file immagine o sonoro, così come si aggiunge una *Scena* o il prototipo di un *Actor* al *Progetto*!



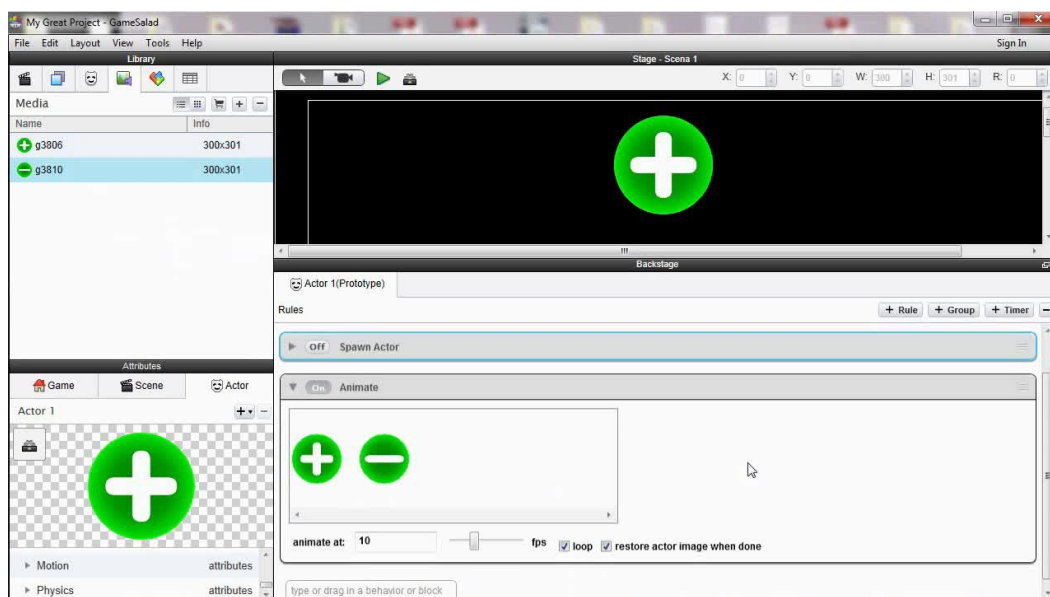
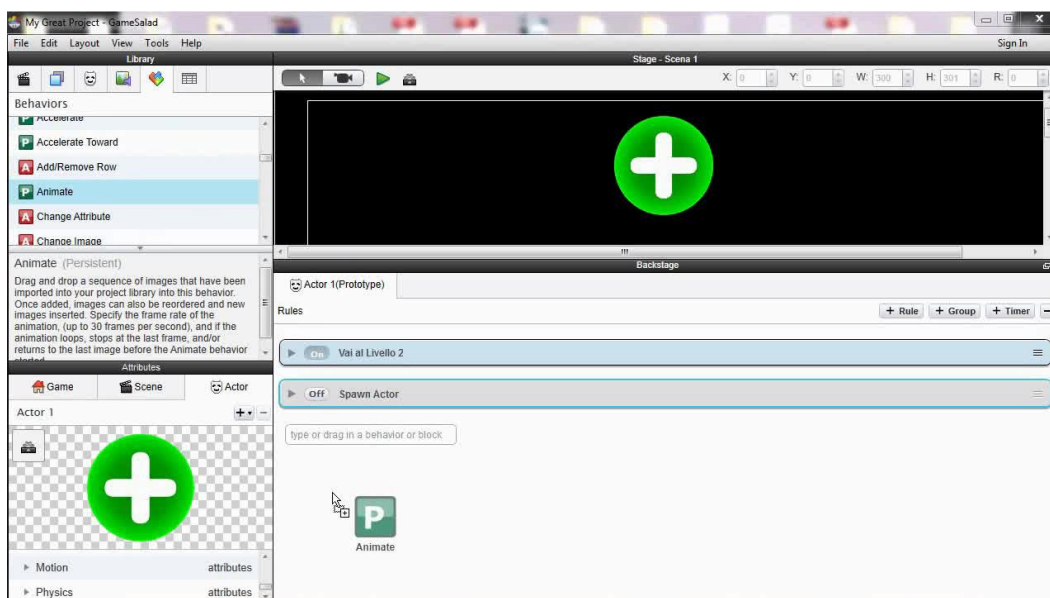
Una volta caricata, ad esempio, **un'immagine**, sarà possibile **associarla ad un Actor** in maniera statica (ovvero in fase di creazione del gioco, mediante *Editor*) selezionando l'*Actor* e **trascinando**, nel riquadro *immagine* nella scheda *Attributes*, l'immagine da utilizzare sullo spazio “*Drag an Image here*”.



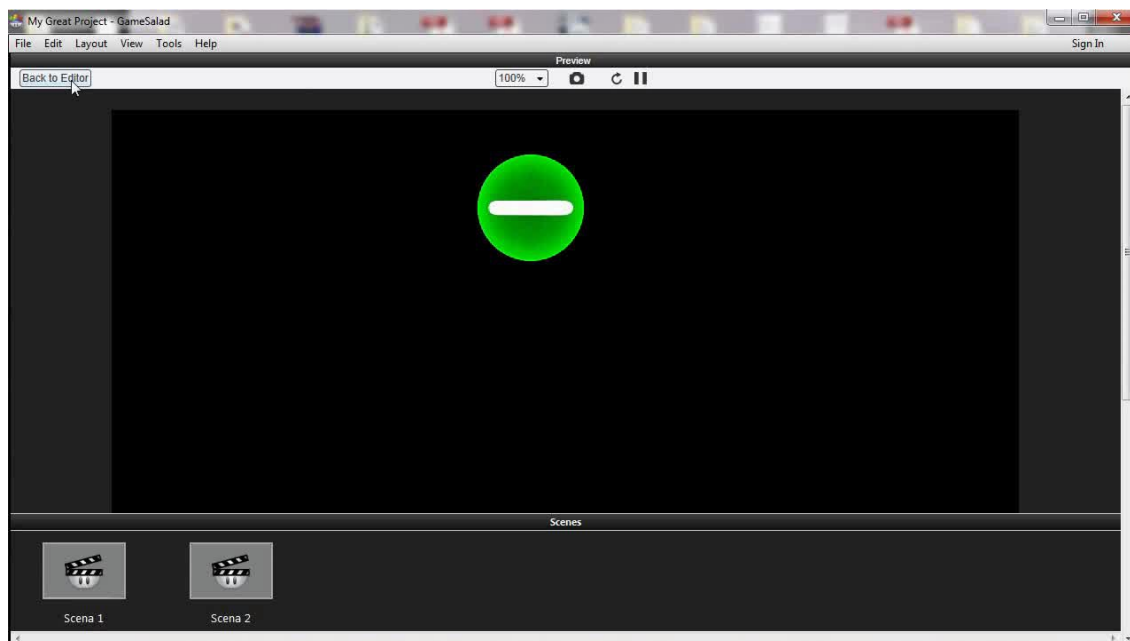
Come accennato velocemente nel tutorial precedente, esiste un **Behavior** che consente di **cambiare l'immagine** di un *Actor* anche a *runtime*, ovvero **durante l'esecuzione del gioco**, in base a degli eventi o condizioni: si tratta del **Behavior Change Image**, che consente di scegliere un'altra immagine tra quelle presenti nella **libreria Media** del progetto.



Per realizzare animazioni dell'aspetto visivo dei personaggi (ad esempio, un ciclo di camminata, una posa particolare, un salto, ecc) bisogna ricorrere al **Behavior** “*Animate*”, dove bisogna inserire delle immagini (le “*sprites*”, i fotogrammi dell'animazione da riprodurre), specificare il **frame rate** e la modalità (“*loop*” per eseguire un'animazione ciclica, altrimenti la sequenza verrà riprodotta solo una volta).



Ovviamente, il **Behavior** può essere inserito in delle **Rule**, per attivare l'animazione in base a degli **eventi**, oppure inserito direttamente nel *backstage* di un *Actor*, per riprodurre l'animazione in continuazione (a meno di non disattivare il *Behavior*, durante il gioco, con un altro *Behavior*).



Una nota **sull'ottimizzazione**: **iOS** allocherà, ovvero occuperà, una quantità di memoria per depositare le immagini basandosi sul valore larghezza per altezza più vicino ai multipli di due; detto in altre parole: se ad esempio un'immagine è grande 22*7 pixel, si ha che la coppia di multipli di due (2, 4, 8, 16, 32) più vicina, ovviamente per eccesso, è 32, per cui verranno allocati 32*32 pixel per memorizzare l'immagine.

Detto questo, **per ottimizzare l'esecuzione del progetto** evitate di passare alla scala successiva per colpa di un pixel (se ad esempio un'immagine ha dimensioni 129*100, magari tagliate la larghezza a 128, in modo da occupare lo spazio di un'immagine 128*128, perché lo slot successivo è quello di un'immagine 256*256, con un notevole spreco di memoria).

* * *

Introduzione a Game Salad - Tutorial 7

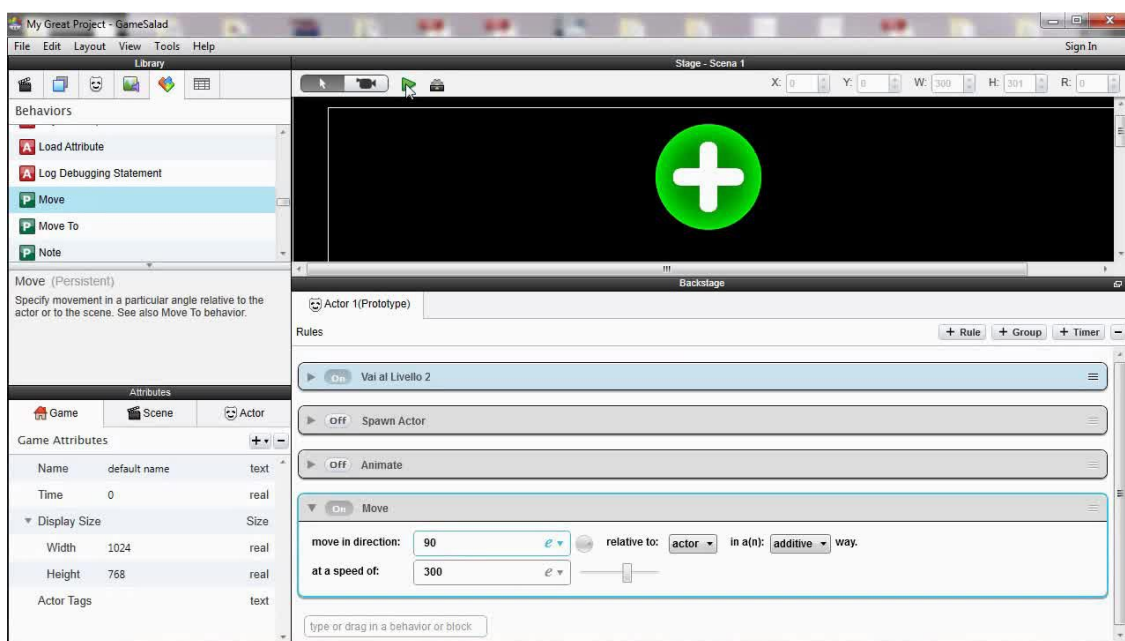
Muovere gli oggetti (Actors)

In questo settimo capitolo del corso introduttivo su **Game Salad** parleremo delle basi dei **movimenti degli oggetti**.

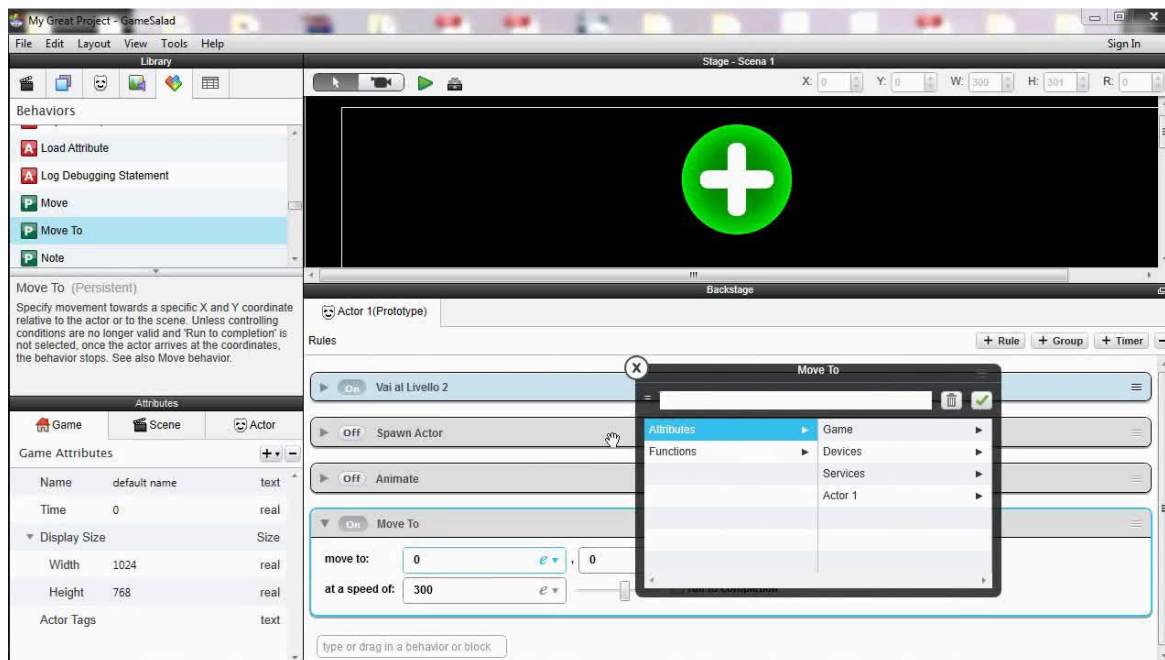
Come accennato nei tutorial precedenti, ci sono diversi **Behavior** relativi al **movimento** o alle **rotazioni** degli oggetti; in particolare, sono 6 e, vista la loro importanza, qui li descriveremo un po' più in dettaglio.

MOVE implementa uno spostamento semplice, lineare, non accelerato, per un oggetto.

È possibile specificare la direzione e se il movimento deve riferirsi alle coordinate di un *Actor* o a quelle globali della schermata. **MOVE** e **MOVE TO** sovrascrivono gli altri Behavior di movimento che stiamo per trattare, ignorandoli.

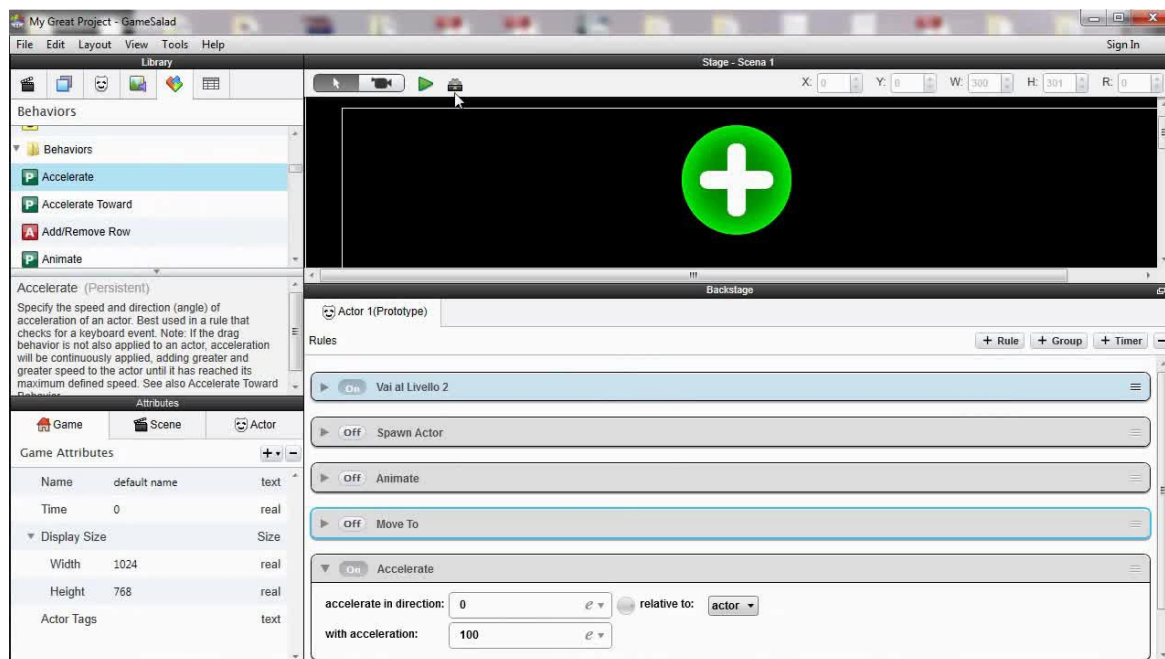


MOVE TO è simile a **MOVE** ma consente di specificare una coppia di coordinate xy verso le quali muovere l'oggetto; dal momento che è possibile reperire le coordinate xy di un altro oggetto (si tratta di *Attributes* degli oggetti), è possibile spostarsi letteralmente verso un altro oggetto di gioco.

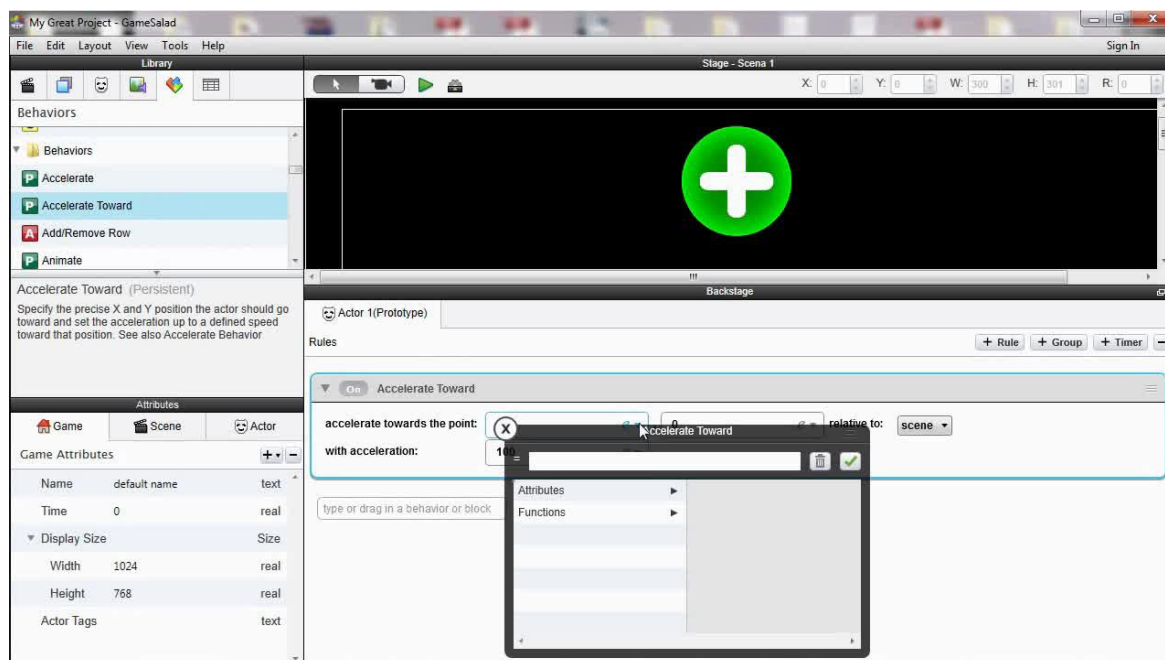


ACCELERATE fornirà un movimento accelerato ad un *Actor*, consentendo di specificare la direzione (utile, ad esempio, quando si spara un razzo) e la velocità massima raggiungibile.

