



# Robomaquia

Di Muro Gerardo - Di Pasquale Valerio - Troisi Vito

Corso di Fondamenti di Intelligenza Artificiale  
Università degli Studi di Salerno

Ottobre-Dicembre 2022

Repository di progetto:

<https://github.com/daqh/robomaquia>

*Words without **thoughts** never to heaven go.  
-The Tragedy of Hamlet, Prince of Denmark*

## Indice dei contenuti

<b>1</b>	<b>Introduzione</b>	<b>3</b>
<b>2</b>	<b>Analisi dell'ambiente</b>	<b>4</b>
2.1	Definizione del problema . . . . .	4
2.1.1	Scope e obiettivi . . . . .	4
2.1.2	Specifica PEAS . . . . .	4
2.1.3	Caratteristiche dell'ambiente . . . . .	6
2.2	Analisi preliminare . . . . .	7
<b>3</b>	<b>Soluzione proposta</b>	<b>9</b>
3.1	Overview . . . . .	9
3.2	Moduli previsti . . . . .	9
3.2.1	Laertes: Il modulo operativo . . . . .	10
3.2.2	Hamlet: Il modulo genetico . . . . .	11
3.2.3	Ophelia: Il modulo di apprendimento . . . . .	17
<b>4</b>	<b>Glossario</b>	<b>18</b>
	<b>Bibliografia</b>	<b>19</b>

## 1 Introduzione

Robomaquia (parola macedonia di *robot* e *tauromaquia*, letteralmente *La Corrida dei Robot*) si propone come un gioco desktop dalle semplici apparenze, ma dalla straordinaria rilevanza, se analizzato sotto la lente critica dei canoni di progettazione che hanno portato alla sua realizzazione.

Il giocatore immerso nell'ambiente di gioco si troverà ad affrontare orde di nemici sempre più numerosi ed abili, capaci di apprendere dalla propria esperienza e dalle azioni effettuate dal giocatore stesso. Ma come regolare l'evoluzione dei suddetti? Come gestire la loro capacità di apprendere? E come fare a descrivere una strategia di azione nemica che sia ampiamente giocabile ed al contempo in grado di offrire una sfida concreta per il giocatore?

La prima sezione riguarderà l'analisi del problema, inteso come l'ambiente di gioco, da un punto di vista strettamente operativo, seppur generico. Saranno dunque valutate le caratteristiche dell'ambiente oltre a quelle richieste dall'agente intelligente (i.e. come si vedrà dal modulo che ne descrive l'azione di gioco). La seconda sezione sarà invece dedicata alla progettazione ad alto livello e di dettaglio di ciascuno dei moduli previsti per l'agente intelligente, in termini di funzionalità, algoritmi ed euristiche impiegate. In coda al presente fascicolo di documentazione compare un glossario di termini particolarmente rilevanti nell'ambito del dominio applicativo considerato e in taluni casi, di quello delle soluzioni.

Comprendere ed applicare la miglior risposta alle questioni di cui sopra descrive solo una minima frazione delle sfide e dei problemi che gli scriventi si sono posti per realizzare un prodotto che fondesse le caratteristiche di un gioco e quelle di un interessante esperimento accademico sulla costruzione di un agente intelligente, e le cui soluzioni sono felici di mostrare nel presente documento.

## 2 Analisi dell'ambiente

Avventuriamoci dunque nella sezione dedicata all'analisi dell'ambiente. Si comincerà parlando dei *boundaries* ad esso associati e agli obiettivi di progettazione, per poi proseguire con la caratterizzazione prevista per l'agente intelligente che opererà nell'ambiente.

### 2.1 Definizione del problema

Non c'è problema che sia risolvibile senza una chiara, non ambigua definizione. Definire i canoni del problema aiuterà a metterne in luce i punti salienti e quelli trascurabili.

#### 2.1.1 Scope e obiettivi

Se volessimo riassumere in una frase l'obiettivo principale di Robomaquia potremmo affermare che esso consista nel garantire un ambiente giocabile in cui è possibile osservare, con l'avanzare dei livelli, l'evoluzione nemica verso una strategia di gioco più consistente.

Si prevede dunque di predisporre un ambiente costituito da una serie di *stanze* di gioco, ciascuna sequenzialmente visitabile sconfiggendo tutti i nemici nella stanza precedente. La visita delle stanze non prevede dunque bivi né percorsi alternativi. La condizione necessaria e sufficiente per giungere alla stanza  $i$  è dunque l'aver eliminato ogni nemico nella stanza  $i-1$ . Fa eccezione la prima stanza, acceduta automaticamente all'avvio di una nuova partita.

