

# Arduino: Videogame finale GALAGA

Giorgio Leonardi, Paola Giannini

<https://orienta.dir.uniupo.it/course/view.php?id=43>

oppure: <https://orienta.dir.uniupo.it> → informatica → Esperienza Formativa di Orientamento per le Scuole Superiori 2023/24: Arduino

# Cos'è Arduino

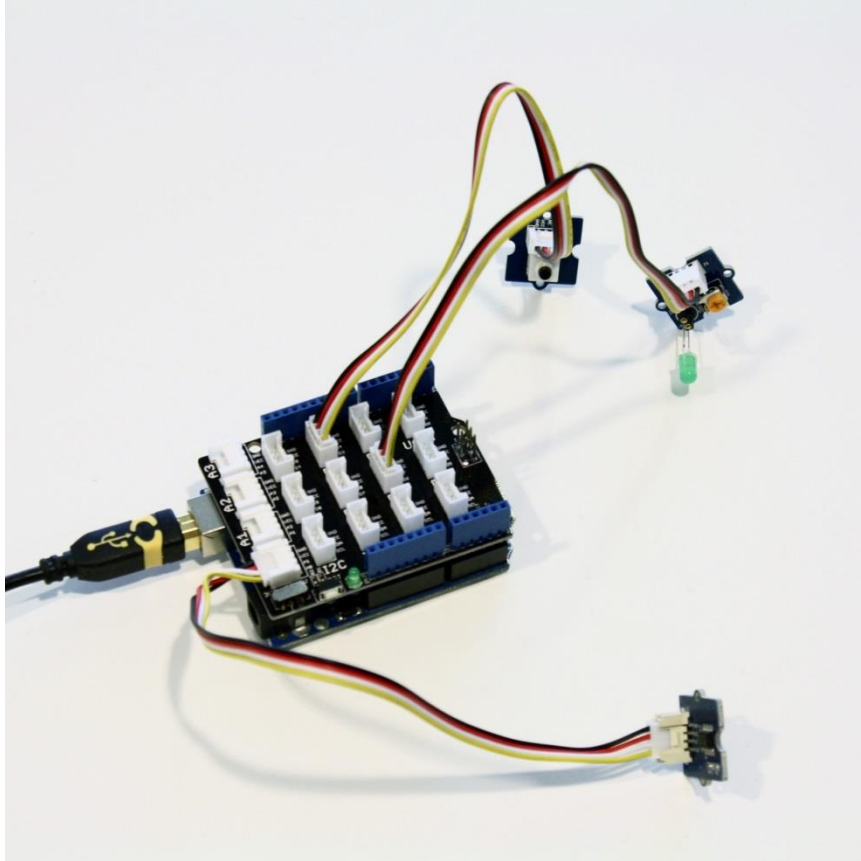


«Arduino è una piattaforma di prototipazione elettronica open-source basata su hardware e software flessibili e di facile utilizzo.»

**E' pensato per artisti, designer, hobbisti e chiunque sia interessato a creare oggetti o ambienti interattivi. »**



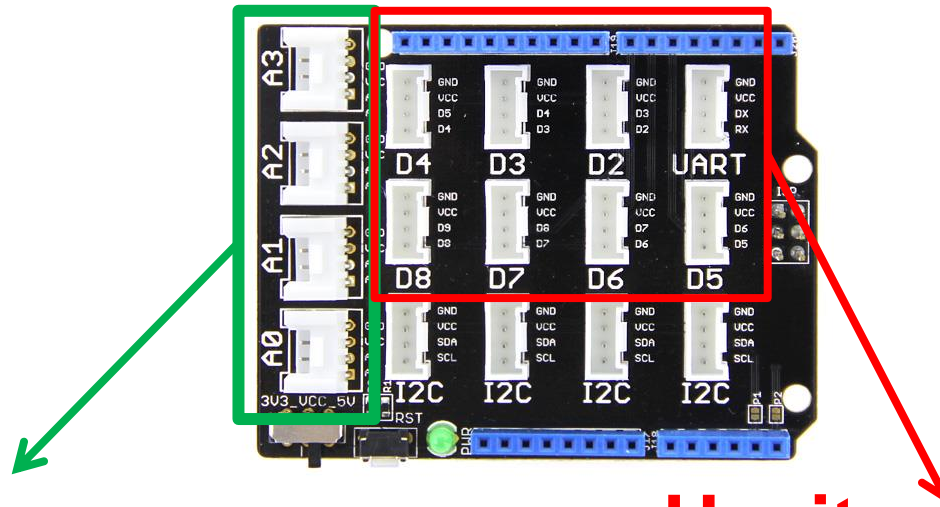
# Grove kit



Il kit Grove utilizza uno «shield» che, installato sui pin di Arduino, permette il collegamento diretto con trasduttori compatibili, senza bisogno di conoscenze di elettronica per creare sistemi di controllo anche complessi.



# Pin utilizzabili con Grove



- **Ingressi**

- A0 → A3: sono SOLO input ANALOGICI!
- D2 → D8: sono SOLO input DIGITALI!