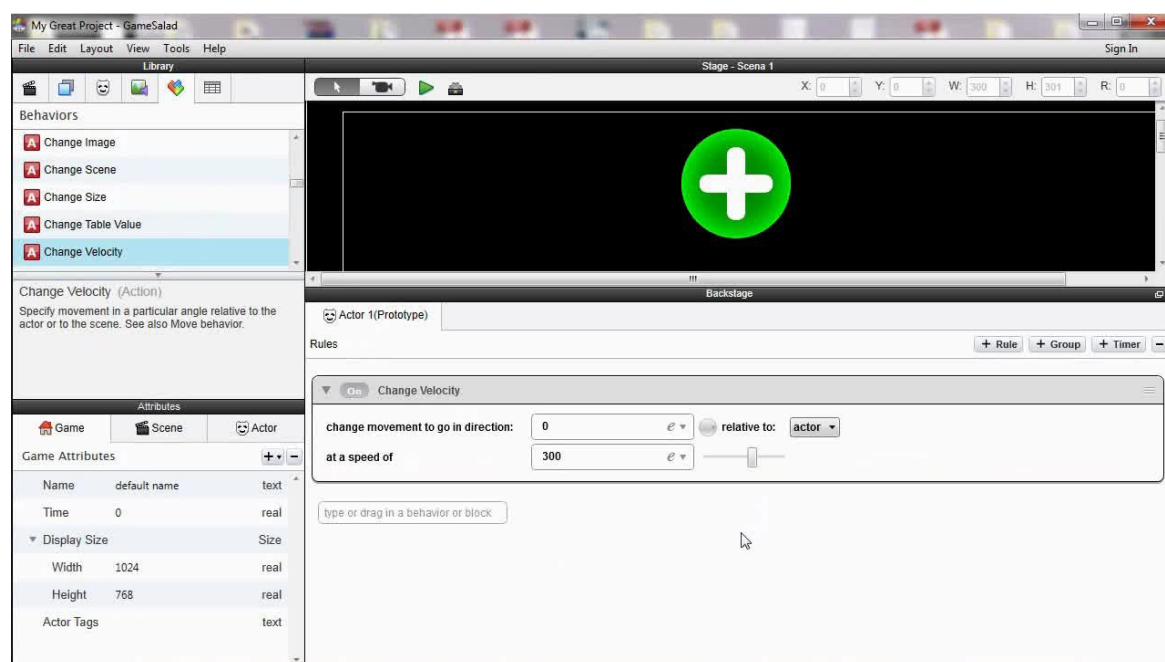
Il punto di origine può riferirsi o ad un *Actor* (ad esempio, in riferimento a chi spara) o in coordinate assolute della schermata.



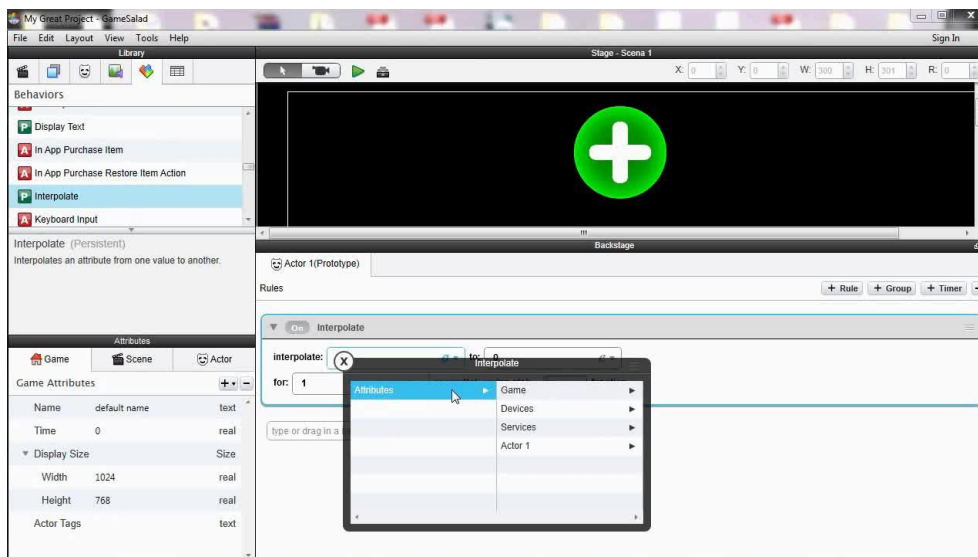
ACCELERATE TOWARD, cioè “accelera verso”, è per **ACCELERATE** quello che **MOVE TO** è per **MOVE**: stessa logica, ma con la possibilità di specificare delle coordinate xy (anche ricavandole dalla posizione di un altro *Actor*) verso le quali accelerare.



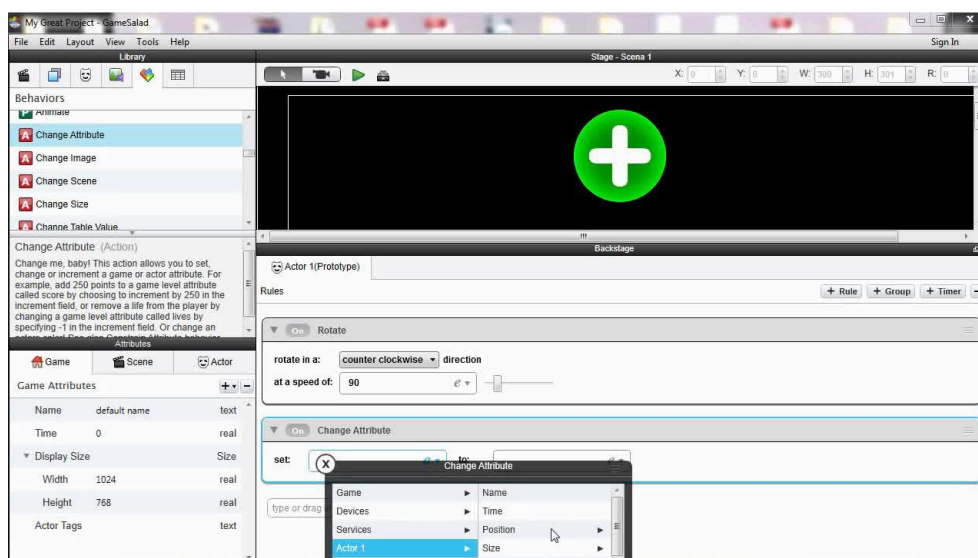
CHANGE VELOCITY consente, come suggerisce il nome, di cambiare la velocità attuale dell'oggetto, tuttavia non opera in maniera costante (come **MOVE** o **ACCELERATE**), ma solo quando viene attivato, per cui subito dopo rientrano in gioco le altre forze (la gravità o altri Behavior)... in pratica, è un “colpo secco”, non una proprietà o una forza che agisce nel tempo.



INTERPOLATE ATTRIBUTE effettua, come suggerisce il nome, un'interpolazione tra due valori (non per forza posizioni o orientamenti, ma attributi, ossia campi e valori in generale) in un determinato periodo di tempo, ovvero: preso un attributo-valore A , un valore B e una durata T , restituisce come valore quello che si ha passando da A a B in un tempo T , per cui all'istante 0 avremo il valore A , all'istante T il valore B e a metà avremo $(A+B)/2$.



A quanto detto vanno aggiunti i due *Behavior* **ROTATE** e **ROTATE TO**, che modificano l'orientamento dell'*Actor* (ossia lo ruotano); inoltre, volendo si può accennare anche a *Change Attribute* che, consentendo di modificare subito il valore di un attributo, può modificare istantaneamente (senza animazioni di spostamento) anche la *Position* X e Y di un oggetto (ad esempio, inserendolo in una *Rule* con *Timer* “for”, “every” o “after”).



* * *

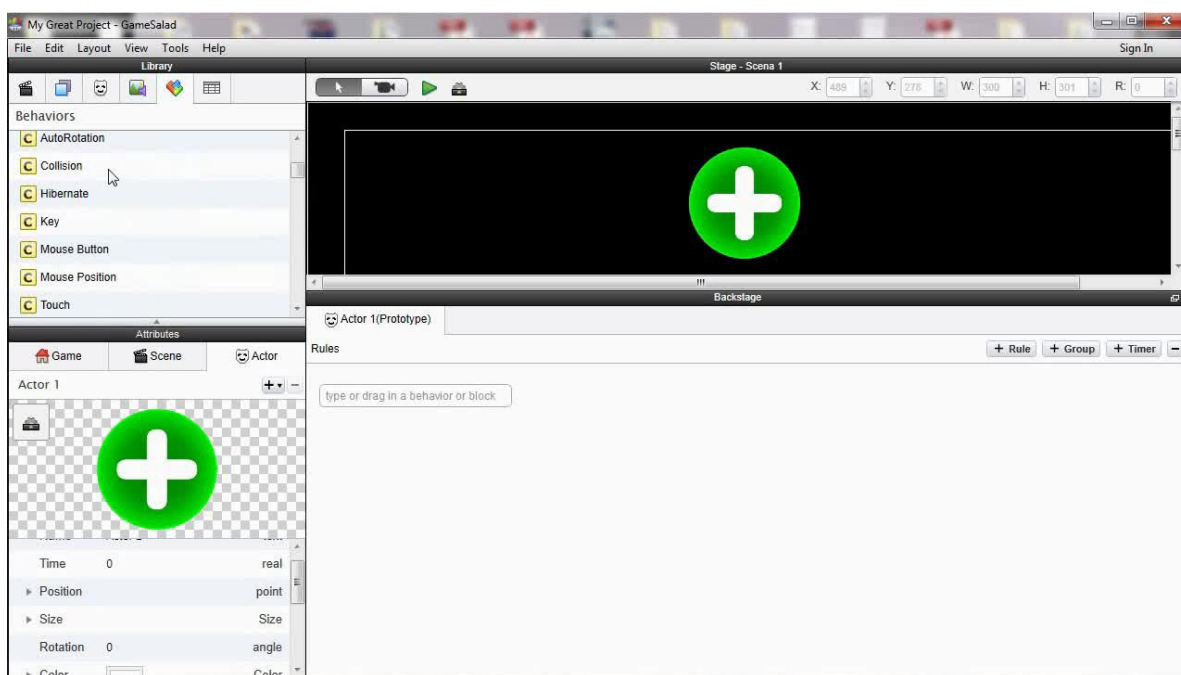
Introduzione a Game Salad - Tutorial 8

Rilevare le collisioni tra oggetti di gioco

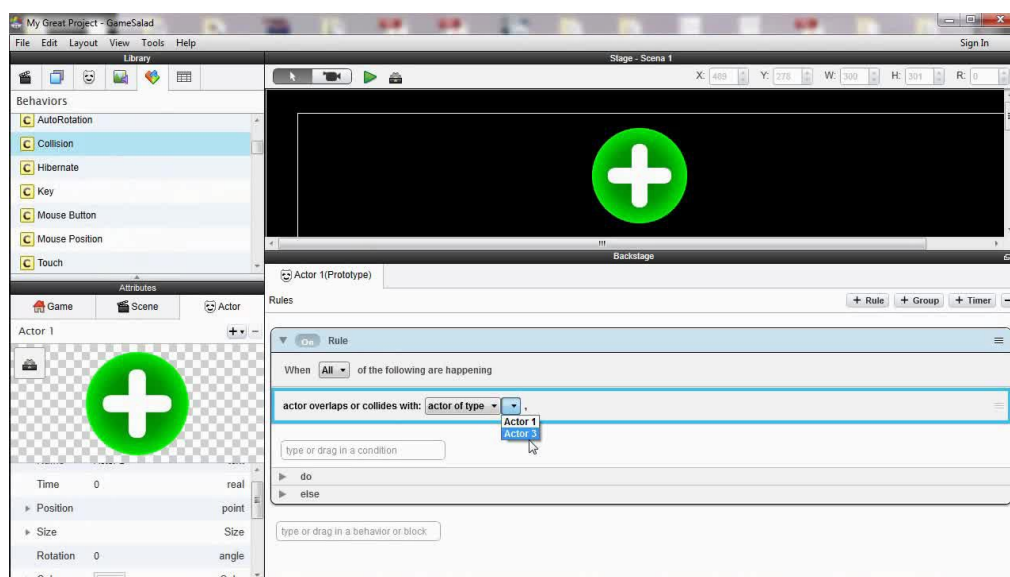
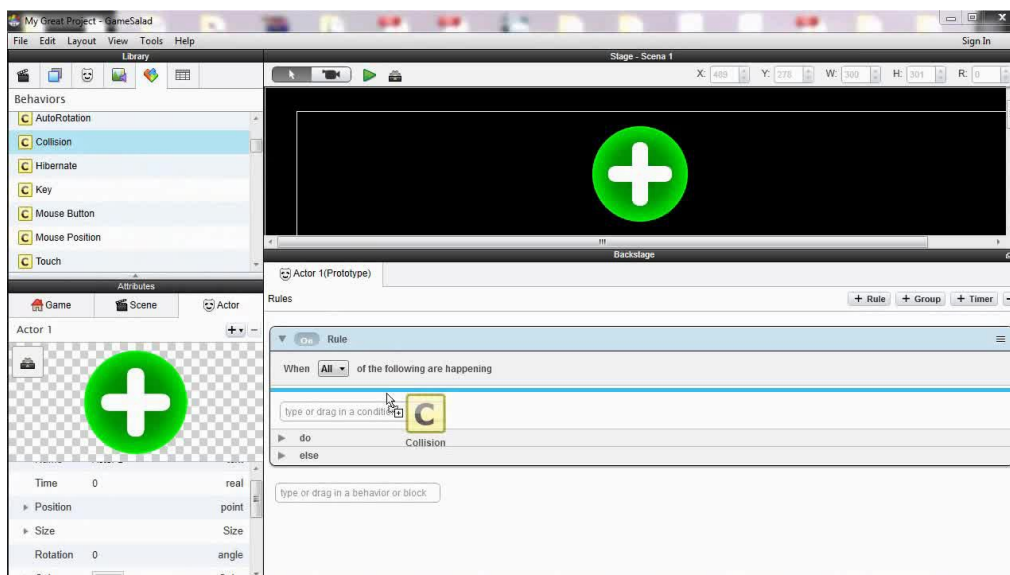
In questo ottavo capitolo parleremo, anche se in maniera puramente introduttiva, del **rilevamento delle collisioni** tra oggetti di gioco in **Game Salad**.

Per **collisioni** intendiamo la capacità di **rilevare il contatto tra due Actors** e la possibilità di **intraprendere delle operazioni** in seguito a tale contatto.

Non si tratta, quindi, di un **evento** prettamente dinamico (ad esempio, una pallina che rimbalza sul pavimento), ma anche “statico”, nel senso che anche un personaggio fermo è in collisione (col pavimento), per questo parliamo di *“rilevare il contatto”*: questo è l'evento, mentre quelle che seguono sono solo **reazioni**, azioni da intraprendere in base all'evento.

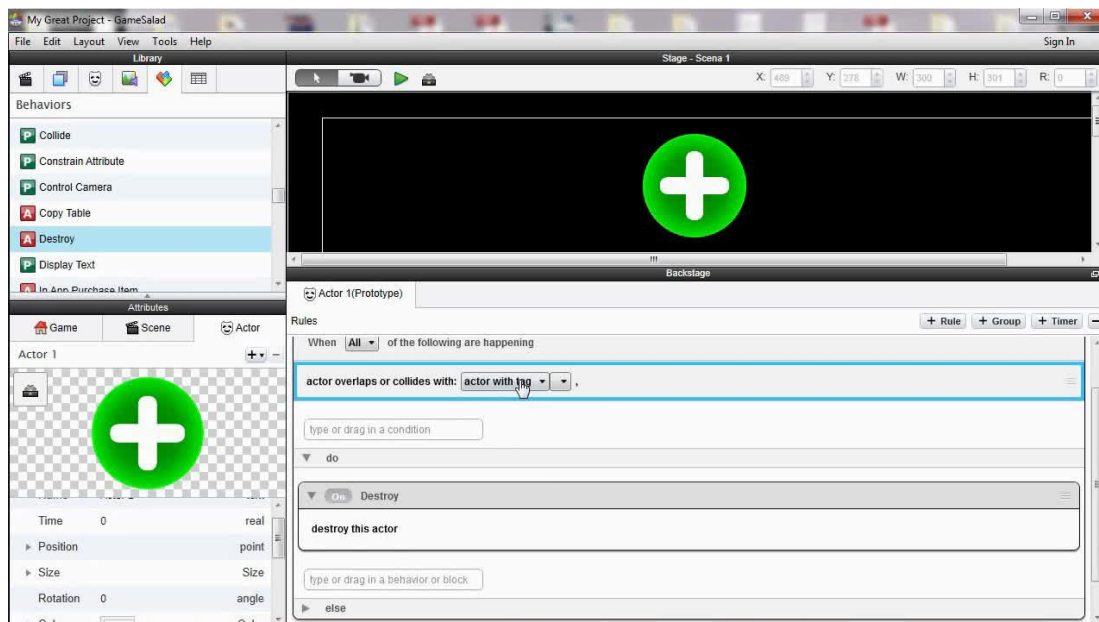


Le collisioni vengono rilevate fornendo un **Behavior Collide** ad un *Actor*; in particolare, se l'oggetto *A* ha un *Behavior Collide* e tocca l'oggetto *B*, verrà segnalata una **collisione** anche se *B* non possiede un *Collide* (per cui non è necessario inserire un tale *Behavior* per pavimenti o altri elementi statici).