I nemici, dal canto loro, dispongono di alcuni parametri che descrivono il loro comportamento nell'ambiente, finalizzati generalmente all'abbattimento del giocatore. All'atto dell'accesso in ogni nuova stanza successiva alla prima, una nuova generazione di nemici verrà generata, sicché al giocatore sia proposta una nuova, più ardua sfida, anche coadiuvata dal fatto che tutti o parte dei parametri dei nemici sarà alterata dalla loro capacità di adattarsi al giocatore.

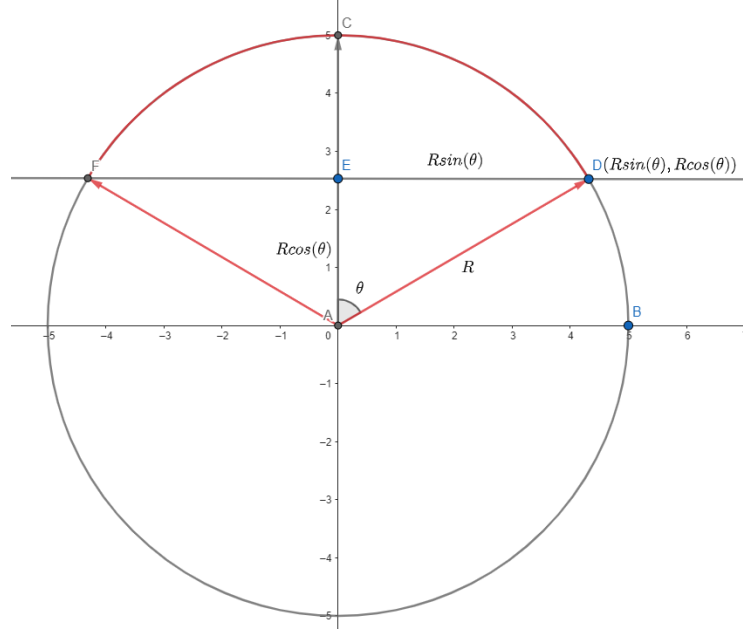
#### 2.1.2 Specifica PEAS

Segue la descrizione della specifica PEAS legata all'agente. Si osservi che, così come le caratteristiche dell'ambiente, la specifica è pesata sul modulo operativo dell'agente, i.e. il prototipo dell'individuo nemico che il giocatore affronterà.

- **Performance** - Le metriche di performance dell'agente riguarderanno parametri offensivi e difensivi. In particolare saranno considerati:
  - Il danno effettuato al giocatore.
  - La precisione dei colpi, definita da:  $P : s/n \Rightarrow 0 \leq P \leq 1$ , dove  $s$  è il numero di colpi andati a segno ed  $n$  è il numero di colpi totali tentati.
  - Il lifespan, i.e. la somma degli intervalli di tempo in cui l'agente è rimasto in vita, mentre si è trovato nel raggio d'azione del giocatore.
- **Environment** - Come già anticipato ai paragrafi precedenti, l'ambiente consta di una sequenza di stanze. In ciascuna stanza vi sono: il giocatore, le stanze dell'agente intelligente, ostacoli che limitano il campo di visibilità e di azione e che non sono attraversabili. L'ambiente presenta inoltre dei punti di generazione di nemici, i quali vengono generati in un'unica iterazione, ma rilasciati nell'ambiente ad intervalli regolari.
- **Actuators** - L'agente agisce in due modalità principali: muovendosi o attaccando. Conseguentemente gli attuatori (virtuali) sono costituiti dalle *armi* con cui l'agente attacca e da ciò che abilita il suo moto rotatorio e traslatorio..
- **Sensors** - L'agente osserva l'ambiente mediante due strutture: il *field of view* (o FOV, o campo visivo) ed il *field of attack* (o FOA, o campo di attacco). Ciascuno è costituito, come si vedrà tra le note sottostanti, da un ventaglio posto dinanzi all'agente che indica il sottoinsieme di punti dell'ambiente in cui l'agente è conscio della presenza del giocatore (il FOV) e quello in cui attacca il giocatore direttamente (il FOA). L'agente dispone inoltre di un supporto di trasmissione/ricezione, che può impiegare per inviare e ricevere messaggi da altri agenti circa la posizione del giocatore. Si assume che l'agente conosca in ogni momento la posizione dei suoi compagni.

Note a valle della specifica PEAS:

- Per quanto concerne i vari campi visivi dell'agente, si pone il problema del calcolo delle coordinate di un punto dato un raggio visivo  $R$  passante per l'origine (i.e. l'agente stesso) e l'angolo formato da tale raggio ed il segmento congiungente l'origine con tale punto. Come mostra la figura sottostante:



Le coordinate di tale punto sono ottenibili ragionando mediante vettori ed impiegando le identità trigonometriche fondamentali. Si osservi inoltre che:

- Se il corpo trattato (l'agente) non si trova nell'origine è sufficiente scartare le coordinate del punto  $D$  sommando ad esse il vettore di componenti  $(x, y)$  dove  $x$  e  $y$  rappresentano le generiche coordinate dell'agente.
- Dal momento che l'agente è considerabile come un corpo esteso, piuttosto che come un punto materiale, è necessario sommare all'angolo  $\theta$  l'angolo di rotazione dell'agente rispetto alla verticale.