- **Uscite**

- D2 → D8: TUTTE uscite DIGITALI
- D3, D5 e D6: sono anche uscite ANALOGICHE

- **I2C**: ingressi/uscite complesse, non utilizzabili nativamente con Snap4Arduino (per ora)

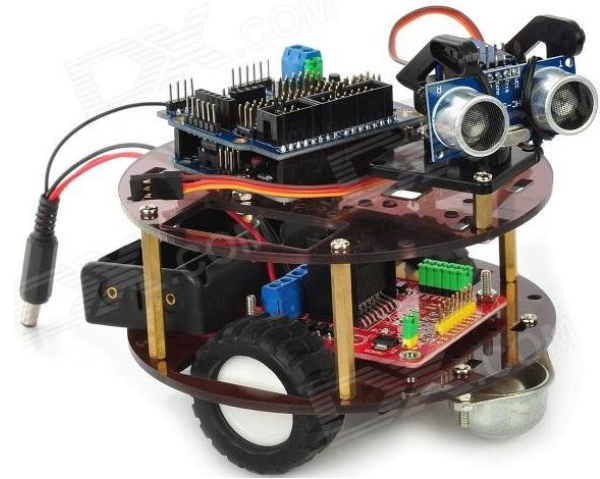




# Modi di utilizzo di Arduino

- Arduino può essere utilizzato principalmente in due modi:
  1. Come micro-controllore indipendente: Una volta caricato il firmware, Arduino può essere scollegato dal PC e alimentato a batteria o dalla rete elettrica.

Arduino svolgerà il suo lavoro come dispositivo autonomo, eseguendo un ciclo infinito per il pilotaggio continuo dei suoi trasduttori.



# Modi di utilizzo di Arduino

- Arduino può essere utilizzato principalmente in due modi:
  1. Come dispositivo per PC. In questa configurazione, Arduino si offre come «ponte» per l'uso dei trasduttori da parte del PC.

La logica del programma è eseguita dal PC, mentre Arduino offre il supporto per l'accesso ai trasduttori