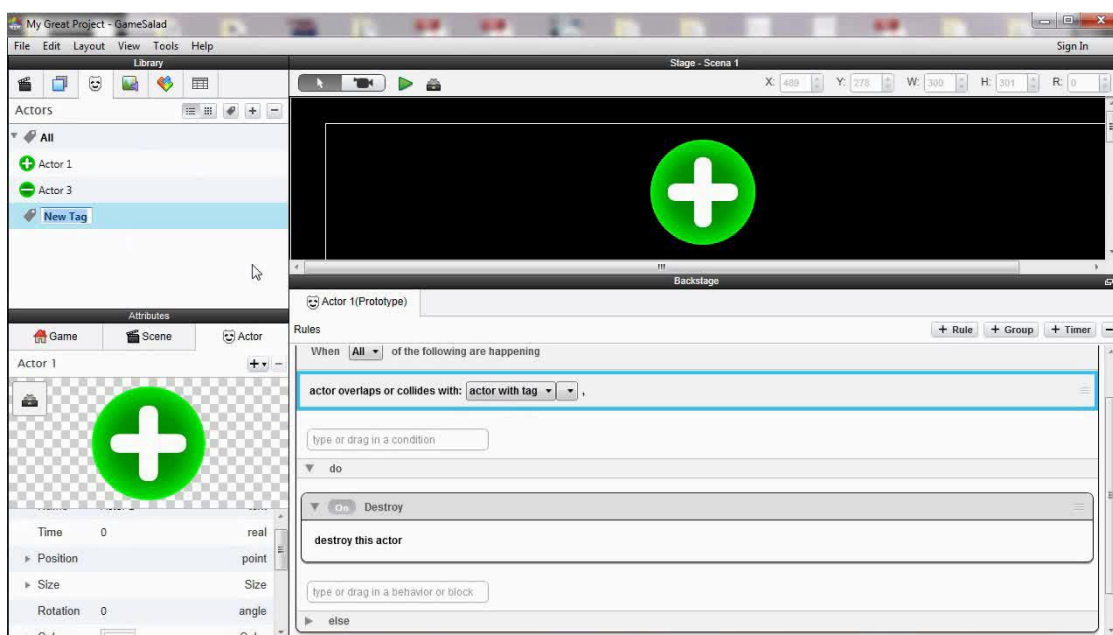
È possibile **limitare il rilevamento di una collisione**, per l'oggetto che possiede *Collide*, a **certi Actors** di scena, specificandoli mediante raggruppamenti o l'utilizzo dei “**Tag**”, ovvero “etichette”; questi sono sostanzialmente degli **attributi** (e infatti possono essere selezionati dal menù associato alla voce **Tag** nella scheda **Attribute** di un *Actor*) che però vanno creati nella scheda **Library – Actors**, cliccando sull'icona dell'etichetta (e in effetti “**Tag**” significa “**Etichetta**”, infatti “etichettiamo” degli *Actor* che appartengono a dei gruppi).

Nell'immagine seguente: come specificare, per un evento di collisione, di limitare il rilevamento della stessa agli *attori* che possiedono una determinata *etichetta (tag)*.

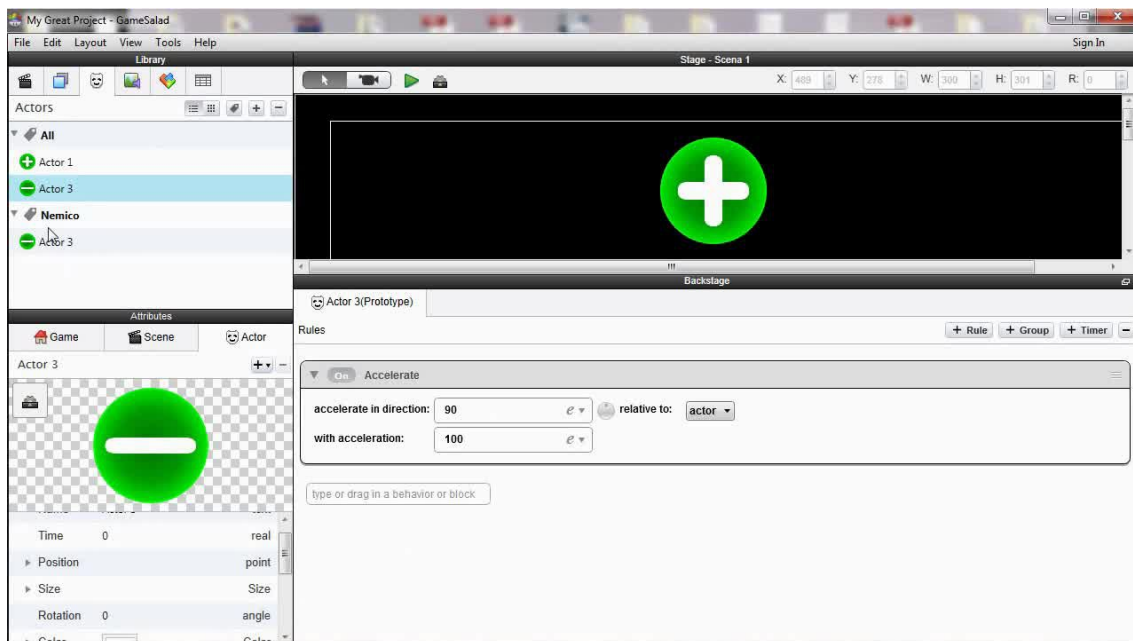


Nell'immagine seguente: creazione di *un'etichetta (tag)* per un attore.

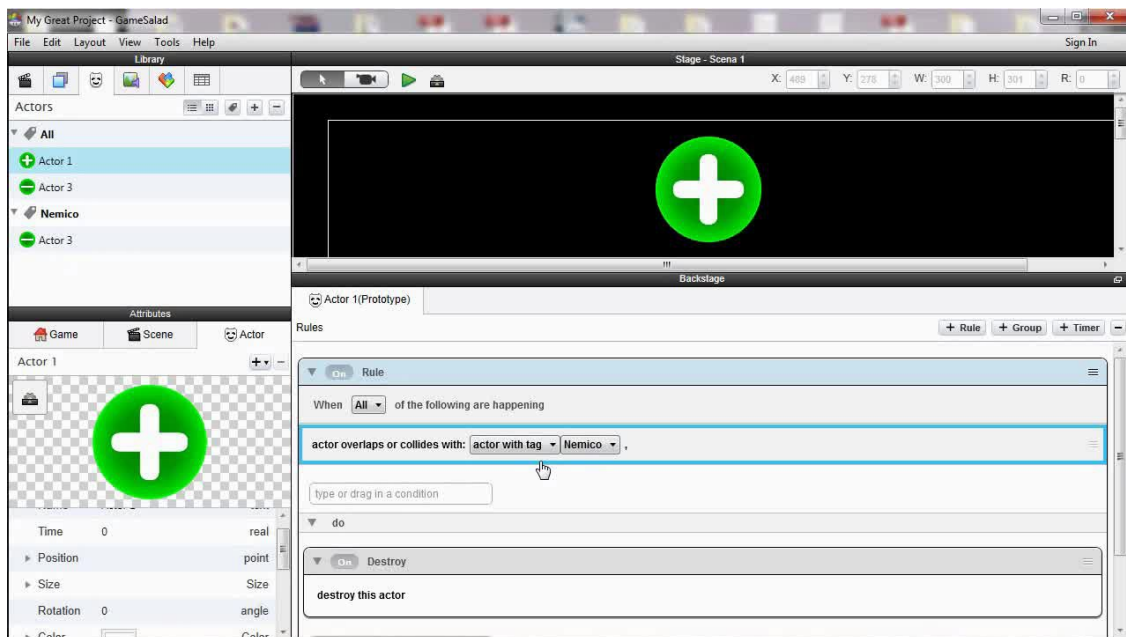
Le etichette sono, sostanzialmente, **attributi** degli oggetti.



Nell'immagine seguente: l'attore *Actor 3* è presente, nella *Library*, sia sotto l'etichetta “*All*” (tutti) che sotto l'etichetta “*Nemico*” (il *tag* che abbiamo creato e associato a questo attore).



Un *Behavior* può essere trascinato direttamente nel *backstage* di un *Actor* (e in questo caso sarà sempre attivo, ossia rileverà sempre delle collisioni con altri oggetti) o utilizzato in *Rules*; un esempio pratico di **utilizzo di *Collide* in una *Rule*** è la distruzione in seguito a un impatto, applicando il *Behavior Destroy* se un oggetto ha una collisione con un altro.



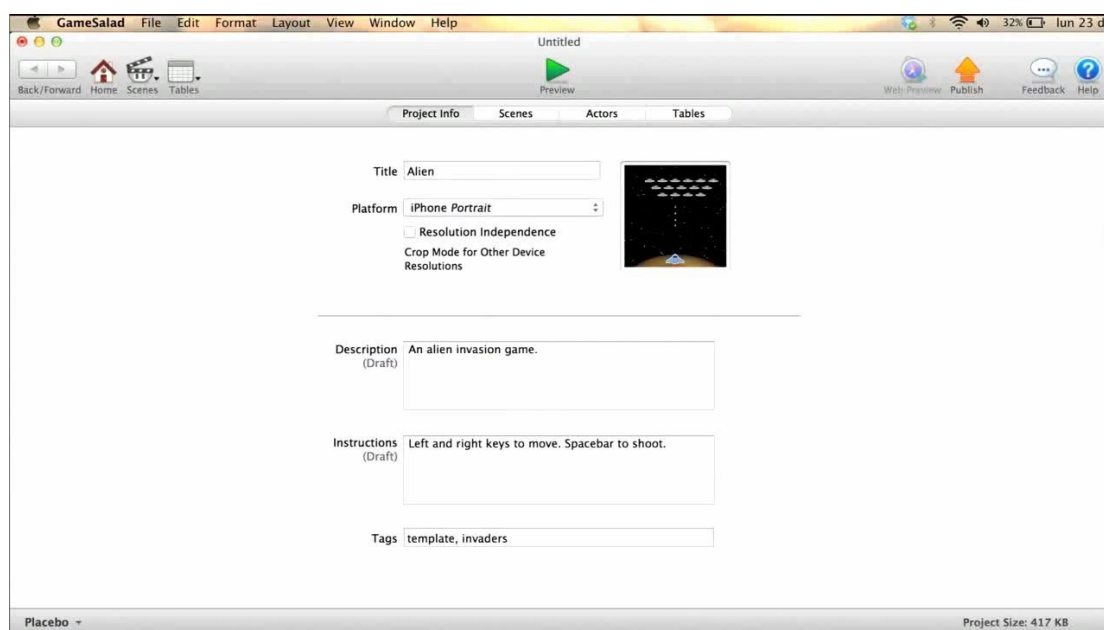
* * *

Introduzione a Game Salad - Tutorial 9

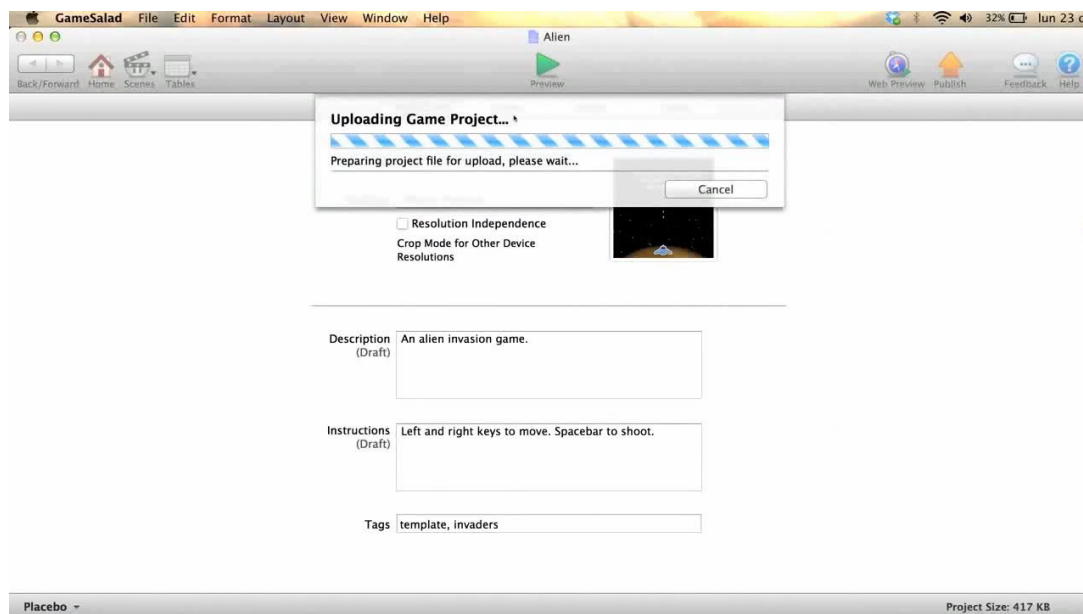
Compilazione e pubblicazione di un Progetto

In questo nono capitolo sulle basi di *Game Salad* parleremo della “compilazione” e della **pubblicazione** di un *Progetto* (ovvero, di un gioco).

Una volta ultimato il progetto, infatti, dobbiamo procedere con la pubblicazione dello stesso, per renderlo disponibile su dispositivi *Mac*, *iOS*, *Android* o per *HTML5*.

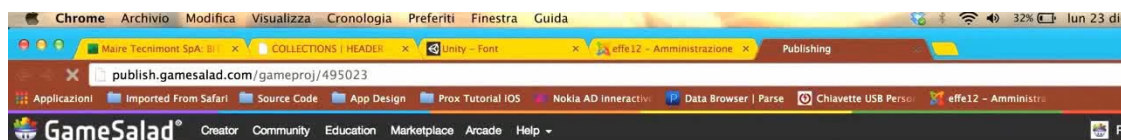


Il primo passo da fare è quello di **effettuare l’upload del Project su Gamesalad**: clicchiamo su “**Publish**” ed aspettiamo che finisca l’upload.

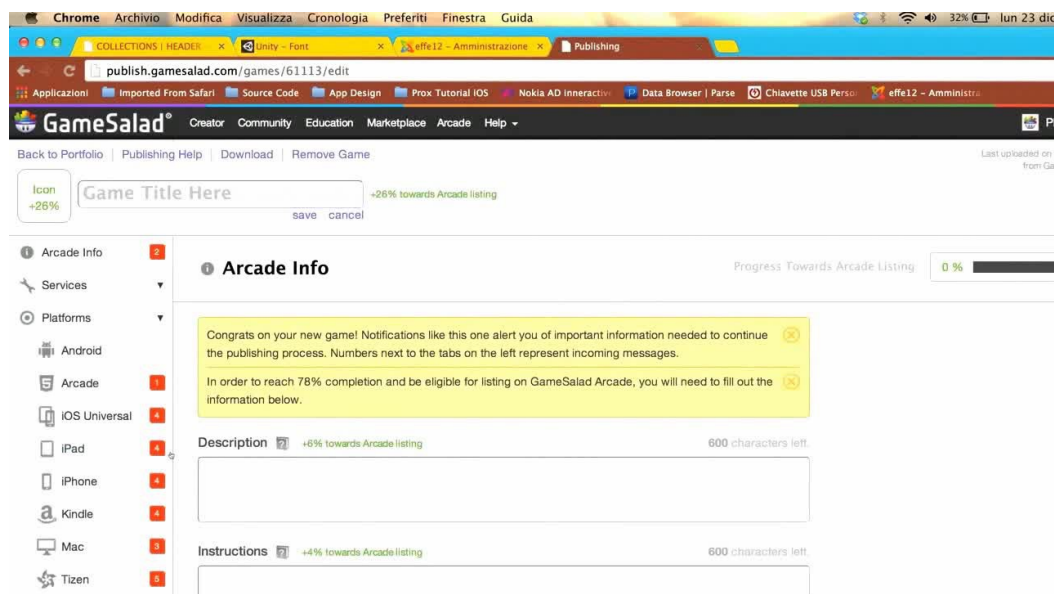


Ad upload ultimato, si aprirà in automatico la **finestra del browser** con la pagina di “**Publish Gamesalad**” all’interno del nostro profilo **GS** e ci verrà chiesto se il pacchetto inviato è una **nuova app** o un **aggiornamento** di una esistente già presente nel nostro *portfolio*.

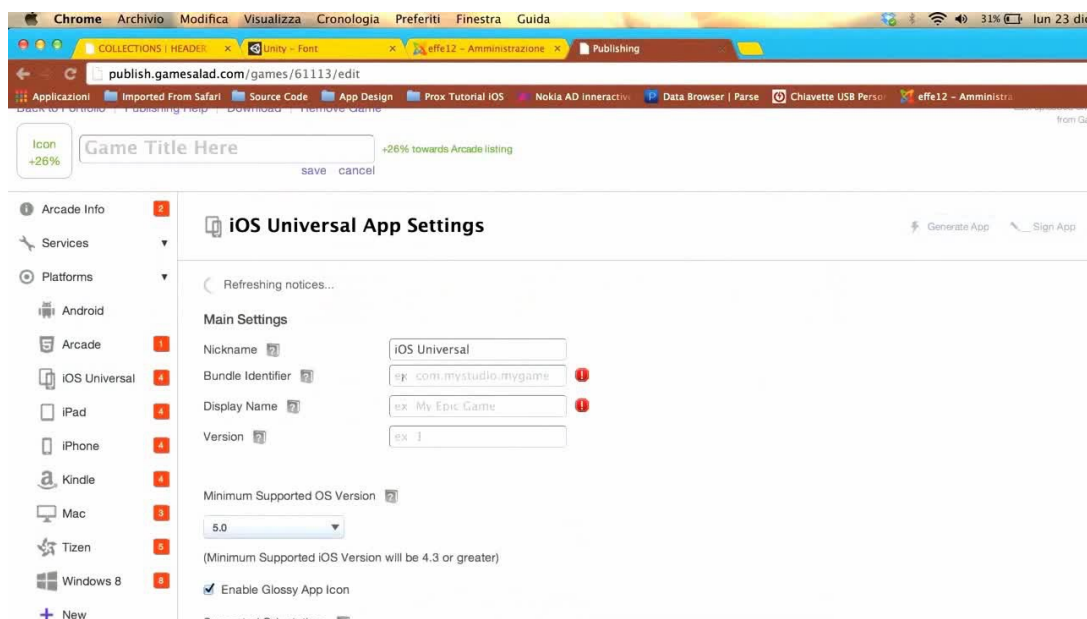
Nel nostro caso, scegliamo “**New App**”.