Ciò consente persino di generare il campo visivo come l'approssimazione dell'arco di circonferenza sotteso dal vettore  $\overrightarrow{AD}$  e dal vettore ad esso simmetrico rispetto alla verticale (un settore circolare dirimpetto all'agente).

L'approssimazione in questione è ottenibile mediante la generazione di una lista di punti tra i due vettori simmetrici.

### 2.1.3 Caratteristiche dell'ambiente

L'ambiente, inteso in tale sezione come la singola stanza, che è ciò che l'agente sperimenta operativamente nel corso della sua esistenza, presenta la seguente tassonomia:

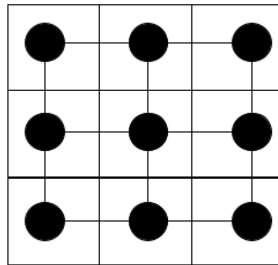
- **Parzialmente osservabile:** come analizzato in precedenza, il FOV dell'agente è limitato ad un dato raggio. Conseguentemente, l'ambiente non è totalmente osservabile.
- **Stocastico:** le azioni di movimento e attacco dell'agente non conducono a risultati certi, per via della libertà di azione del giocatore, che agisce in contemporanea all'agente.
- **Sequenziale:** l'agente agisce mediante una sequenza di azioni non atomiche.

- **Dinamico:** ancora una volta, per via della presenza del giocatore e di ulteriori istanze dell'agente, l'ambiente varia nel tempo.
- **Discreto:** le percezioni dell'agente sono limitate e numerabili.
- **Multi-agente cooperativo:** come ribadito, coesistono nell'ambiente più istanze dell'agente in questione. L'obiettivo di ciascuna è sconfiggere il giocatore.

## 2.2 Analisi preliminare

La formulazione rigorosa del problema di ricerca risolto dalla componente operativa dell'agente intelligente, è immediatamente fornibile. Tuttavia, prima di addentrarci in essa, è cruciale analizzare cosa caratterizza uno *stato* dell'ambiente che pone tale problema.

Si supponga di dividere la stanza di gioco in sezioni quadrate (d'ora in poi *quadranti*). Conseguentemente, l'esplorazione di una stanza, utile alla ricerca del giocatore nel caso in cui egli non rientri nel campo visivo dell'agente, si traduce nell'esplorazione del seguente grafo:



In cui ciascun nodo rappresenta un quadrante e ogni arco descrive la relazione di adiacenza tra i due quadranti in una stanza. Abbiamo in tal modo descritto l'intera stanza mediante l'uso di un grafo, le cui tecniche di esplorazione sono certamente ben note.

Uno stato del problema di ricerca affrontato dal modulo operativo sarà pertanto costituito dalle seguenti informazioni:

- Il quadrante attuale.
- Lo *status della ricerca* (true se trovato, false altrimenti): la percezione che descrive se nel quadrante attuale l'agente vede il giocatore.
- La *segnalazione* (rappresentata dall'identificativo di un quadrante o *null*): la percezione che descrive se l'agente ha ricevuto un segnale da un'altro, il quale indica che il giocatore è stato trovato.

Alla luce di tali conclusioni, la definizione formale del problema si presenta come segue:

- **Stato iniziale:** l'agente è situato in un quadrante casuale. Il giocatore non è visto (ma può entrare immediatamente nel campo visivo dell'agente). Dal momento che ciò vale per ciascuna istanza dell'agente, egli non riceve alcuna segnalazione. Formalmente:
  - $\text{In}(\mathbf{q}, \text{false}, \text{null})$