# Snap!4Arduino

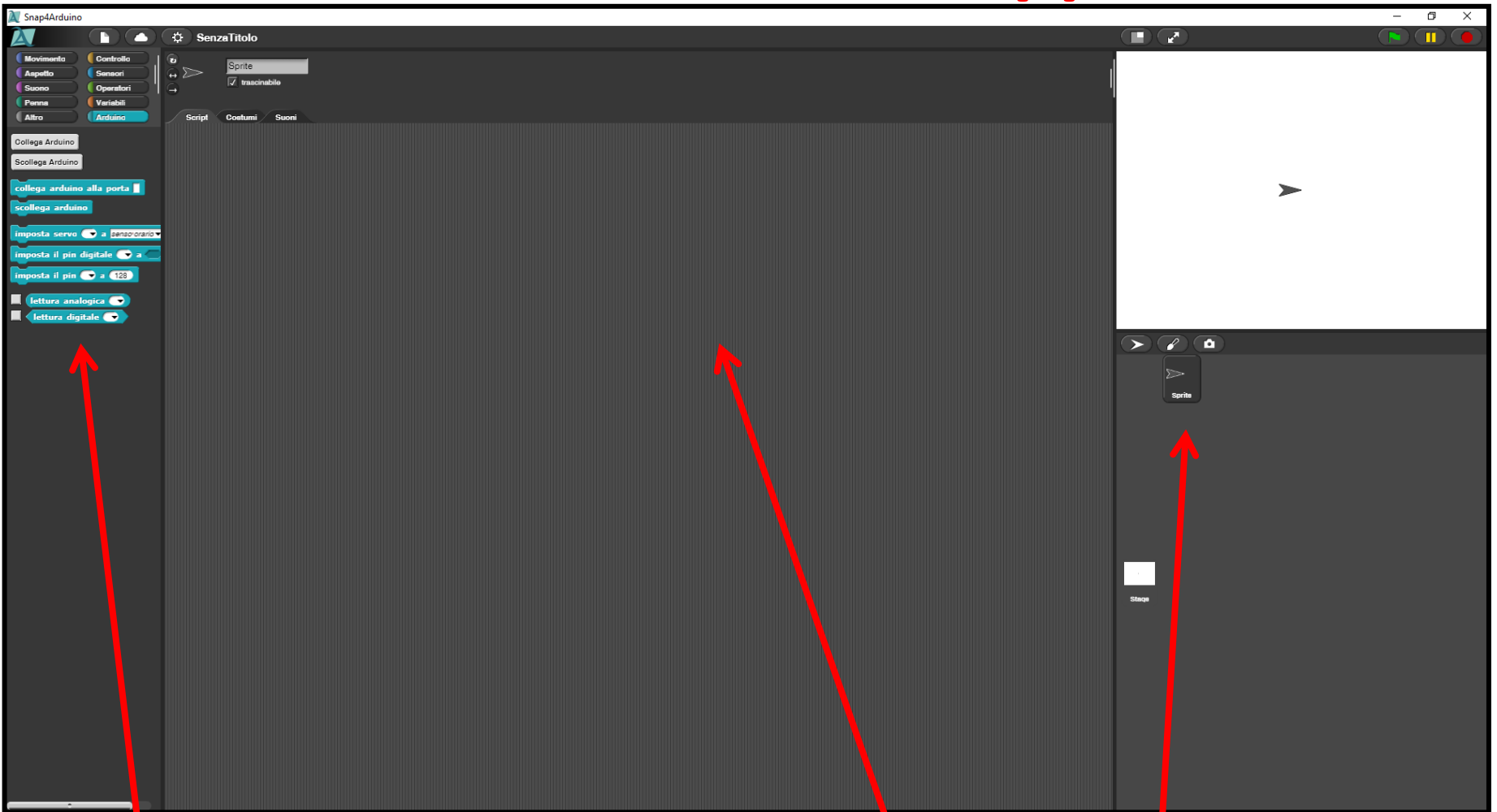
<http://snap4arduino.rocks/>

- Ambiente di sviluppo implementato dall'Edutec research group from Citilab (Cornellà, Barcelona),
- Incorpora il linguaggio Snap! Sviluppato dall'Università di Berkley, estendendola con blocchi specifici per la gestione dell'I/O da e verso i trasduttori collegati





# Ambiente di sviluppo



Nuovi blocchi per gestione dei sensori (sezione «Arduino»)

Area degli script

Area degli Sprites



# Ambiente di sviluppo

- Grazie alla nuova sezione «Arduino», UNO sprite (ad eccezione dello Stage), può essere collegato ai sensori e agli attuatori.
  - SENSORI: la lettura dei valori in input può essere «analogica», oppure «digitale»:
    - **Lettura «analogica» (A0-A3):** il sensore restituirà un valore compreso tra 0 (minimo) e 1023 (massimo)
    - **Lettura «digitale» (D2-D8):** il sensore restituirà un valore TRUE (input attivato) o FALSE (input disattivato)
  - Attuatori: si può attivare un'azione in due modi:
    - **Attivazione «analogica» (D3,D5 e D6):** l'attuatore può essere spento (valore 0); acceso alla massima potenza (valore 255) o acceso gradualmente a qualsiasi valore intermedio.
    - **Attivazione «digitale» (D2-D8):** l'attuatore può essere solo spento (valore «false») o acceso alla max potenza (valore «true»)



# Connessione di Arduino

- Prima di poter usare i sensori di Arduino, la nostra board deve essere «collegata» all'ambiente di sviluppo.
- Solo dopo il collegamento avremo accesso a tutti i sensori e attuatori.
- Al termine del lavoro (opzionale), o se dovessimo cambiare la board attualmente inserita, possiamo scollegarla con il pulsante «scollega»
- Collegamento e scollegamento possono essere fatte «al volo» da un programma in esecuzione, per «switchare» tra due o più board collegate contemporaneamente



# Blocchi per i sensori

- Questi blocchi permettono al nostro programma Snap! di accedere alle letture dei sensori e alle scritture (comandi) agli attuatori.
- Lettura digitale: Lettura del sensore digitale collegato al pin indicato.
- Lettura analogica: Lettura del sensore analogico collegato al pin indicato.
- Scrittura digitale: attiva o disattiva l'attuatore indicato con impostazione «true» o «false».
- Scrittura analogica: attiva o disattiva l'attuatore indicato con intensità da 0 a 255.
- Controllo servo-motori: imposta direzione e velocità di rotazione di un servo-motore.



# Arduino: Videogame finale GALAGA

Giorgio Leonardi, Paola Giannini

<https://orienta.dir.uniupo.it/course/view.php?id=43>

oppure: <https://orienta.dir.uniupo.it> → informatica → Esperienza Formativa di  
Orientamento per le Scuole Superiori 2023/24: Arduino

# Videogame da costruire:

- Una versione di «Galaga», in cui però la nostra navicella si muoversi in orizzontale, e
- I «nemici» alieni potranno muoversi nello spazio, cercando di spararci o di venirci addosso
- Eventualmente, possiamo disseminare lo «spazio» di meteoriti, che non dobbiamo urtare (i meteoriti possono stare fermi, o se volete potete farli viaggiare nello spazio, ma NON sparano)
- Il nostro scopo è eliminare tutti gli alieni, sparando loro, nel frattempo evitando i meteoriti