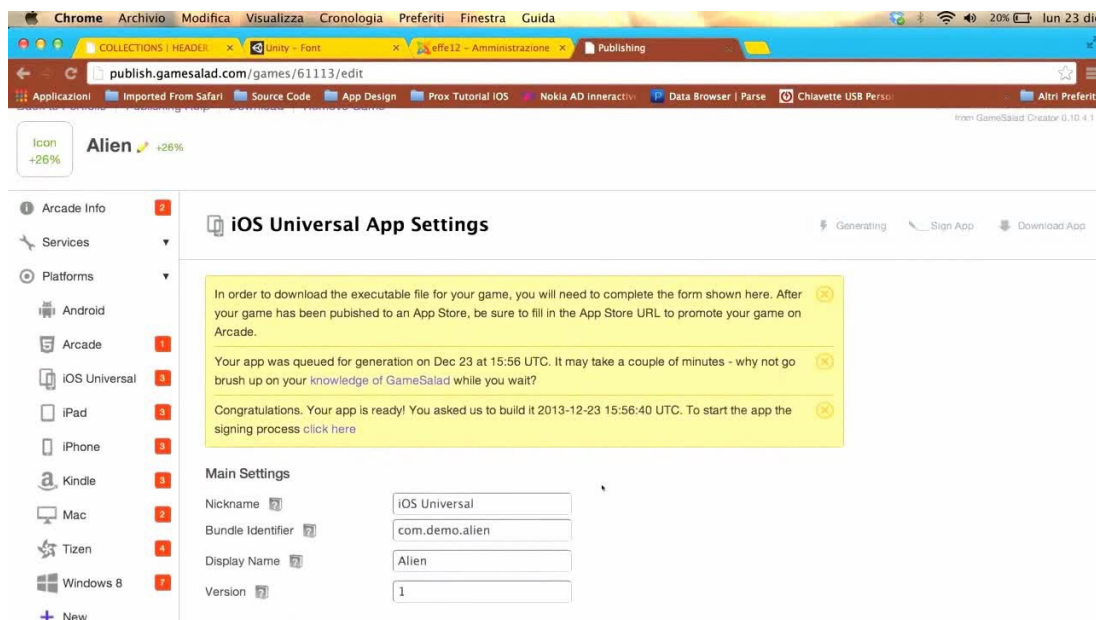
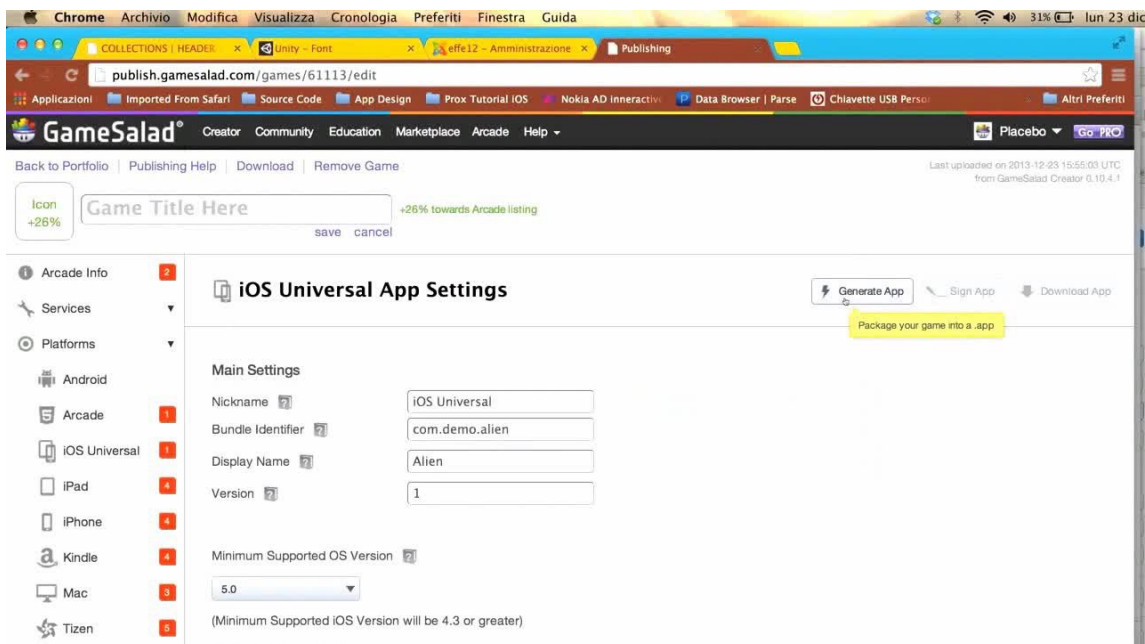
Riempiamo i campi delle informazioni necessarie e inseriamo gli **screenshot** che ci vengono richiesti (per mostrare schermate o parti della nostra *App*, in quella che sarà la pagina web descrittiva della stessa), quindi scegliamo dal **menù “Platforms”** di sinistra la piattaforma su cui vogliamo pubblicare l’app realizzata con **GS**; in questo caso, sto scegliendo **“iOS Universal”**.



Anche qui, inseriamo le **informazioni** che vengono richieste (nome app, versione, bundle id, screenshot, ecc..).

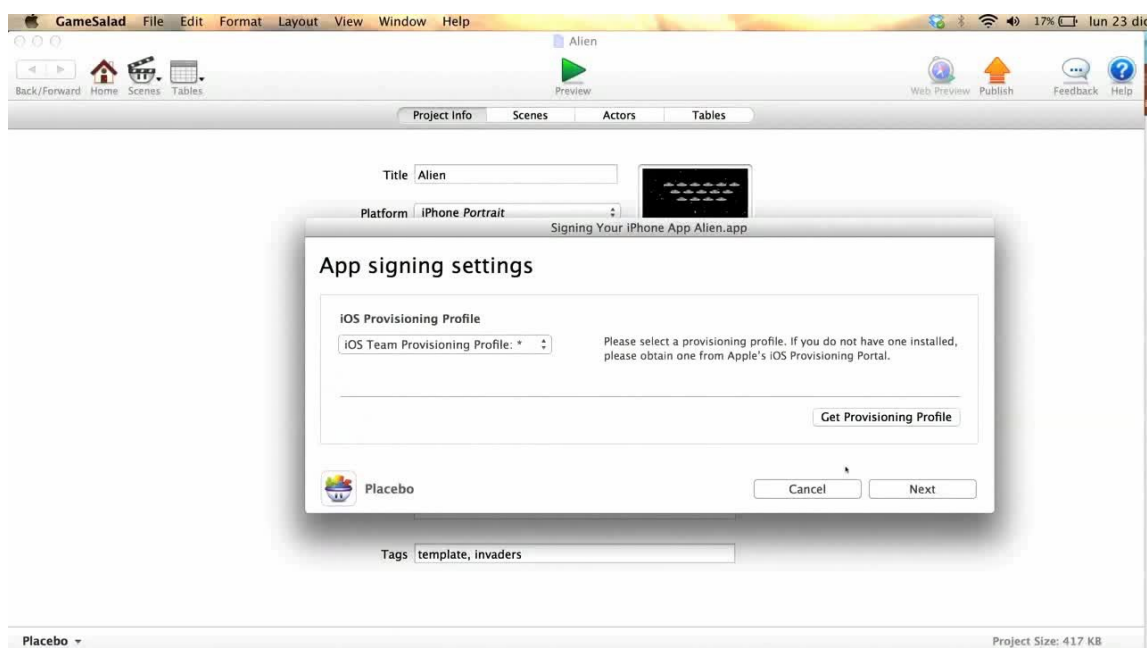
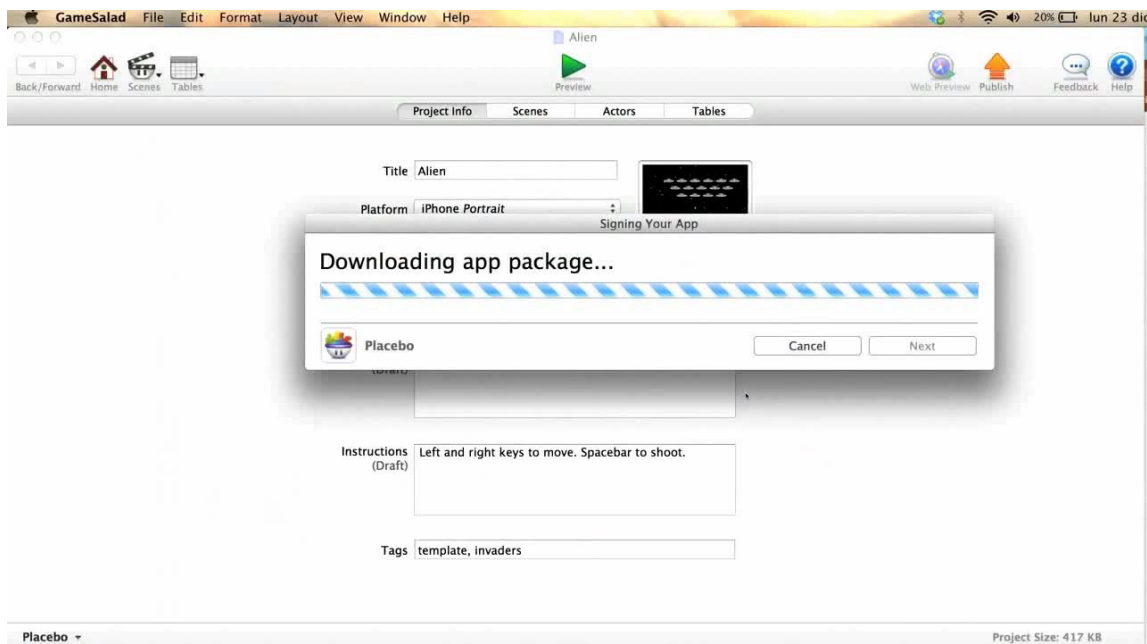


Finiti gli inserimenti clicchiamo su **“Generate App”** in alto a destra ed attendiamo che **GS** elabori il **pacchetto** con le informazioni che vi abbiamo inserito.

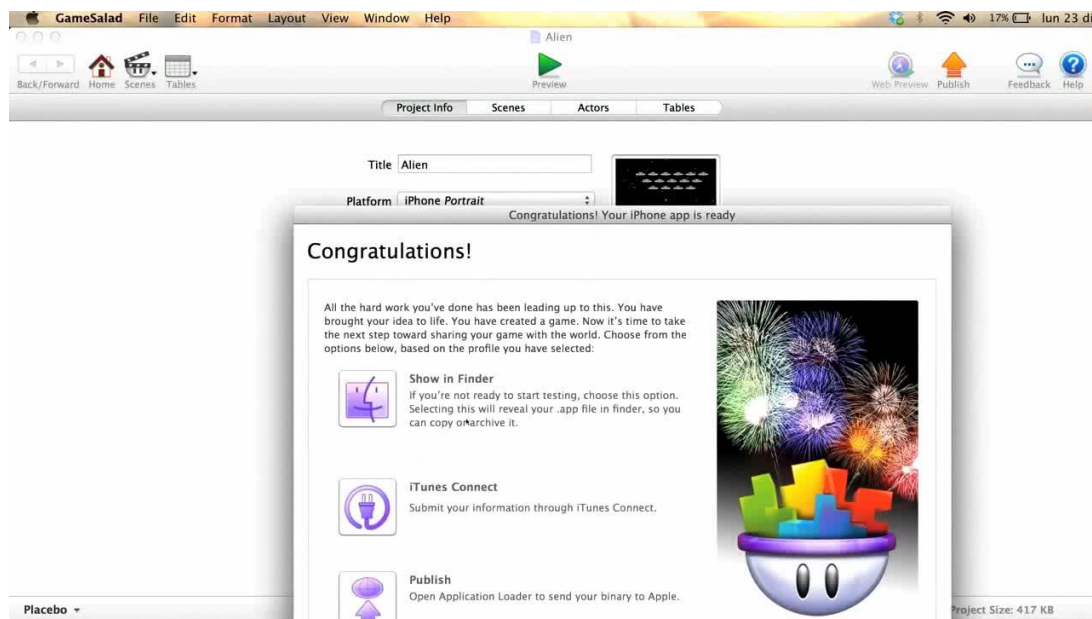


Quando leggiamo il messaggio *“Congratulations. Your app is ready! You asked us to build it (data). To start the app signing process click here!”*, clicchiamo nel link presente, autorizziamo il browser ad aprire il programma **Gamesalad** ed attendiamo che finisca di **scaricare il file .App**; in

questo caso, essendo una pubblicazione **Apple**, ci verrà richiesto di **selezionare un certificato** per firmare l'app.



Completato anche questo passaggio, **salviamo sul nostro computer il file**, che sarà pronto per essere **inviato agli store di pubblicazione** per i vari dispositivi (in questo caso, allo **store della Apple**).



Va detto che, attualmente, per pubblicare per dispositivi **Apple** è necessario effettuare la compilazione del progetto **su sistemi operativi Mac**, mentre è possibile pubblicare per **Android** anche da sistemi **Win**, a patto di avere la **licenza Pro**; sul primo punto, comunque, il team di **Game Salad** attualmente è al lavoro per consentire la **compilazione per dispositivi Mac e iOS** anche su sistemi **non Apple**.

* * *

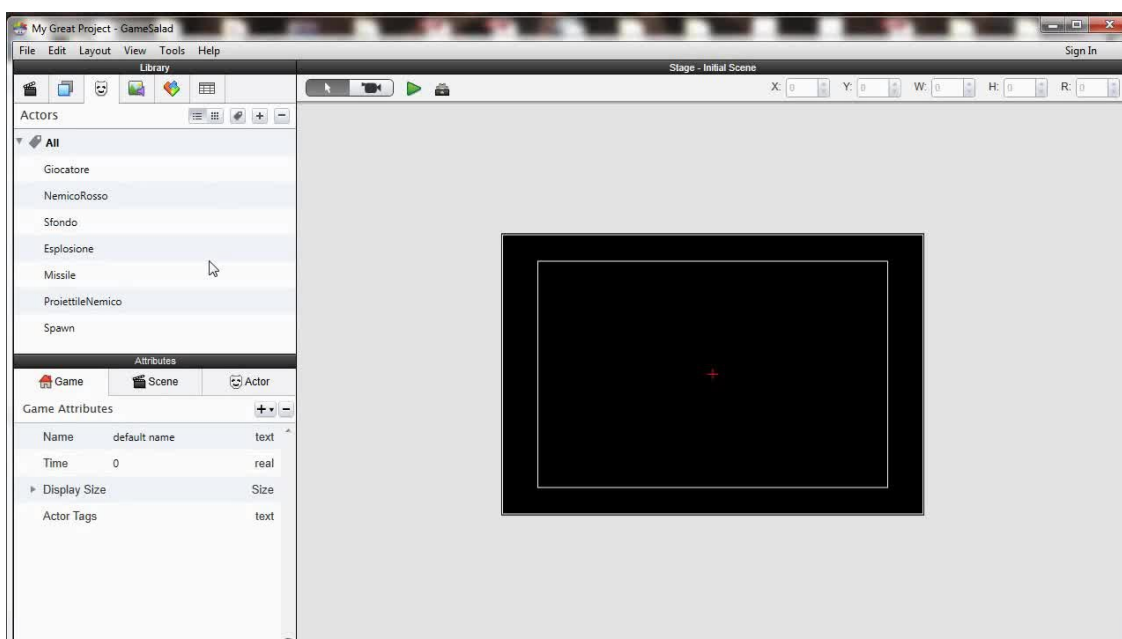
Introduzione a Game Salad - Tutorial 10

Realizzazione pratica di un semplice livello di gioco

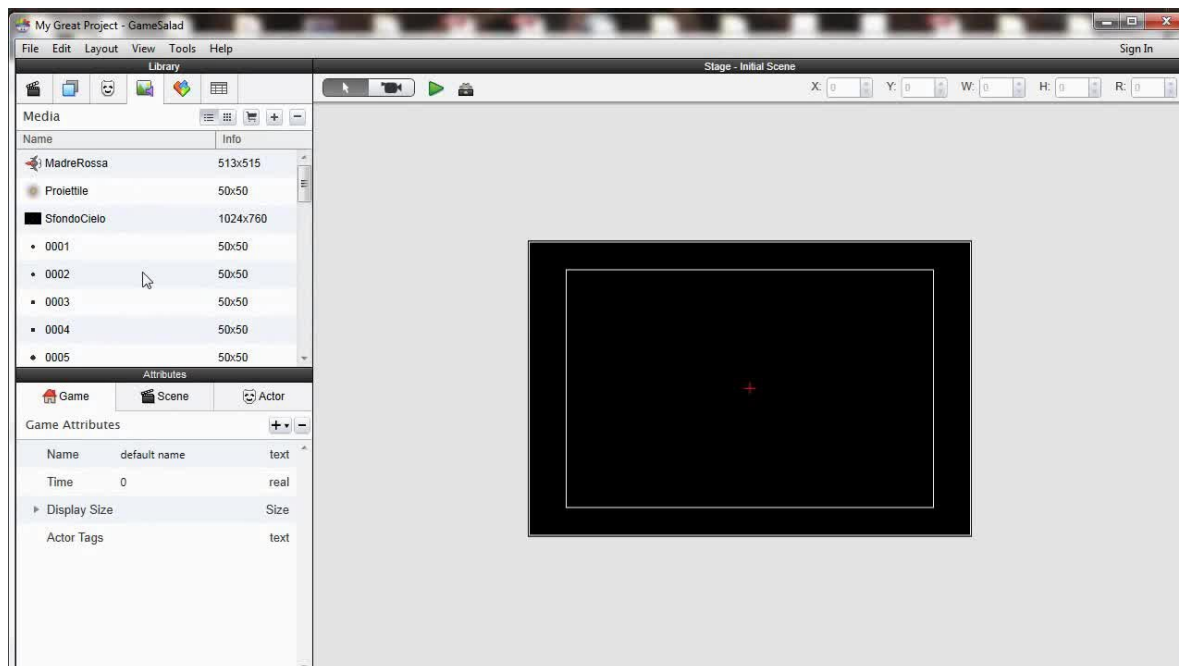
In questo capitolo (l'ultimo sulle basi di **Game Salad**), vedremo come realizzare in poco tempo una semplice schermata di gioco (un livello di un gioco più ampio), dove potremo pilotare un'astronave che dovrà abbattere le astronavi nemiche... tutto qui, niente menù o altre schermate, solo azione.

Si tratta di uno scenario semplicissimo, che tuttavia ci consente di **mettere insieme quanto visto finora** e di analizzare anche le **espressioni**, ossia dei campi particolari di certi *Behavior* che ci consentono di **recuperare dati e attributi della scena** o di altri oggetti (qualcuno di voi avrà notato, in un paio di capitoli precedenti, dei click sulle icone con la “*e*” o con la “*a*” e il recupero di *Position.x* e altro, comunque qui vedremo tutto con calma).

Ho creato un nuovo *Progetto Game Salad* scegliendo, come tipo, *Platform “GameSalad Arcade”*, in modo da poter poi inglobare il gioco creato **nelle pagine del nostro sito web**.

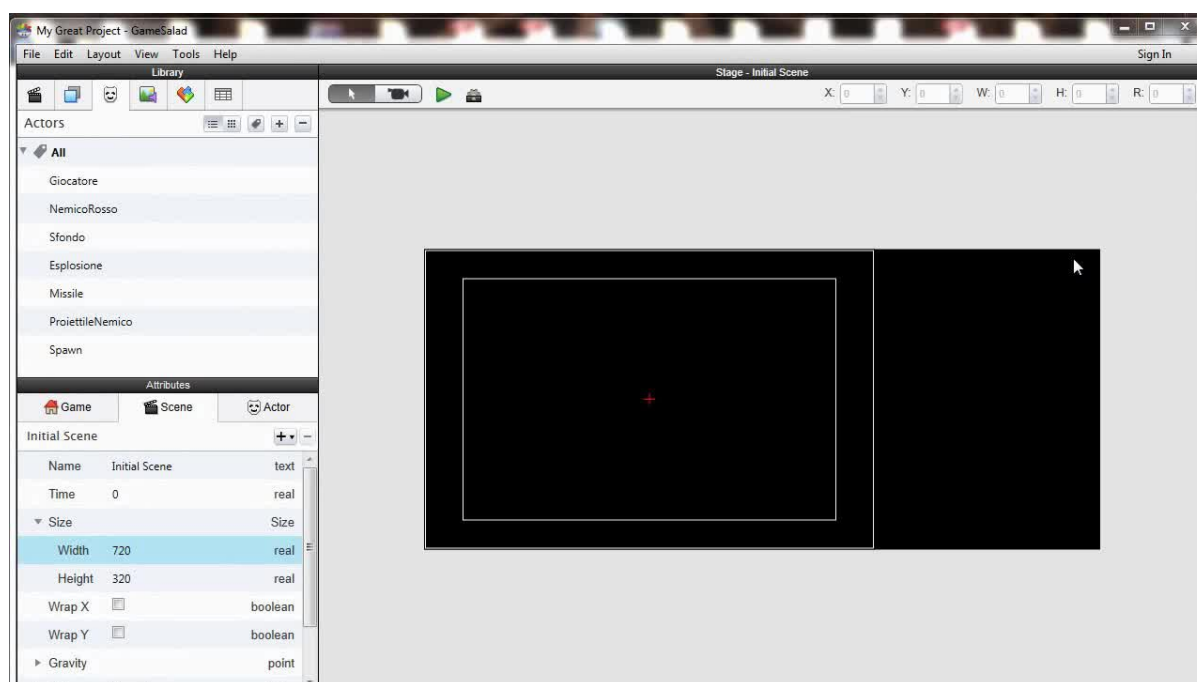


Nella **Library** ho importato quindi tutti i **media files** che ci serviranno: astronavi, proiettili, animazione di esplosione e sfondo.