# Il gioco originale

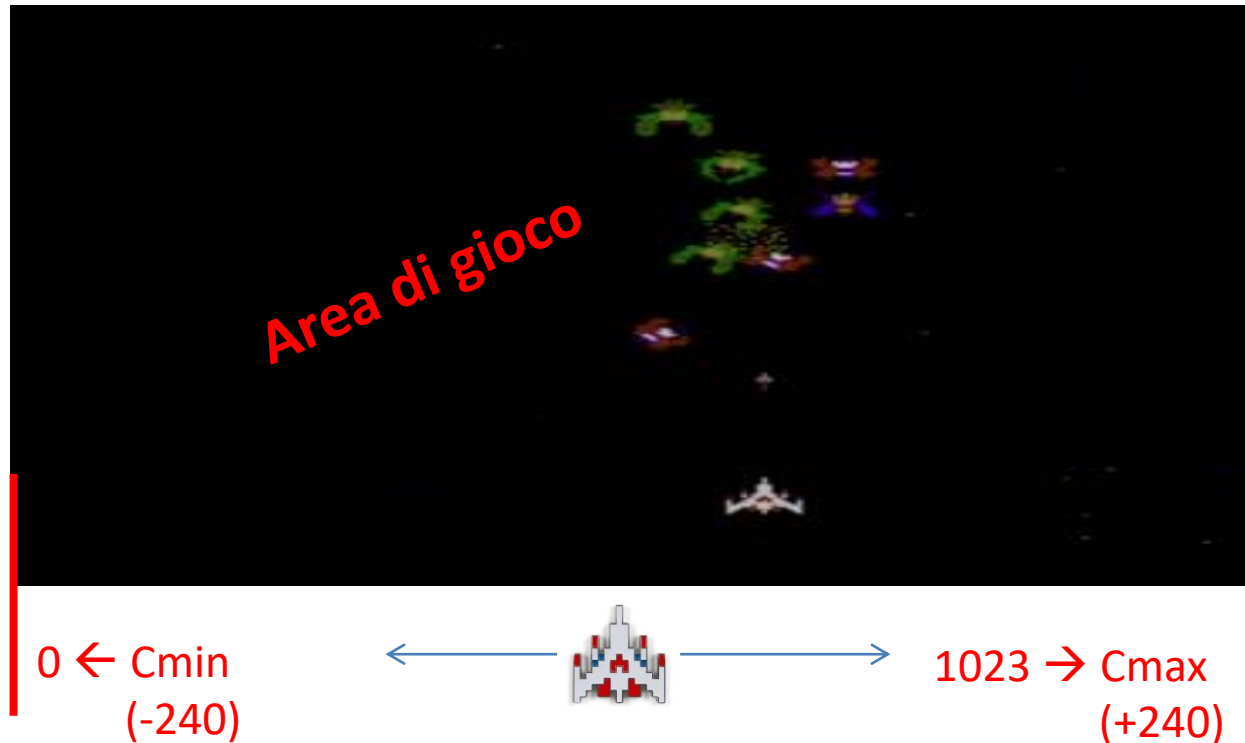


# Passi per costruire il gioco



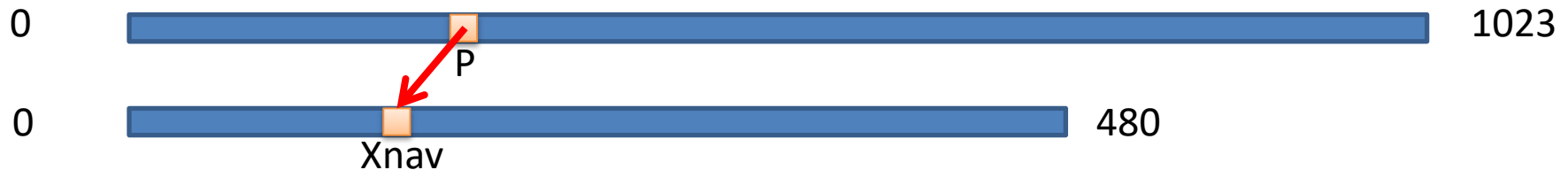
# Passo 1: la nostra navicella

- La nostra navicella si muove in orizzontale nell'area di gioco, guidata da un potenziometro:



# Passo 1: Posizioniamo la navicella

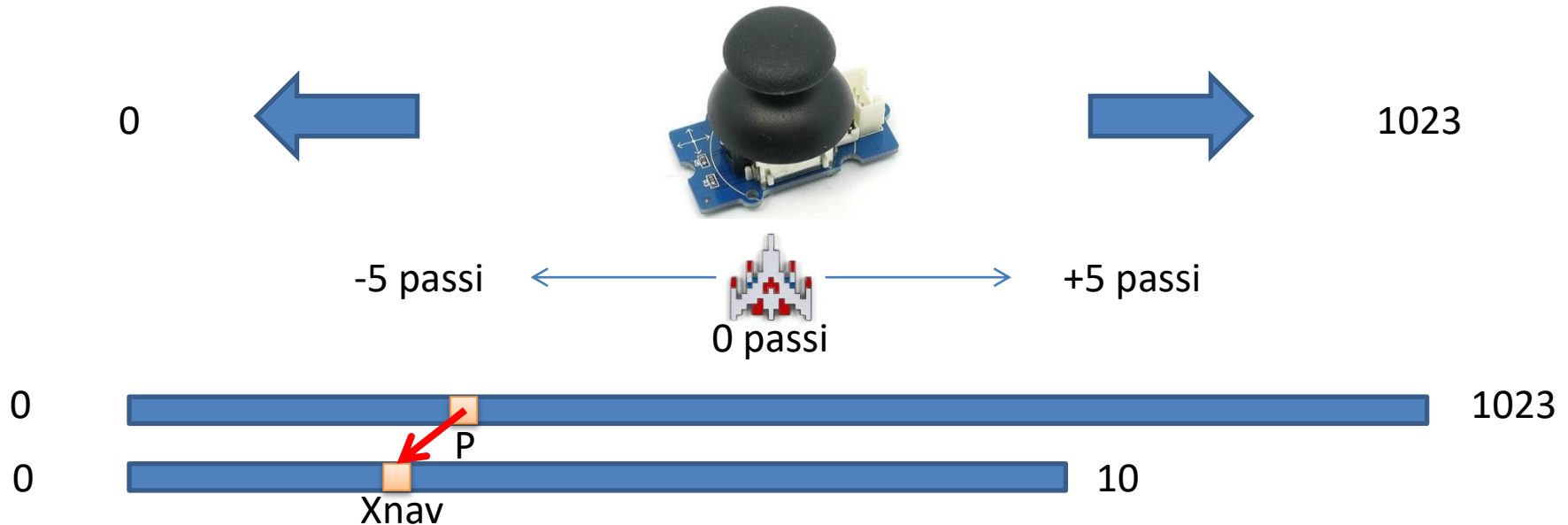
- Scrivete la proporzione che mappa la posizione della nave XNAV, facendo in modo che P=0 mandi la nave in posizione Cmin (-240) e P=1023 mandi la nave in Cmax (+240)



- Inserite questa formula in un nuovo blocco **calcola\_posizione**  
 **$X_{nav} = \text{arrotonda} ((P/1023) * 480) - 240$**
- Algoritmo della navicella:
  - Vai dove Y è -160
  - Per sempre:
    - Leggo da Arduino il valore del potenziometro P
    - XNAV = **calcola\_posizione** (P)
    - Vai dove X è XNAV



# Opzione 2: Joystick



- Inserite questa formula in un nuovo blocco calcola\_movimento

$$\text{Passinav} = \text{arrotonda}(((P / 512) * 10) - 5)$$

- Algoritmo della navicella:
  - Vai dove Y è -160
  - Punta in direzione 90 (posizione orizzontale)
  - Per sempre:
    - Leggo da Arduino il valore del potenziometro P
    - $\text{Passinav} = \text{calcola\_movimento}(P)$
    - Fai «Passinav» passi



## Passo 2: il nostro «fuoco»

- La nostra navicella spara dei proiettili, ma ne spara uno per volta. Per cui, dobbiamo avere uno sprite che rappresenta il nostro proiettile.
- Per sparare, abbiamo un pulsante di «fuoco», e possiamo agire così: ogni volta in cui il pulsante è premuto, mando un messaggio «FUOCO» allo sprite del proiettile. Al ricevimento del messaggio, lo sprite del proiettile:
  - Si posiziona davanti al muso della navicella
  - Punta in alto e si «mostra»
  - Avanza continuamente di un certo numero di passi (fisso), e sparisce (si nasconde) quando:
    - Arriva al bordo dello schermo
    - Colpisce un alieno (quando ci saranno)
    - Colpisce un meteorite (quando ci saranno)



# Passo 2: ATTENZIONE!

- Non posso sparare un nuovo proiettile, finché il precedente non ha terminato la sua corsa!
- usiamo quello che in Informatica è definito un “Semaforo” (o “flag”), per disabilitare il pulsante di sparo finché il proiettile non ha raggiunto la fine della sua corsa.
  - Uso una variabile «sparo»: se  $\text{sparo} = 1$ , allora posso sparare. Se è a 0, allora lo sparo è disabilitato
  - A inizio partita, la variabile «sparo» deve essere posta a 1
  - Quando dalla navicella premo il pulsante, allora invio un messaggio «SPARA» SE E SOLO SE «sparo = 1»,
- Lo sprite del proiettile, al ricevimento del messaggio «SPARA»:
  - $\text{sparo} = 0$  (non posso sparare un altro proiettile nel frattempo)
  - Si posiziona davanti al muso della navicella
  - Punta in alto
  - Avanza continuamente di un certo numero di passi (fisso), e sparisce quando:
    - Arriva al bordo dello schermo
    - Colpisce un alieno (quando ci saranno)
    - Colpisce un meteorite (quando ci saranno)
  - $\text{sparo} = 1$  (da ora in poi si può sparare un nuovo proiettile)



# Passo 3: i meteoriti

- I meteoriti sono piccoli massi, contro cui possiamo **morire** se andiamo a **sbattervi** contro
- I meteoriti sono **sprites**, a forma di masso, che **partono** da una posizione **casuale** e si **muovono continuamente**, rimbalzando contro i bordi dello schermo
- I meteoriti **possono ucciderci**, e lo fanno quando vi sbattiamo contro. Usiamo quindi il concetto di «collisione» offerto da Snap.
- Quando un meteorite **colpisce** la nostra **navicella**, manda un **messaggio** (chiamato, ad esempio, «**colpito**») allo sprite della navicella stessa.
- Quando la **navicella riceve** un messaggio di nome «**colpito**», **esplode** (cambiando costume) e si perde una vita.
- Fatto un meteorite, possiamo copiarne lo sprite e aggiungere quanti meteoriti vogliamo!