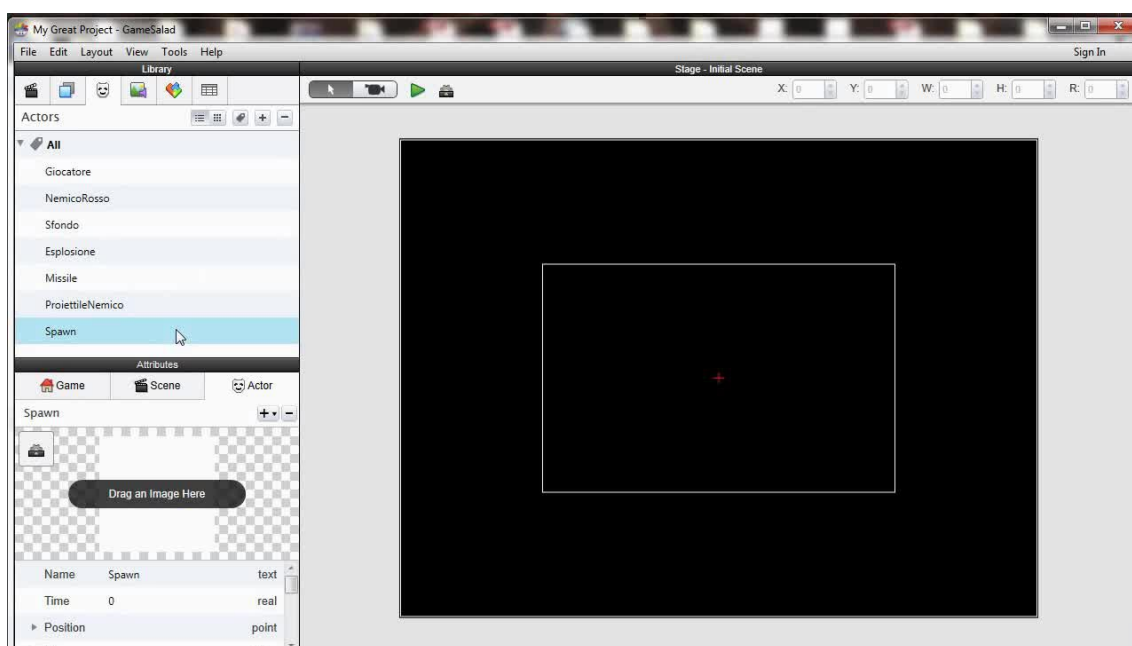
Opero quindi sulla **Scena** di base: il nostro gioco inizierà direttamente con quest'unico **livello**.

Di base, **lo scenario e la telecamera** hanno larghezza 480 e altezza 320; siccome voglio inglobare il gioco nel sito, con risoluzione 720x540, inserisco questi valori in *Width* e *Height* per *Size* e *Camera* nella scheda **Attributes** della *Scena*.

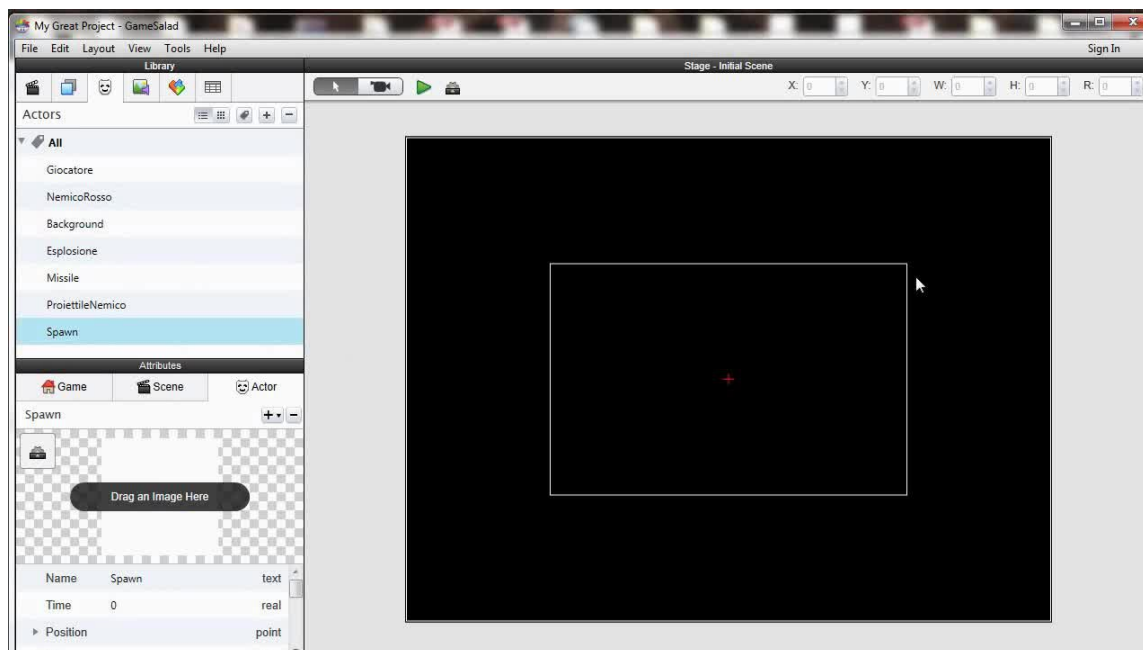


Creo quindi i **7 Actors necessari** per questo livello di gioco:

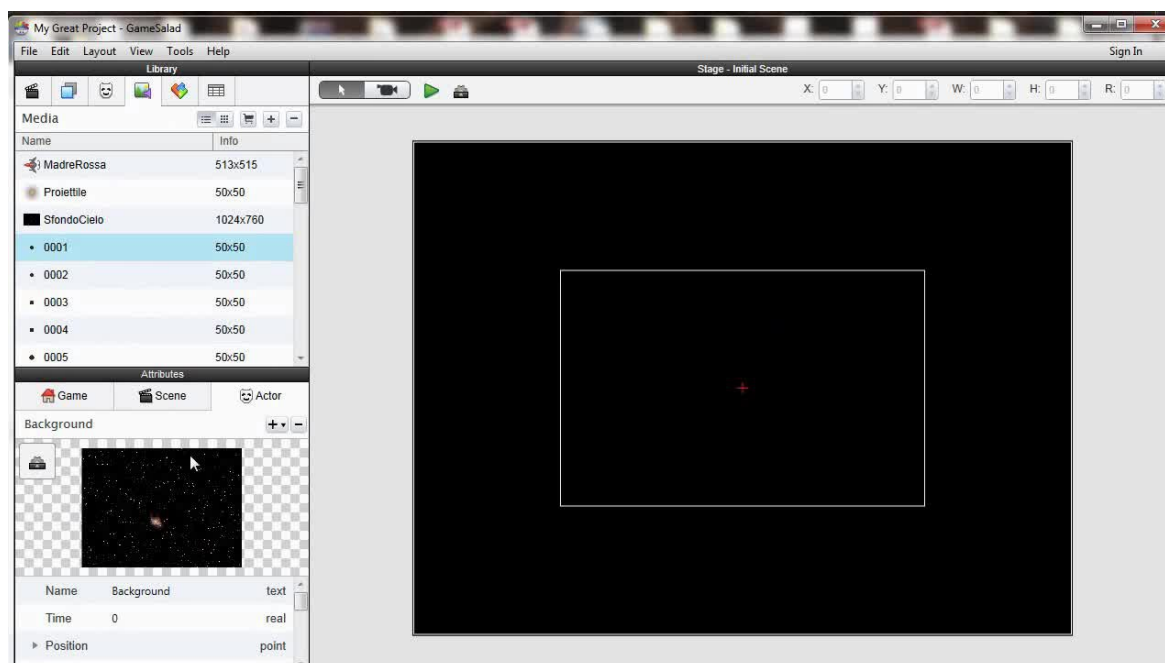
1. **“Giocatore”**, l'astronave che piloteremo usando i tasti freccia;
2. **“NemicoRosso”**, il prototipo delle astronavi nemiche;
3. **“Missile”**, il prototipo della nostra arma, ovvero dei missili;
4. **“ProiettileNemico”**, il prototipo dell'arma nemica, ovvero – in questo caso – delle sferette luminose;
5. **“Spawn”**, ovvero il punto nel quale verranno generati, con posizioni verticali casuali (random) i nemici, a destra dello schermo;
6. **“Background”**, cioè un'immagine di sfondo – che per semplicità manterremo statico, in questo progetto – per la Scena;
7. **“Esplosione”**, il prototipo dell'animazione di un'esplosione, che si verifica quando colpiamo le astronavi nemiche (o veniamo colpiti).



Di questi elementi, dobbiamo **trascinare fisicamente nella scena** solo *Giocatore*, *Background* e *Spawn*: gli altri oggetti verranno **creati dinamicamente**, in base agli eventi del gioco.



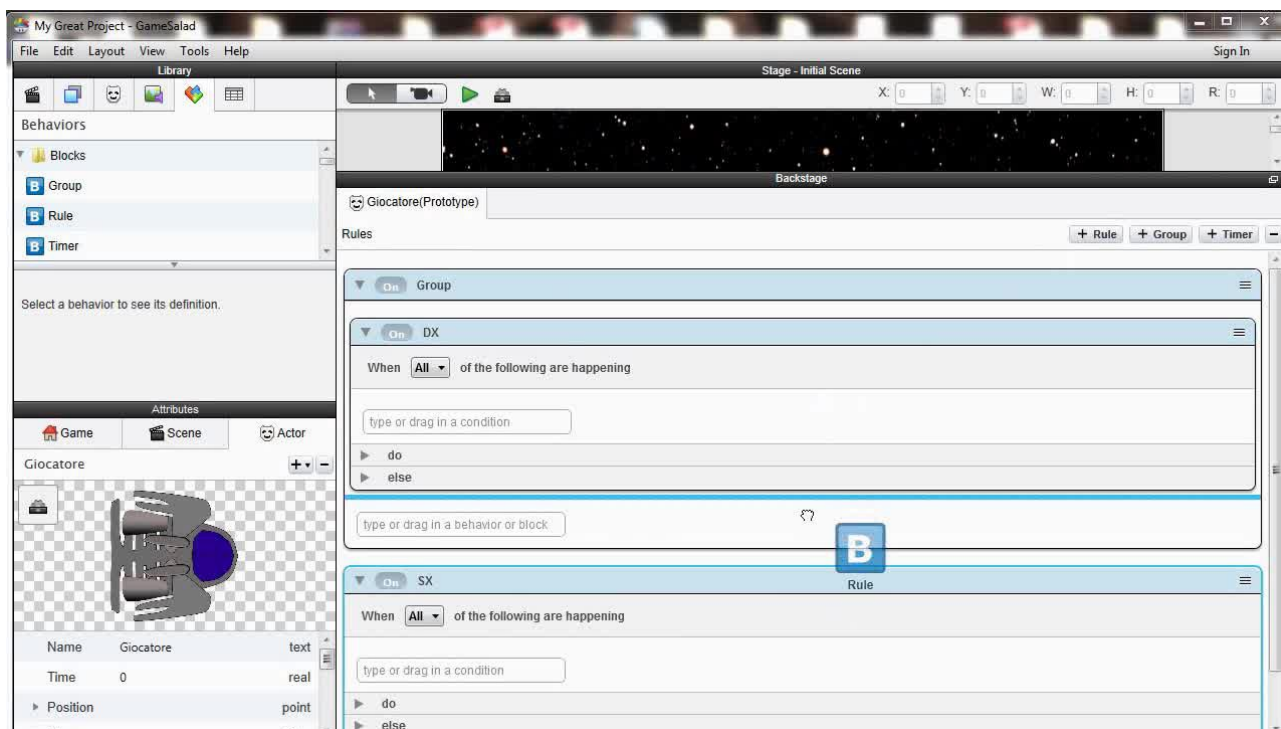
Lo **sfondo** è la cosa più semplice da inserire (almeno, in questo progetto, dove è **statico** e non è previsto lo **scrolling**): lo si trascina nella scena e si impostano, interattivamente oppure digitando i valori precisi nella sezione *Attribute*, coordinate d'origine e dimensioni, in modo da coprire tutto lo sfondo.



Iniziamo quindi dando all'Actor “**Giocatore**” la sua immagine e creando **Rules** e **Behavior** nel *backstage*.

Per prima cosa, il **movimento**: associamo la pressione dei tasti “freccia” al movimento dell'astronave, facendo in modo tra l'altro di mantenerla all'interno del rettangolo di gioco.

Creiamo quattro **Rules**, inserendole per comodità di lettura in un **Group**: **DX**, **SX**, **SU**, **GIU**.



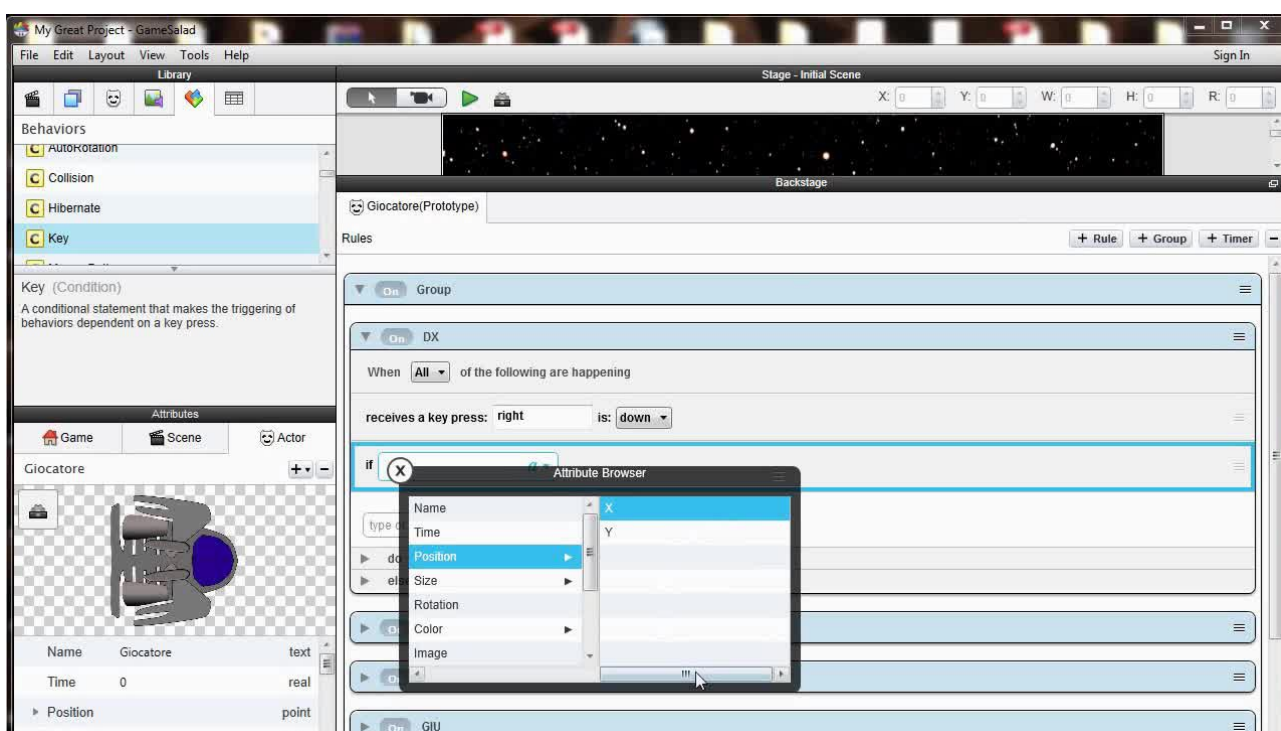
Tutte loro avranno, come **condizioni** (*When*), “**Key**” (e quindi, a seconda dei casi, premeremo i tasti freccia destra, sinistra, sopra, sotto) e una condizione **Attribute**.

La condizione **Attribute** ci consente, cliccando sulla “a” visibile nel blocco, di accedere, “navigando” negli elementi di gioco (con la finestra “**Attribute Browser**” che apparirà), ai campi che ci interessano; i campi **Position** X e Y (e, quindi, i valori di posizione nella scena) dell'attore possono essere raggiunti con *Attributes – Giocatore (il nome dell'attore) – Position – X* o *Y*.

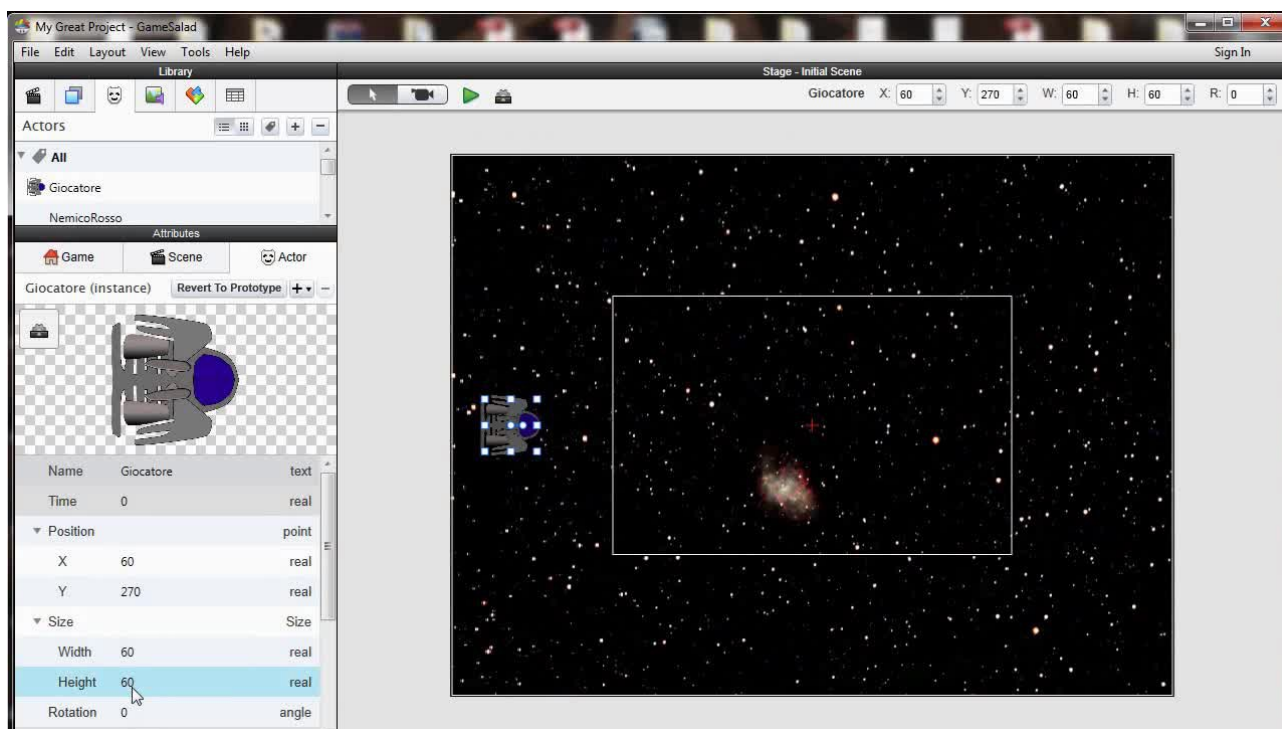
Le **condizioni** sulla posizione e i **Move** avranno valori diversi a seconda del tasto premuto, perché dobbiamo inserire, nel primo caso, i **limiti x e y della schermata**, mentre nel secondo dovremo

impostare la **direzione** (che si misura in gradi, da 0 verso destra ruotando); in particolare:

- per lo spostamento **a destra**, *Position.x* deve avere valore minore di 660 (visto che la dimensione della schermata è 720 e l'Actor ha dimensione 60); in caso positivo, *Move* avrà direzione 0 e velocità 300;
- per lo spostamento **a sinistra**, *Position.X* dovrà essere maggiore di 60 e, in tal caso, *Move* avrà direzione 180° e velocità 300;
- per lo spostamento **verso l'alto**, *Position.Y* minore di 540, *Move* 90° con velocità 300;
- per lo spostamento **verso il basso**, *Position.Y* maggiore di 60, *Move* 270° e velocità 300.



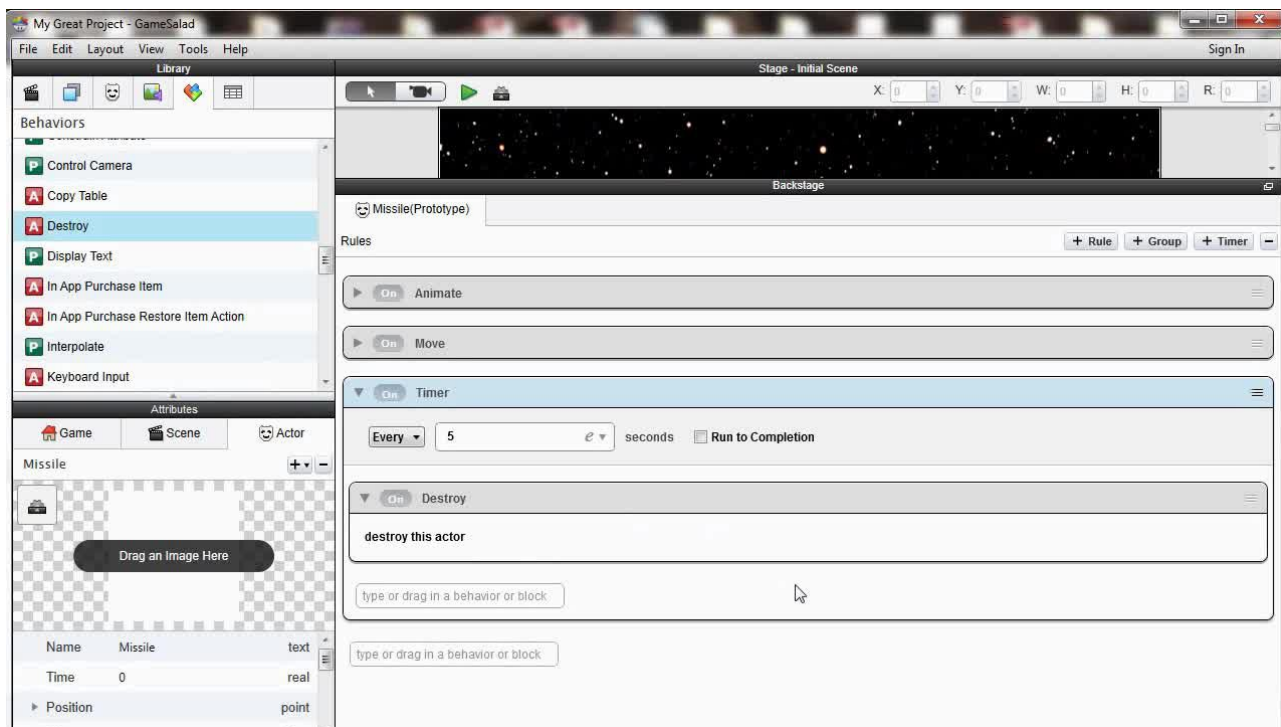
Per provare il tutto è sufficiente **trascinare l'attore dell'astronave** dentro la scena (magari impostando, se necessario, *Position X* 60 e *Y* 270, *Size X* e *Y* a 60, nella scheda *Attributes*) ed eseguire la *preview*.



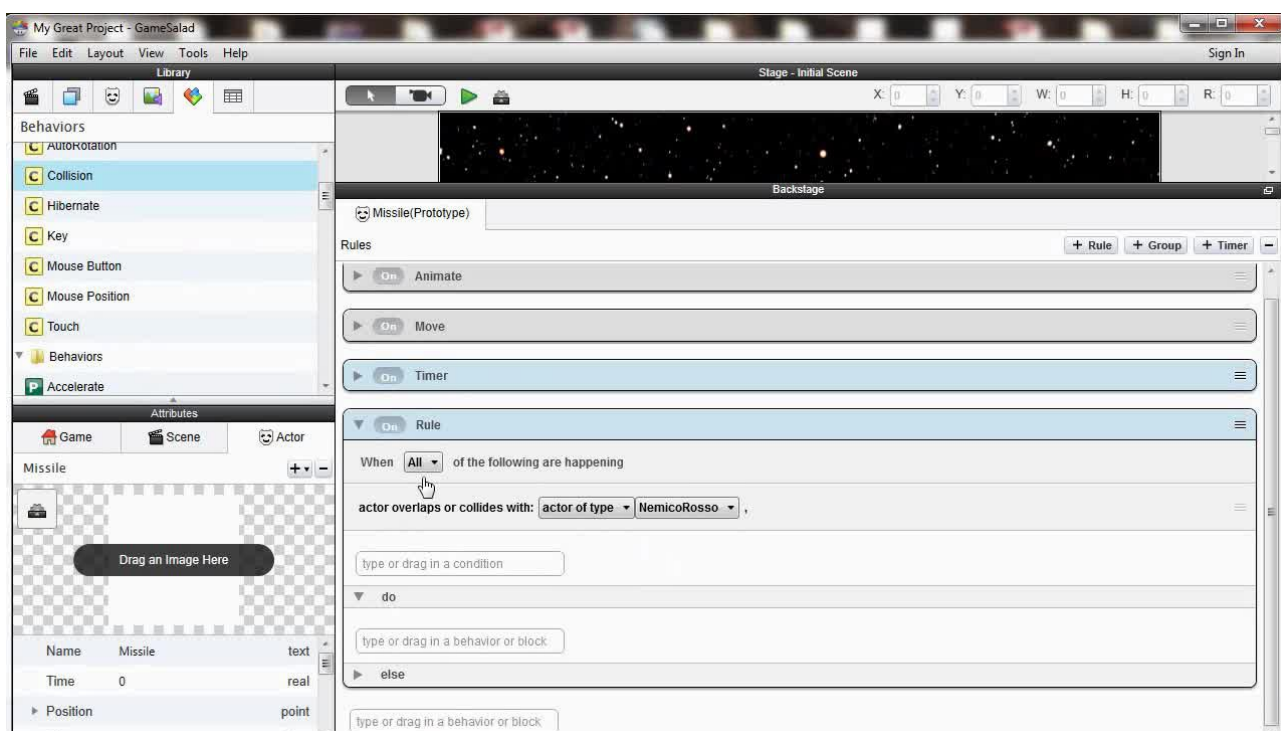
Passiamo quindi al **Missile** sparato dalla nostra astronave.

Apriamo il *backstage* e impostiamo i seguenti **Behaviors**:

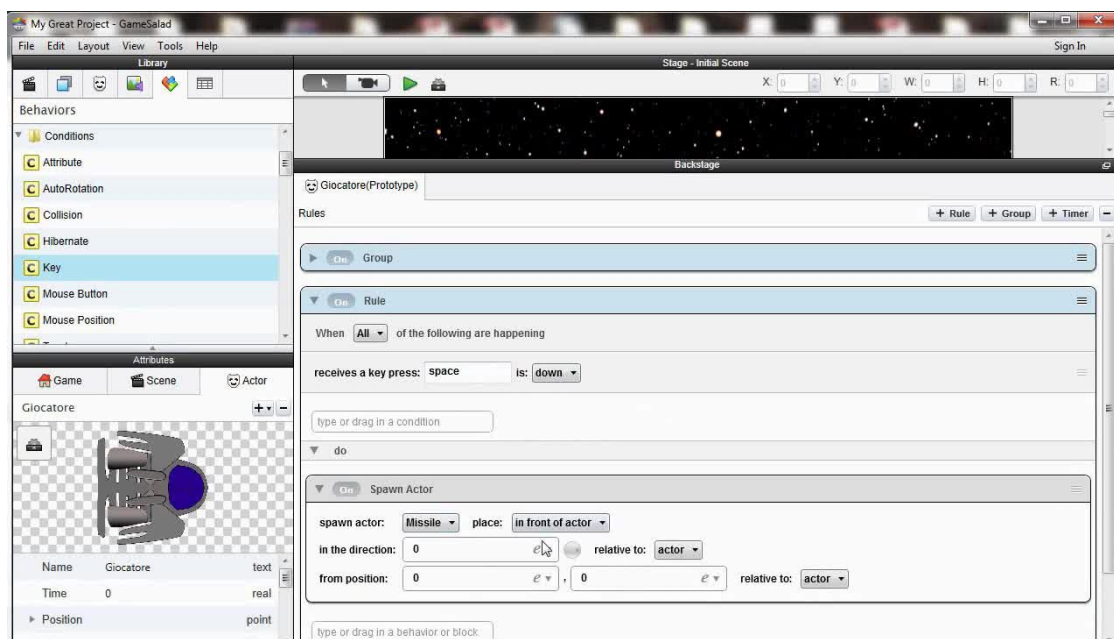
- **Animate**, per animare l'aspetto visivo dell'oggetto quando viene creato, con un *Loop* e disattivando la seconda checkbox (che ripristina l'aspetto iniziale al termine dei fotogrammi), trascinando nell'area bianca vuota le varie immagini e regolando la velocità di animazione (frame rate, fps) a piacere; nella scheda *Attribute*, impostiamo *Size X* e *Y* dell'*Actor* in base alle dimensioni dell'immagine (nel mio caso, 50x15); se abbiamo un file audio, possiamo mettere un **Play Sound**, in modo da eseguire un file sonoro (che dovremo caricare come *Media* e scegliere mediante la checkbox del *Behavior*) quando l'*Actor* viene caricato nella scena, ovvero quando spariamo un colpo;
- **Move**, per spostare il missile (in questo caso con movimento lineare, ma se volete potete provare *Accelerate*), con direzione 0 e velocità 500;
- **Timer**, con valore 5, con all'interno un **Destroy** per distruggere l'oggetto dopo 5 secondi (in modo da assicurarci di togliere l'oggetto dalla memoria del dispositivo, liberando risorse e lavoro per il processore, quando questo esce dalla parte destra dello schermo).



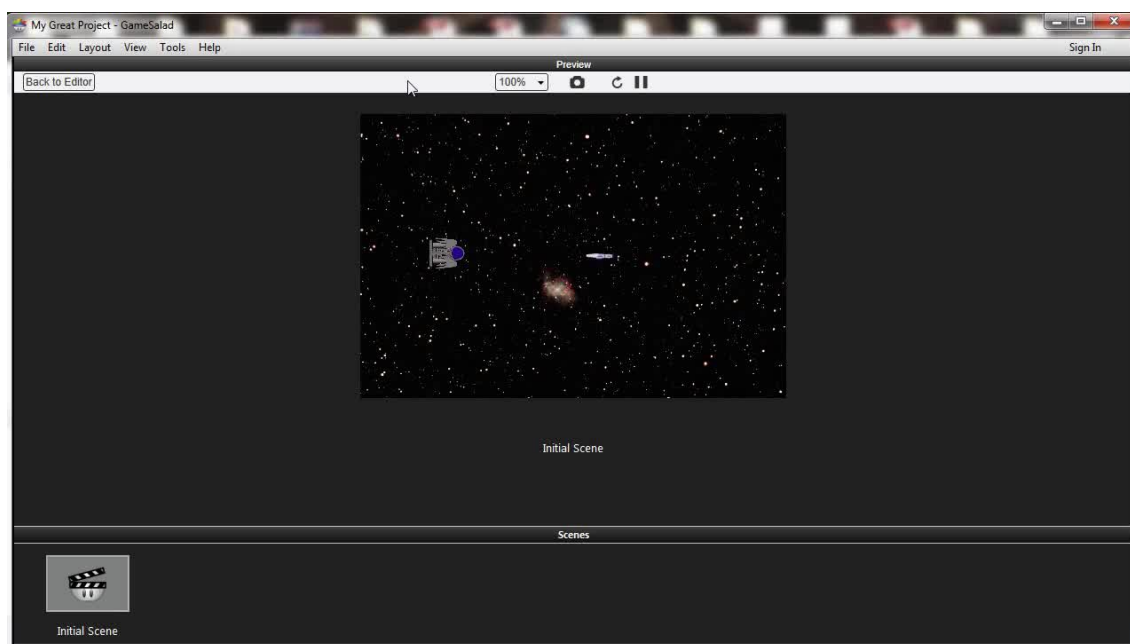
Impostiamo inoltre una **Rule**: se il proiettile va a collidere (**Collision**) con l'attore nemico (nel nostro caso, “*NemicoRosso*”), allora **distruggi** il proiettile (gestiremo dall'attore del nemico l'altra parte della collisione, ovvero la distruzione del nemico).



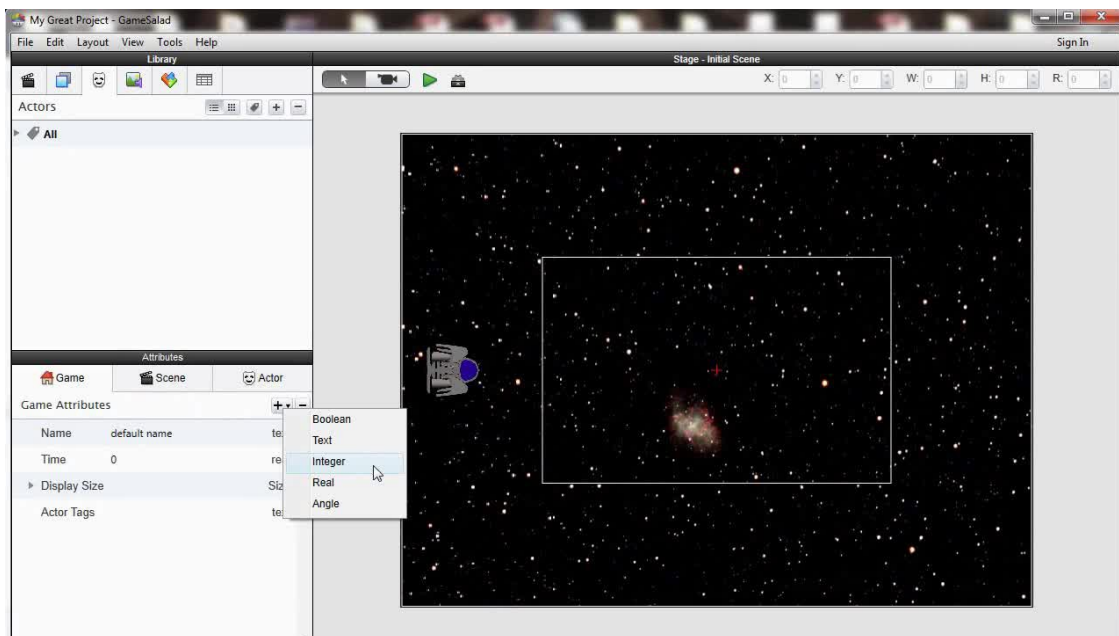
Torniamo all'astronave “*Giocatore*” ed impostiamo la **regola** del “*fire*”, per sparare un missile, alla pressione del tasto “*space*”, la barra spazio: è sufficiente uno **Spawn Actor**, indicando il *Missile* come oggetto da produrre, senza specificare origine (di default, sarà il punto ove si trova la navicella) o movimento, visto che l'oggetto *Missile* ha già un **Behavior Move**.



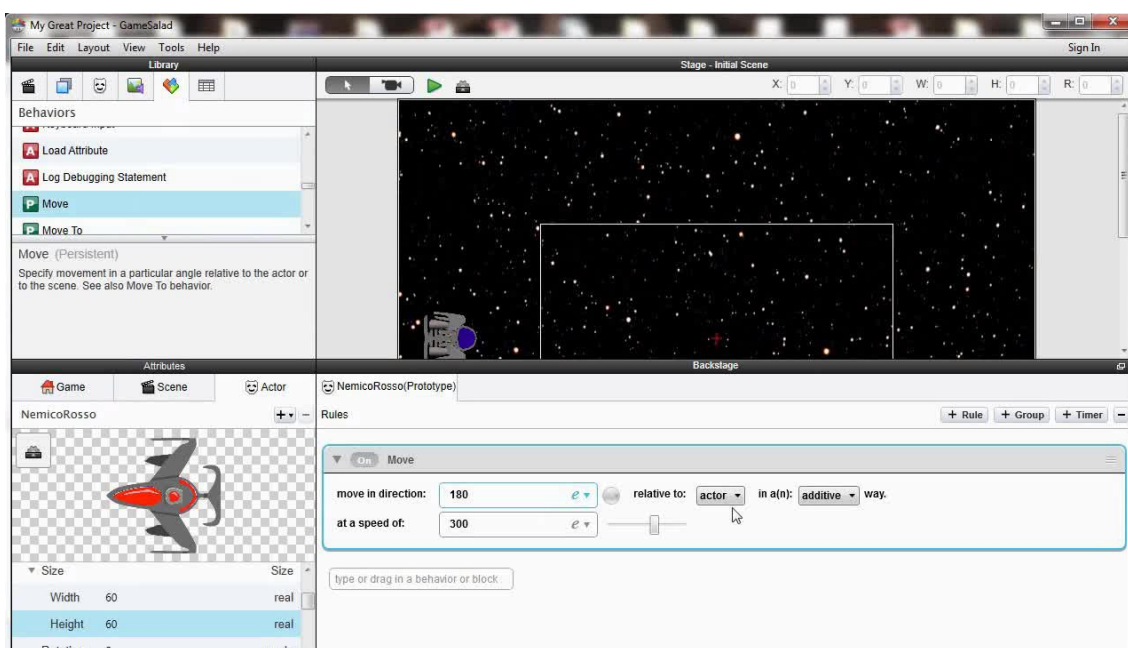
Eseguiamo il **Play** della scena per testare quello che abbiamo fatto fino ad ora.