## Passo 4: gestione del punteggio e delle vite

- Il **punteggio** e le **vite** sono due **variabili** accessibili a tutti gli sprites.
- Alla partenza del gioco, il numero di vite è inizializzato a 3 (o 5, o 7, vedete voi) e il punteggio è posto a 0
- Ogni volta in cui lo sprite navicella riceve un messaggio «colpito»:
  - Esplode cambiando costume
  - Decrementa il numero di vite
  - Se il numero di vite arriva a zero, allora termina il gioco con una schermata di «game over», altrimenti:
    - Aspetta un po' di tempo (magari un secondo o giù di lì)
    - Ritorna navicella e si continua a giocare



# Passo 4: ATTENZIONE!

- Mentre la nave si trova in stato di «esplosione», bisogna impedire che venga nuovamente colpita e che perda una vita inutilmente
- Utilizzo un altro «semaforo», chiamato `in_gioco`:
  - Se `in_gioco` vale 1, allora la nave è attiva e sta giocando (all’inizio della partita, `in_gioco` dev’essere posta a 1)
  - Se `in_gioco` vale 0, allora non può essere colpita
  - **Quindi, invio il messaggio «colpito» solo se «`in_gioco` = 1»**
- Ogni volta in cui lo sprite navicella riceve un messaggio «colpito»:
  - **Porto `in_gioco` = 0 («proteggo» la mia navicella mentre è in esplosione)**
  - Esplode cambiando costume
  - Decrementa il numero di vite
  - Se il numero di vite arriva a zero, allora termina il gioco con una schermata di «game over», altrimenti:
    - Aspetta un po’ di tempo (magari un secondo o giù di lì)
    - Ritorna navicella
    - **Porto `in_gioco` = 1 (e quindi si continua a giocare)**





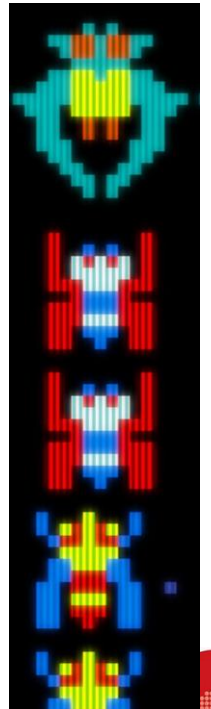
# Schermata di game over

- Per terminare il gioco, creiamo uno **sprite** di «**Game Over**», grande quanto una scritta al centro dello schermo, oppure grande quanto tutto lo schermo (magari con un bel background da eroi nello spazio).
- Questo sprite è **nascosto all'inizio** del gioco. Quando il gioco finisce (le **vite** arrivano a **0**), allora:
  - **Mostriamo** lo sprite di **Game Over**
  - **Fermiamo** tutti gli **script** (interrompendo il gioco)



# Passo 5: finalmente gli alieni!

- I **nemici** sono esseri **alieni** o navicelle nemiche, che **partono** in **formazione**, ci **sparano** (ogni tanto) e cercano di **venirci addosso** uscendo dalla formazione.
- Un nemico **deve sparare**, quindi creiamo uno **sprite** per il **nemico** e uno sprite per il suo **proiettile** (magari con forma e colore diversi da quelli del nostro proiettile!)



## Passo 5: sprite del nemico

- Iniziamo creando un **nemico** «**stupido**», cioè che si **muove** a **destra** e a **sinistra**.
- L'alieno si muove sull'asse orizzontale a destra, poi a sx, poi a dx, ... Creare il movimento in uno script a ciclo infinito che parte all'avvio del gioco
  - A inizio gioco, sceglie una posizione iniziale e punta verso destra (o a sinistra). Poi, per sempre:
    - Fai 20 passi (aumentate o diminuite per cambiare la velocità)
    - Rimbalza contro il bordo



# Passo 6: l'alieno esplode

- L'alieno **esplode** quando viene **colpito** dal **nostro proiettile**:
  - se **l'alieno** entra in **collisione** con il **proiettile**, fa una **piccola animazione** (ad esempio: aumenta dimensione 20%, diminuisce dimensione del 20%, aumenta e poi diminuisce, oppure create un costume “esplosione” anche per l'alieno)
  - Poi, devono **sparire** sia **l'alieno**, sia il **proiettile**



# Passo 6: l'alieno esplode

- Gestisco questi eventi tramite messaggi:

**Alieno:** mentre mi sto muovendo:

- Se sto toccando il proiettile:
  - Mando un messaggio «alieno\_colpito»
  - Eseguo l'animazione per l'esplosione
  - Incremento il punteggio del gioco
  - Mi nascondo

**Proiettile:** quando ricevo il messaggio «alieno\_colpito»:

- Imposto il semaforo «sparo» a 1
- Mi nascondo



# Passo 7: l'alieno ci spara!

- L'alieno **spara**, e lo fa “quando gli pare” (cioè dopo un **tempo “random”**):
  - creare un **nuovo sprite** “Alien shoot”, che è lo sparo che parte da un alieno. Quando il gioco parte, questo sprite deve essere **non visibile**.
  - Aggiungere una **variabile** intera «**tempoSparo**» nello sprite Alieno (questa variabile deve essere “locale”, non accessibile a tutti perché è usata SOLO dal particolare alieno che ci stiamo costruendo)
  - A inizio gioco, a «tempoSparo» assegnamo un **valore random**, che simula il **tempo** di “**inattività**” tra uno sparo e il successivo



# Passo 7: l'alieno ci spara!

- Nel ciclo infinito del movimento dell'alieno, far **decrementare** questa **variabile** appena prima della fine di ogni ciclo (come ultima operazione del ciclo, ad esempio) e, quando **arriva** a **0**, faccio partire le procedure per **sparare**:

## Nello sprite "Alieno":

- Mentre mi sto muovendo:
  - Decremento «tempoSparo»
  - Se «tempoSparo» < 0:
    - Porto Xsparo alla mia posizione X
    - Porto Ysparo alla mia posizione Y
    - Crea un clone di «Alien shoot»
    - Assegno a «tempoSparo» un nuovo numero random

## Nello sprite "Alien shoot":

- Quando vengo clonato:
  - Mi posiziono nella posizione Xsparo, Ysparo
  - Mi rendo visibile
  - Scendo in verticale fino al fondo dello schermo (opzionalmente, l'alieno potrebbe sparare nella nostra direzione, invece che in verticale)
  - Elimino questo clone



# Passo 7: l'alieno ci fa del male..

- Quando la uno sparo alieno colpisce la nostra navicella, quest'ultima deve esplodere e perdere una vita
- **Quando** lo sprite «**Alien shoot**» **colpisce** la nostra **nave**, allora **invia** un messaggio «**colpito**» e poi si **nasconde**
- La nostra navicella contiene già il codice per esplodere e perdere una vita, quando riceve un messaggio «colpito»





# Passo 8: replico i nemici

- Fatto un nemico e il suo proiettile, possiamo **replicare** i **nemici** per **popolare** il **gioco** di tanti avversari.
- All'inizio del gioco, **ogni nemico** dovrà prendere una **posizione** iniziale **fissata** da voi, in modo da apparire in uno «**schieramento**» iniziale che vi soddisfi.



# VITTORIA!

Vinciamo quando uccidiamo tutti i nemici. Per cui, il gioco deve **memorizzarsi quanti nemici rimangono** ancora **da uccidere** e, se questo numero va a **0**, fate apparire un bellissimo **sprite** che **celebri la vostra vittoria contro il male!!!**



**Passi successivi: aggiungiamo  
imprevedibilità al gioco creando  
avversari dal comportamento  
«intelligente» e diverso da nemico a  
nemico!**

# Nemici dal carattere diverso

- Aggiungiamo un po' di carattere al nostro nemico: facciamogli decidere se vuole sparare oppure no. Applichiamo la prima tecnica di intelligenza artificiale

## DECISION THRESHOLD

- Se il mio nemico avesse voglia di sparare una volta su 1000, allora basterebbe contare fino a 999, poi quando arrivo a 1000 sparo. Ma vorrei che il mio nemico fosse più «aleatorio»



# Quando sparare

- Per aggiungere un comportamento meno prevedibile, uso una soglia e i numeri random, da attivare ogni volta in cui è arrivato il momento di sparare.



- Costruito questo schemino, estraggo un numero random tra 1 e 10.000. Se il numero estratto è  $\leq$  di 9.000, vuol dire che non ho voglia di sparare e non lo faccio; se il numero è  $>$  di 9.000, allora ho voglia di sparare e sparo!!!
- Questa tecnica mi dà il controllo sul comportamento del mio nemico: se ne creassi 10, potrei crearne qualcuno più «cattivo» e qualcuno meno «cattivo», semplicemente variando la soglia di sparo.