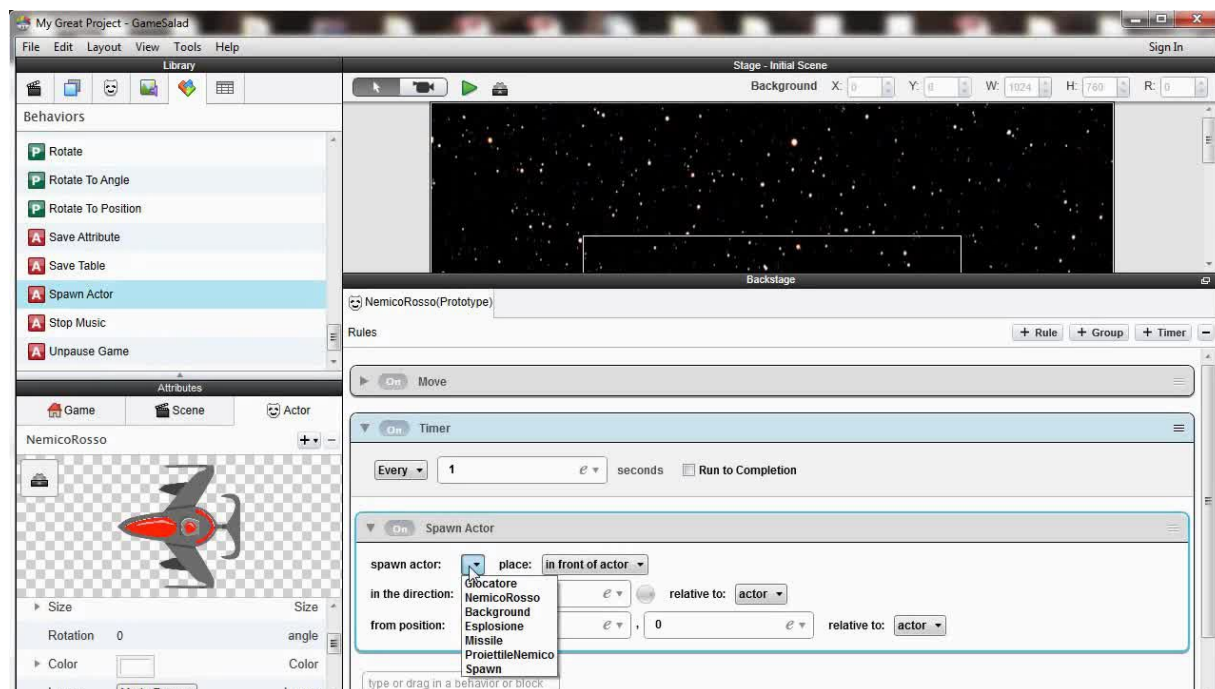
Per gestire le **vite** (o, se volete, **l'energia** del giocatore), dobbiamo creare un **Attribute** personalizzato, tuttavia non possiamo crearlo per il *Giocatore* perché gli altri oggetti di gioco non potranno leggerlo; tutti gli oggetti, però, possono leggere e scrivere **attributi del Game** o delle **Scenes**, per cui aggiungiamo un **Attribute** proprio per il **Game** (visto che qui c'è solo questo livello), di tipo **Integer** (numero intero), chiamato “*Vite*” e impostandolo a 10.



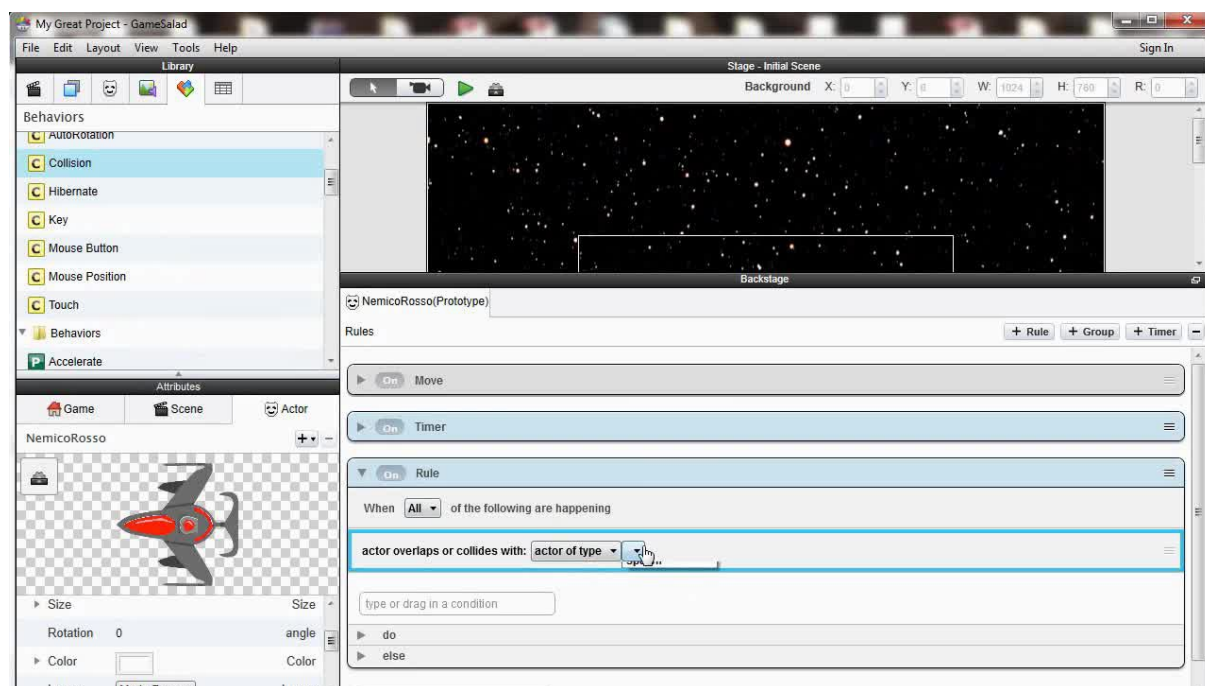
Passiamo adesso al “*NemicoRosso*” (con immagine dell'astronave, da trascinare nello slot **Attribute** dell'*Actor*, impostando poi SizeX e Y a 60), che si muoverà in direzione 180 e sparerà ogni secondo. Impostiamo quindi un **Behavior** “*Move*” con direzione “180” a velocità “200”.



Dentro un “**Timer**” in modalità “*Every 1 seconds*” aggiungiamo il **suono** dello sparo (se ne abbiamo uno) e lo **Spawn Actor** dell'oggetto *Proiettile Nemico*.

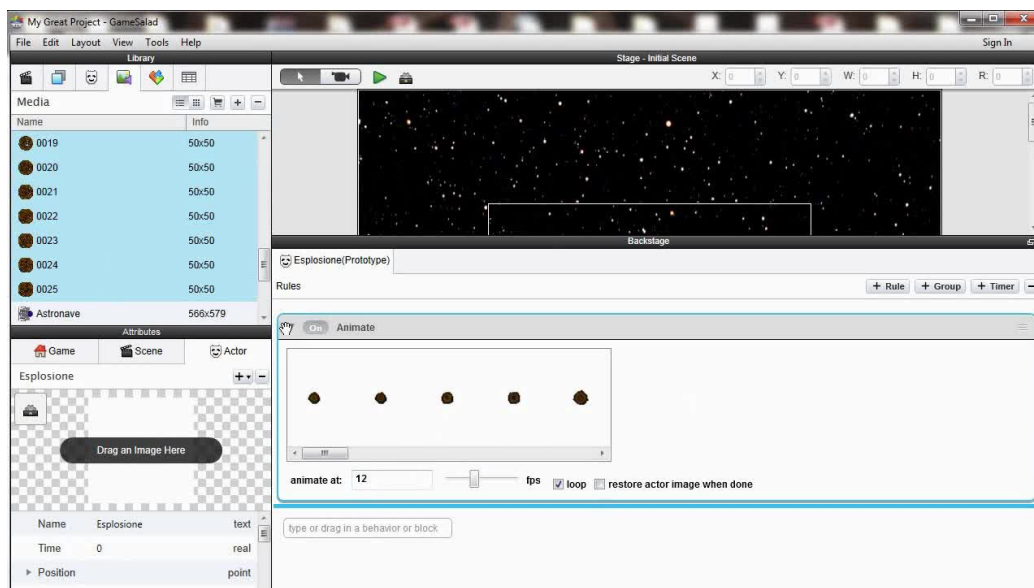


Impostiamo inoltre una **Rule**: in caso di collisione con *Missile*, facciamo uno **Spawn** dell'attore *Esplosione* e un **Destroy** del nemico stesso).

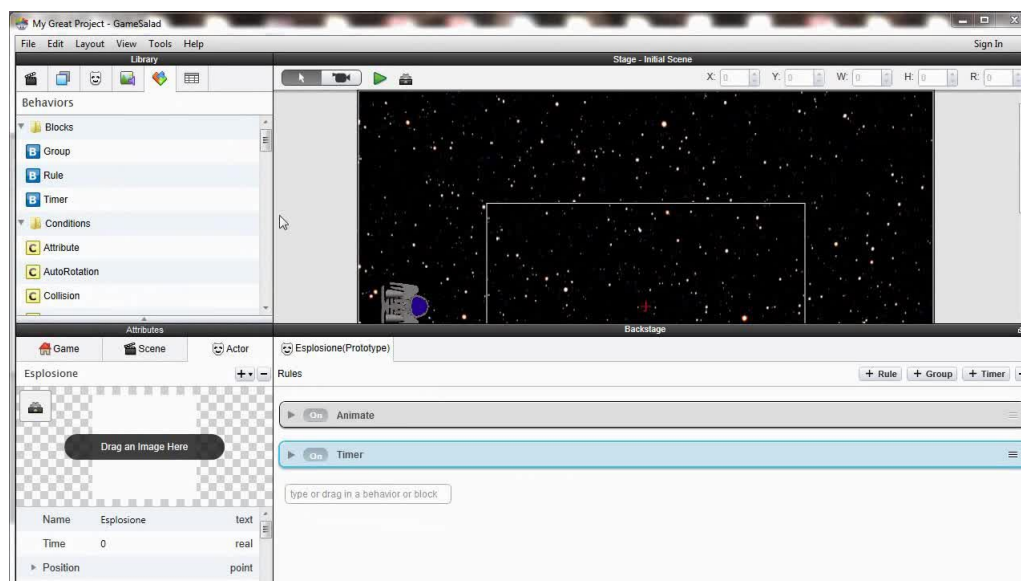


Non abbiamo parlato, in effetti, di **Esplorazione**: dobbiamo fornirgli un **Animate** e passargli le *sprites* (le sequenze di immagini o fotogrammi) che servono per rappresentare l'animazione dell'esplosione, ma dobbiamo dargli anche un **Timer – After** con un **Destroy**, per toglierlo al termine dell'animazione... già, ma *quando termina l'animazione?*

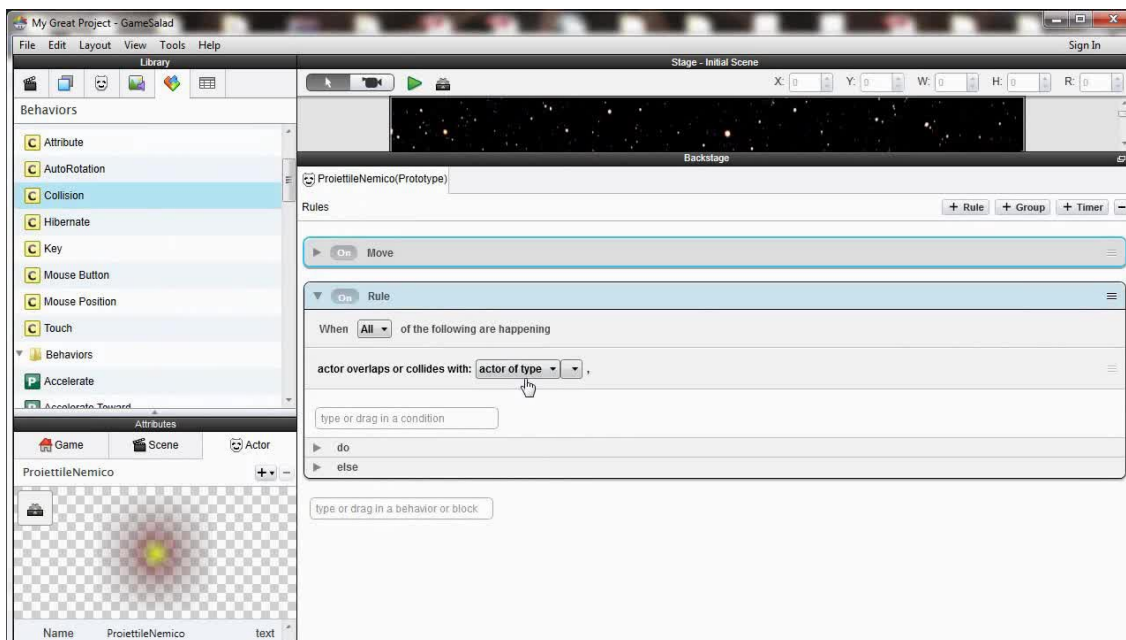
Qui dobbiamo fare due conti: se ad esempio le *sprites* sono 60 a 15 frame al secondo, avremo $60/15 = 4$ secondi, quindi **Destroy** dopo 4 secondi.



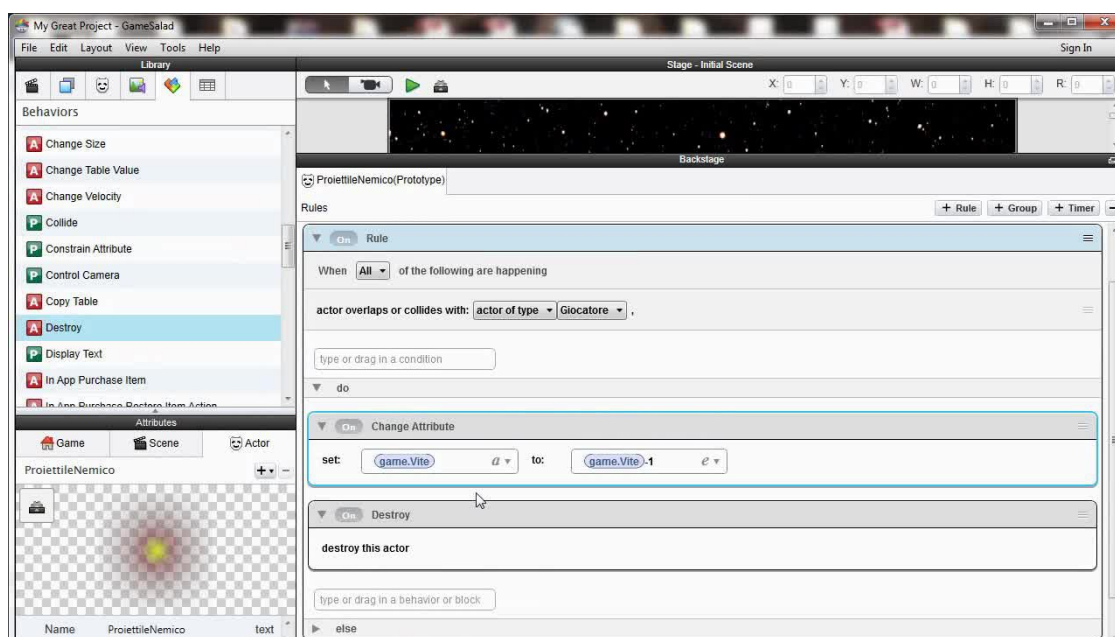
Selezioniamo adesso il “**Proiettile Nemico**” per aggiungergli l'immagine (nel mio caso una sola, statica, a differenza del missile, che aveva **Animate** e delle *sprites*; impostiamo Size X e Y a 50). Configuriamone quindi il movimento, con un “**Move**” in direzione “180” e velocità 300 o più.



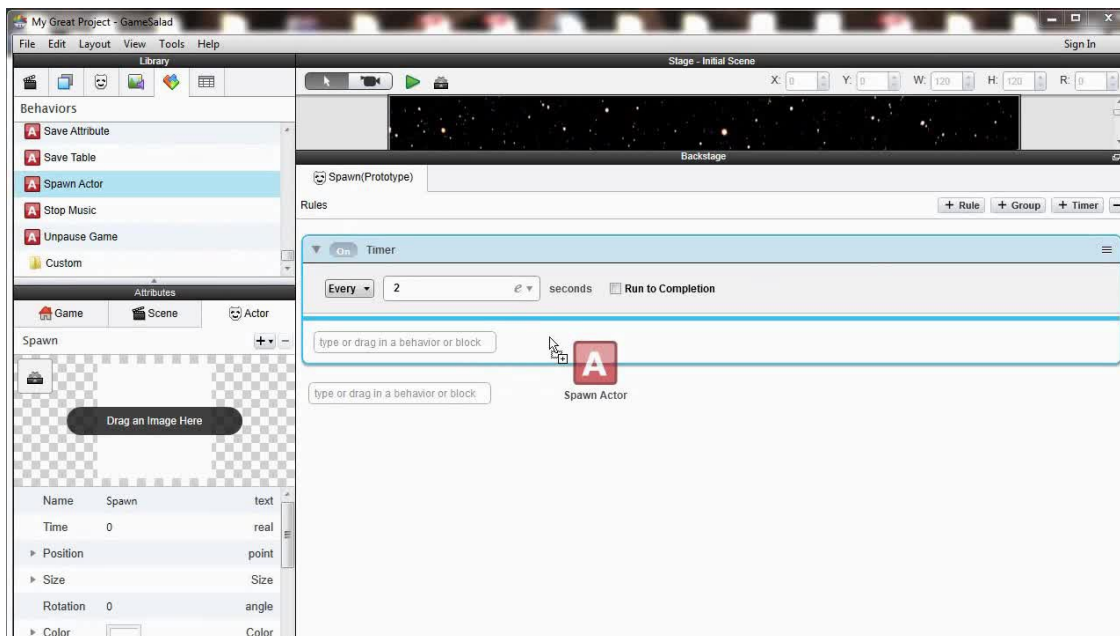
Sempre per **ProiettileNemico** impostiamo la seguente **regola**: se si ha una collisione con “*Giocatore*”, allora si attiva un **Change Attribute** che va a recuperare l'attributo globale del gioco “*Vite*” e lo imposta a “*Vite-1*” (ossia, sottrae un'unità); distruggiamo, inoltre, l'oggetto **ProiettileNemico**, per via dell'impatto.



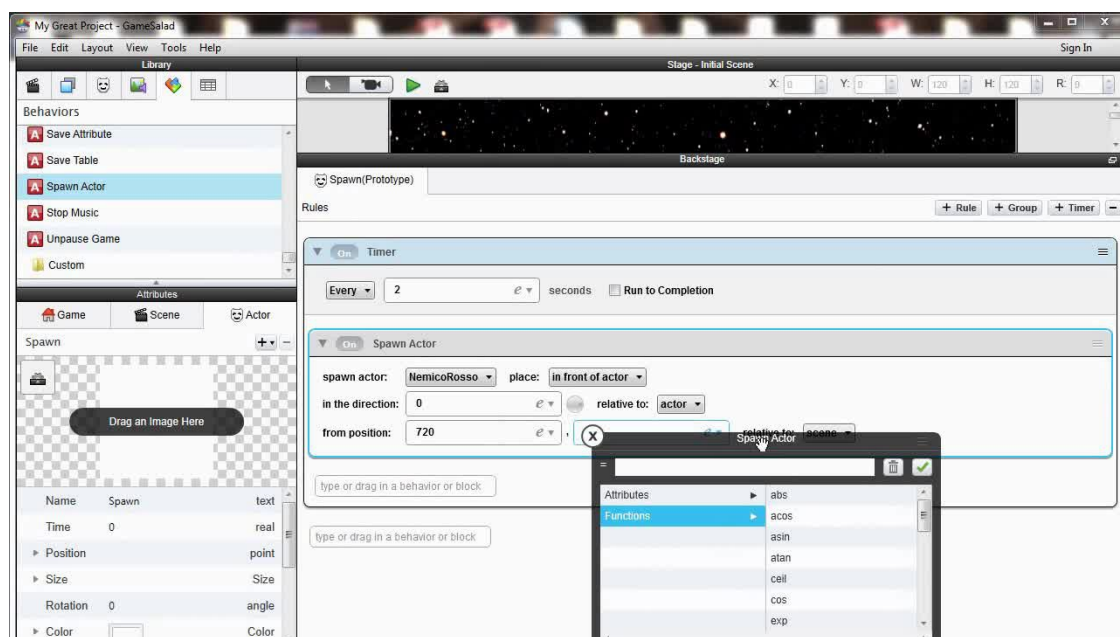
Per eliminare il **ProiettileNemico** quando non serve più (ossia, finisce a sinistra dello schermo), possiamo mettere un **Timer** con **Destroy**, come fatto con il nostro missile, oppure una **Rule** che verifica la posizione dell'Attributo **Position.X** dell'oggetto e attiva un **Destroy** se questa è minore di 0... come vedete, anche qui sono disponibili più soluzioni, a voi la scelta.