# Nemici duri a morire

- Non tutti gli alieni devono per forza morire al primo colpo. Alcuni muoiono dopo essere stati colpiti 2 volte (o più)
- Ogni alieno ha una variabile locale «vite\_aliene», che parte dal numero di vite che ha l'alieno prima di esplodere
- Quando l'alieno è colpito dal nostro proiettile, decrementa il suo numero di «vite\_aliene».
  - Se «vite\_aliene» diventa 0, allora esplode e sparisce, incrementando il nostro punteggio
  - Altrimenti, il nostro alieno potrebbe cambiare forma, per capire che è stato colpito (magari gli cambiamo colore)



# Difendiamoci, finchè si può!

- Dotiamoci di uno «scudo spaziale»!
- Premendo il pulsante touch (o un secondo pulsante), possiamo mandare un messaggio a uno sprite che ci circonda e ci segue per 5 secondi.
- Per questi 5 secondi, nessuno può ucciderci (non reagiamo al messaggio «colpito», possiamo farlo disabilitando il semaforo «in\_gioco»).
- Terminati questi 5 secondi, lo scudo se ne va e possiamo morire di nuovo (riattiviamo il semaforo «in\_gioco»).
- Abbiamo a disposizione solo 3 scudi spaziali, quindi usate una variabile che conti quanti scudi avete usato e non lo attivi più se li avete già usati tutti!!!



# Kamikaze!!!

- Allo scadere di un intervallo di tempo “random”, l’alieno decide di scendere per intercettarci.
- Usiamo la stessa tecnica dello sparo: usiamo un nuovo contatore «banzai», inizializzato in maniera random, per decidere quando è il momento di lanciarsi
- Quando il contatore «banzai» arriva a 0, l’alieno si lancia, seguendo una certa traiettoria
- L’alieno, se colpisce la nostra navicella, invia un messaggio «colpito» e ci fa esplodere (e perdere una vita)
- Quando l’alieno raggiunge il fondo dello schermo, torna nella sua posizione iniziale e assegna a «banzai» un nuovo numero random, per capire quando lanciarsi la prossima volta



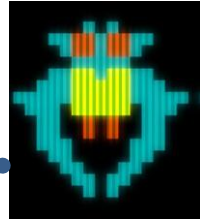


# Movimenti possibili

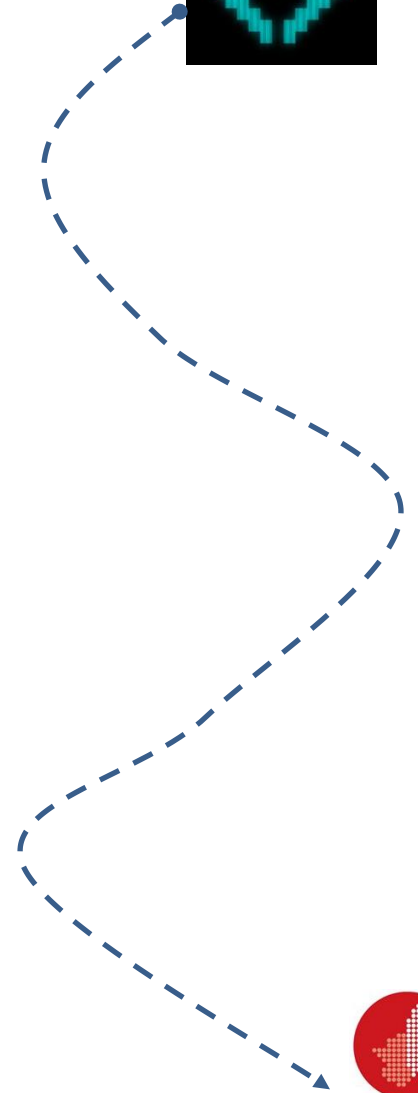
- Quando decide di lanciarsi, l'alieno potrebbe muoversi in modo diverso. Potrebbe, ad esempio:
  - Scendere dritto in verticale
  - Prendere la direzione della nostra astronave e scendere giù in diagonale
  - Muoversi a zig-zag
  - Movimento «sinusoidale»



# Movimento sinusoidale



- E' un movimento sinusoidale, con l'asse attorno alla posizione orizzontale dell'alieno stesso.
- Ad esempio, memorizzo la posizione corrente dell'alieno in due variabili locali (ad esempio  $X_{discesa}$  e  $Y_{discesa}$ ), poi parte un ciclo in cui: decremento la posizione verticale  $Y_{discesa}$  e pongo la posizione corrente  $X$  a:  $X_{discesa} + \cos(Y_{discesa})$ , finchè l'alieno non arriva in fondo allo schermo



# Livello di difficoltà crescente

- Aggiungiamo un livello incrementale di difficoltà: definiamo una nuova variabile globale (ad esempio “difficolta”), impostata a 1 (livello facile) all’inizio del gioco.
- Uso una variabile “contatore\_difficolta” inizializzata (ad esempio) a 1000, e la decremento alla fine del ciclo “Per sempre” della nostra navicella.
- Quando “contatore\_difficolta” arriva a 0: controllo se “difficolta” < 3, incremento la difficoltà e pongo nuovamente “contatore\_difficolta” a 1000. (questo meccanismo mi permette di simulare un “timer” customizzato).
- L’alieno si comporta compatibilmente con difficoltà: la lunghezza del “passo di marcia” dell’alieno deve aumentare in proporzione alla difficoltà (proporzionalità diretta), mentre il tempo random per decidere quando sparare diminuisce in proporzione alla difficoltà (proporzionalità inversa).