Passiamo quindi all'oggetto **Spawn**, che genera le astronavi nemiche: con un **Timer** “*Every 2 seconds*”, facciamo uno **Spawn Actor** di “*NemicoRosso*”, con direzione “*0 relative to Actor*”, mentre per le posizioni metteremo *Relative to Scene*, con *Position X* “*720*” e, per la posizione Y, un'espressione che restituisce un valore casuale tra 100 e 500, in modo da generare i nemici ad altezze differenti.



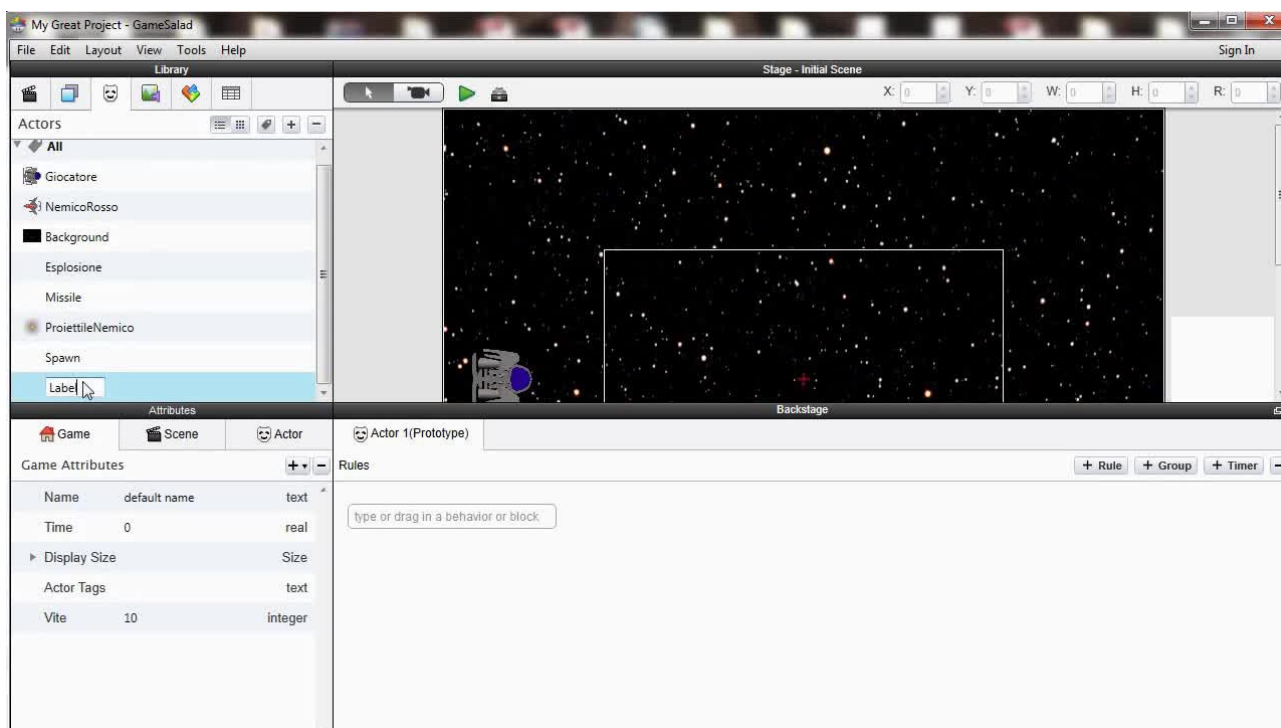
Per creare un'espressione, clicchiamo sull'icona con la “*e*” accanto al campo, quindi scegliamo “*Functions – Random*” dalla finestra che apparirà e facciamo doppio click; nella scritta che apparirà nel campo Y, sostituiamo *min* e *max* con 100 e 500.



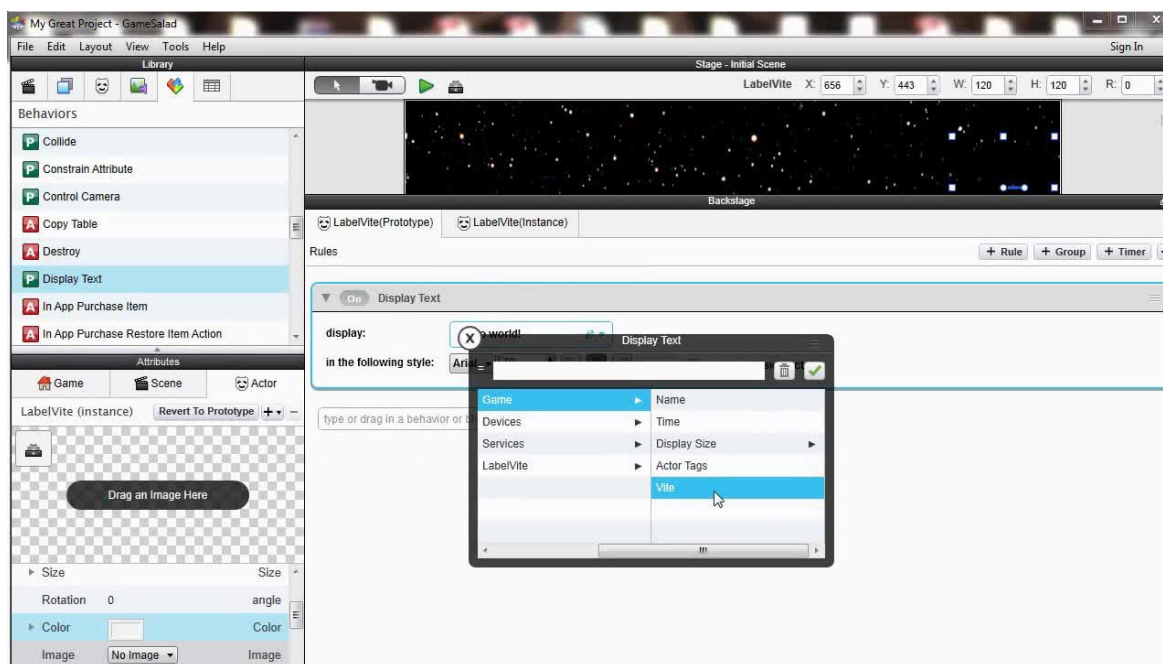
Come avrete intuito, è possibile generare vere e proprie espressioni, più o meno complesse, combinando i dati “**Attributes**” (quindi, ad esempio, un punteggio, il numero di munizioni, le vite, le posizioni, ecc...) e delle funzioni matematiche messe a disposizione da **Game Salad** nell'elenco **Functions**.

Come detto nel primo capitolo, non si ha la possibilità di inserire script, tuttavia lo schema a “*mattoncini logici*” (**Actors**, **Attributes**, **Rules** e **Behaviors**), arricchito con le **Expressions**, consente comunque una grandissima varietà di applicazioni.

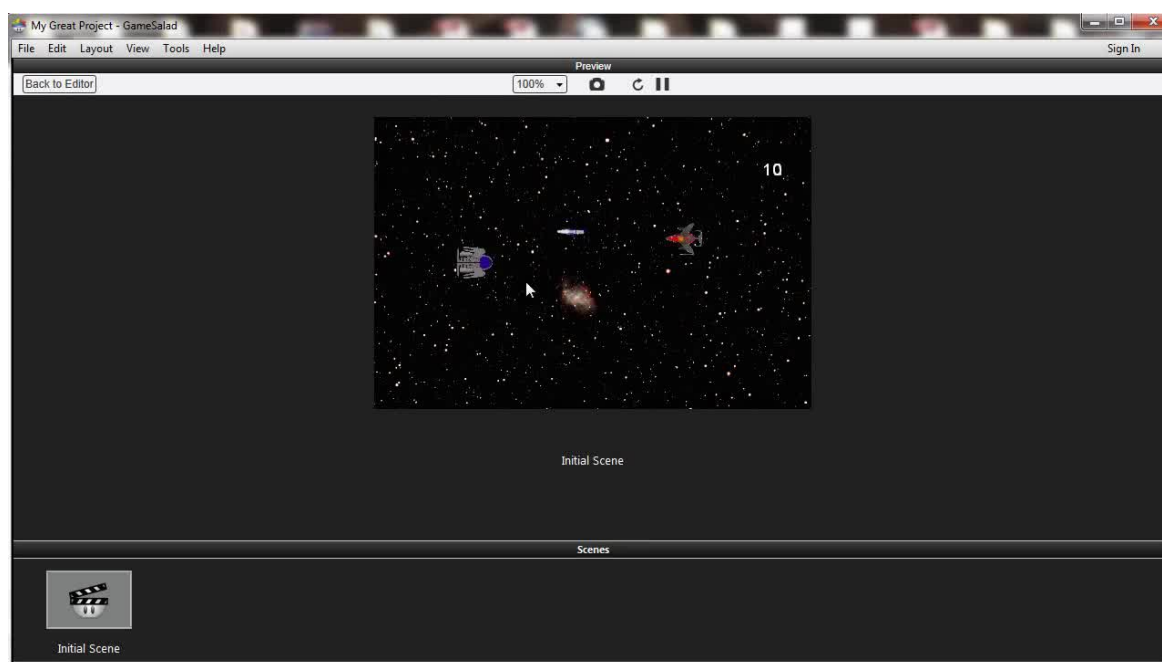
Per poter **visualizzare le vite a disposizione**, infine, creiamo un nuovo **Actor** vuoto, “*LabelVite*”, posizionandolo ad esempio in alto a destra nella schermata.



Clicchiamo su **Color**, in **Attribute**, e abbassiamo a 0 il valore **A** (*Alpha*, l'opacità), quindi diamogli un **Behavior** “*Display Text*” che mostrerà, con le modalità (colore e font) che specificheremo nel **Behavior**, il valore di *game.Vite*, ovvero il valore dell'**Attribute** “*Vite*” del gioco.



Mandando **in esecuzione**, vediamo che c'è un po' tutto (anche se forse manca l'autodistruzione per le navicelle nemiche che escono a sinistra... quelle poche che riusciranno a sfuggire ai nostri colpi!), ma... c'è un “*ma*”: i tasti freccia destra e sinistra **spostano lateralmente la nostra navicella**, cosa che non è esattamente un comportamento accettabile... ha molto più senso, invece, **ruotare la navicella** (con i tasti freccia laterali) e **farla andare avanti e indietro lungo la sua direzione** con i tasti freccia verticali.



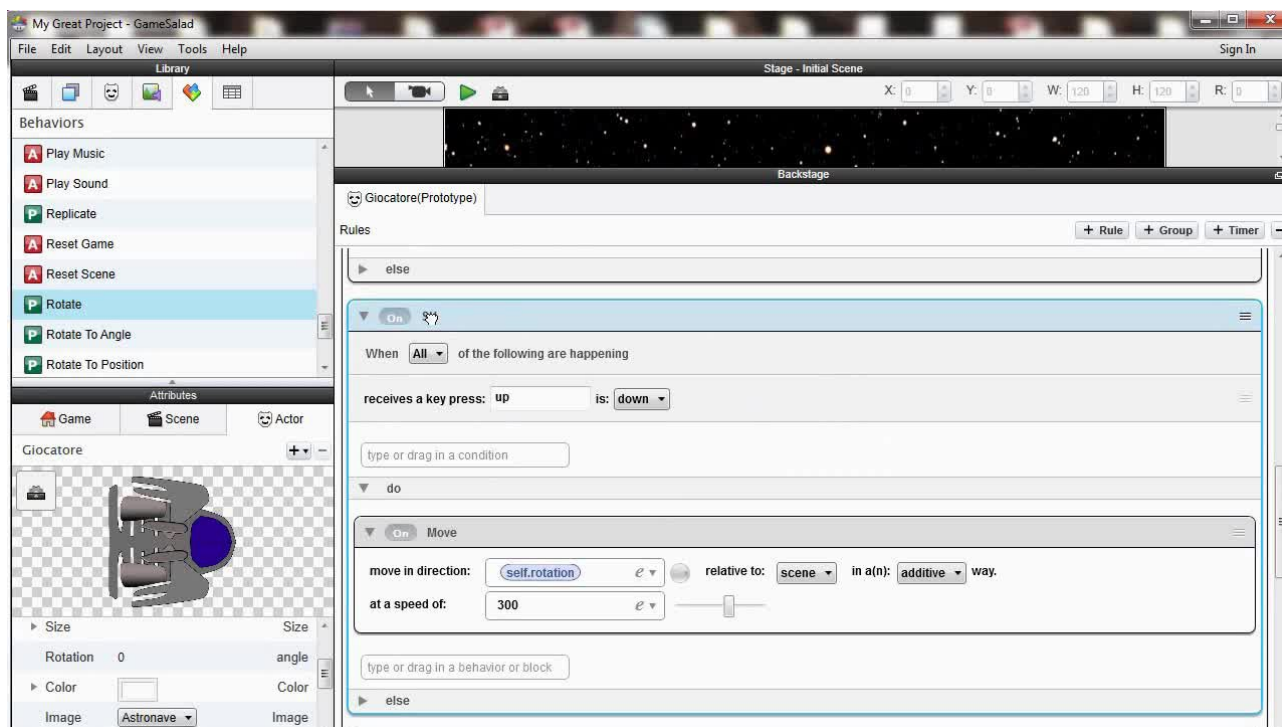
Ora che sappiamo **definire le funzioni**, possiamo farlo: selezioniamo l'astronave *Giocatore* e torniamo alle **Rules** di movimento.

Cambiamo la **Rule DX** togliendo il limite sulla posizione X, rimuovendo il **Move** e mettendo un **Rotate Clockwise 90**.

Cambiamo la **Rule SX** togliendo il limite sulla posizione X, rimuovendo il **Move** e mettendo un **Rotate CounterClockwise 90**.

Cambiamo la **Rule SU** (che ora sarà un *AVANTI*) togliendo il limite sulla posizione Y e impostando, per il **Move**, la seguente espressione: direzione **self.rotation** (ovvero, attributo *Rotation* di *Giocatore*, nella finestra che apparirà), con **Relative to Scene**, relativo alla scena.

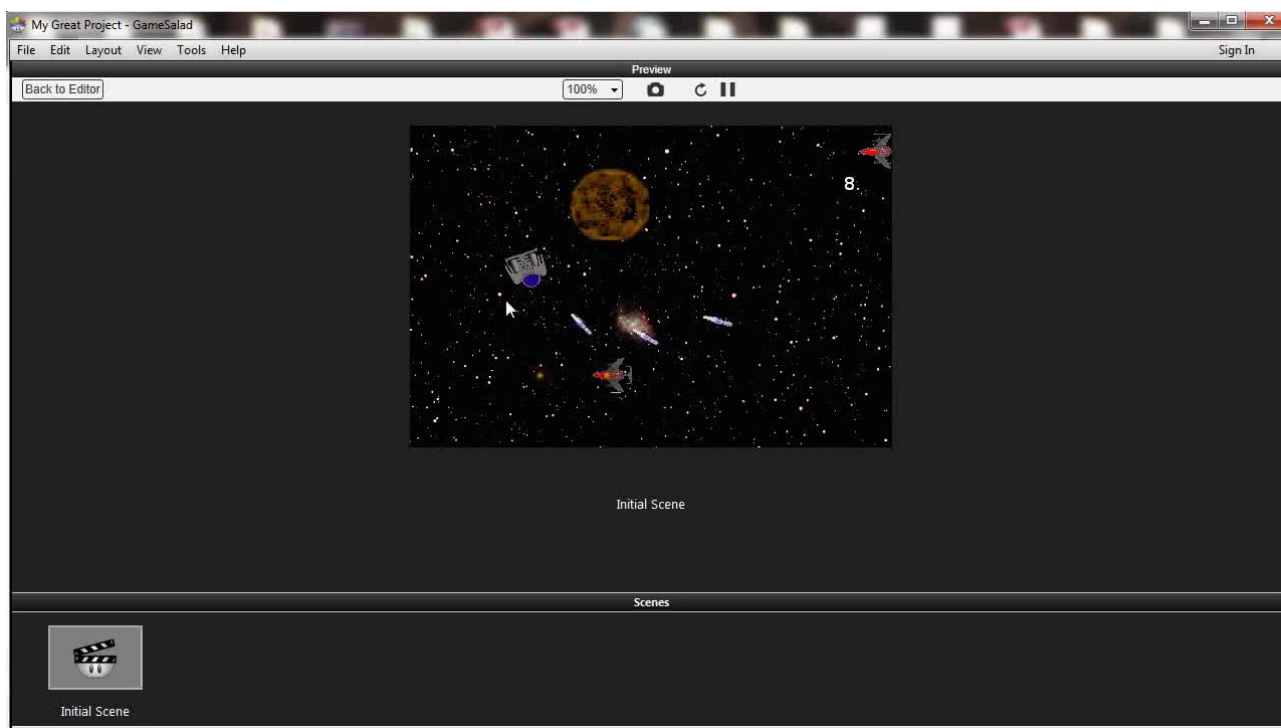
Cambiamo la **Rule GIU** (che ora sarà un *INDIETRO*) togliendo il limite sulla posizione Y e impostando, per il **Move**, la seguente espressione: direzione **self.rotation+180** (ovvero, attributo *Rotation* di *Giocatore*, nella finestra che apparirà, più 180, per andare nella direzione opposta, visto che *Rotation* è espressa in gradi), con **Relative to Scene**, relativo alla scena.



Tecnicamente ci siamo, per la realizzazione di un solo **livello di gioco**, tuttavia questa partita **non avrà mai fine**, non c'è un vero e proprio scopo (se non quello di abbattere nemici all'infinito)... prima di chiudere questo capitolo (e questa serie di 10 tutorial introduttivi su **Game Salad**), quindi, **un compito per casa**: gestire il **punteggio** del giocatore (ad esempio sommando 100 punti a una variabile globale... e vi sto già dicendo troppo...), in modo da aprire un'altra **Scena** quando le **vite** del protagonista arrivano a 0 e mostrare il **punteggio finale** ottenuto.

Altro compito per casa: risolvere il problema del “**confinamento**” dell'**astronave**, visto che abbiamo rimosso i controlli su *Position X* e *Y* per modificare il movimento.

Ci sono varie soluzioni disponibili... pensateci!



Bene, per questa introduzione a **Game Salad** è tutto!

* * *