- **Azioni:** l'agente può muoversi verso un quadrante o attaccare il giocatore se quest'ultimo è nel campo visivo dell'agente. La ricezione di una segnalazione da parte di un'altra istanza dell'agente, si traduce anch'essa in un movimento. Sia  $V$  l'insieme di nodi del grafo che descrive la stanza, le azioni sono descritte da:
  - **Move**( $q$ ),  $\forall q \in V$
  - **Attack**( $d$ ),  $\iff State(q, true, null) = true$
- **Modello di transizione:** il movimento dell'agente verso un quadrante causa il passaggio per i quadranti intermedi che sono situati sul cammino tra il quadrante iniziale e quello finale. Il risultato sarà, in ultimo, il raggiungimento del quadrante finale, a meno che il giocatore non venga trovato prima. Siano  $q_1, \dots, q_k$  i quadranti attraversati:
  - **Result**(**state**, **Move**( $n$ )) =  $State(n, false, null)$ ,  $\iff, \forall q \in q_1, \dots, q_k$ ,  $SearchStatus(q) = false$
  - **Result**(**state**, **Move**( $n$ )) =  $State(\bar{q}, true, null)$ ,  $\exists \bar{q} \in q_1, \dots, q_k : SearchStatus(\bar{q}) = true$
  - **Result**(**state**, **Attack**( $d$ )) =  $State(CurrentNode, true, null)$
- **Test obiettivo:** il giocatore è stato trovato.
  - ( $q, s, null$ ) è un obiettivo  $\iff s = true$
- **Costo di cammino:** è dato dal numero di quadranti tra il quadrante attuale e quello da raggiungere. Si osserverà in seguito, come il costo di cammino sarà combinato con delle euristiche per ottimizzare la ricerca del giocatore nella stanza. Dunque, se  $q_1$  è il quadrante attuale e  $q_2$  la destinazione:
  - $C(q_2) = quad(q_1, q_2)$

Una nota, a valle della definizione del problema: perché la vita del giocatore non è parte della sua definizione, per quanto una possibile azione dell'agente sia l'attacco?

Il punto è che l'agente attacca il giocatore (o quantomeno lo ingaggia in combattimento) non appena lo vede. Per quanto concettualmente, l'obiettivo degli agenti sia sconfiggere il giocatore, ciò non costituisce il problema modellato ai punti precedenti. Il problema effettivo sta nella caccia al giocatore nella stanza.

Le azioni di attacco e inseguimento saranno (visivamente) simultanee, oltre che sempre eseguite. Come possibile espansione futura del progetto, si osservi come l'azione **Attack**( $d$ ) possa a tutti gli effetti descrivere un problema di ricerca in sé, che definisce come il modulo operativo attacca il giocatore una volta individuato. Questo sotto-modulo potrebbe essere istruito dal modulo di apprendimento ad essere più o meno conservativo, ma è forse più simile ad un agente reattivo semplice, nella sua forma elementare.



### 3 Soluzione proposta

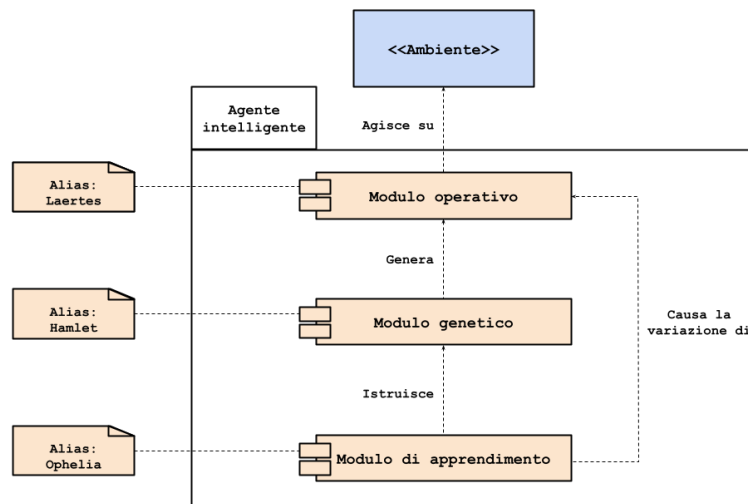
Passiamo dunque ad analizzare le scelte di progettazione effettuate circa la costruzione dell'agente intelligente.

#### 3.1 Overview

In questa sezione saranno analizzati nel dettaglio ciascuno dei moduli che, assemblati, andranno a costituire il *vero* agente oggetto della presente documentazione. Saranno dunque affrontate le scelte algoritmiche ed euristiche legate al modulo operativo dell'agente (colui il quale opererà concretamente nell'ambiente ed affronterà il giocatore), opportunamente descritto nelle fasi precedenti.

Sarà inoltre trattata l'integrazione del modulo genetico, che mediante un Genetic Algorithm consentirà di affrontare, ad ogni nuova stanza, una generazione più abile di nemici. Infine si osserverà la presenza di un modulo di apprendimento che istruirà i moduli precedenti analizzando il comportamento del giocatore, abilitando l'agente a combattere in modo più efficiente.

Si osservi che i moduli proposti possono essere schematizzati nel seguente diagramma:




#### 3.2 Moduli previsti

Alla luce di quanto osservato in precedenza, cominciamo con l'analizzare i moduli di cui sopra.

### 3.2.1 Laertes: Il modulo operativo

*By indirections find directions out.*  
*Laertes - Act II Scene I*

Il modulo operativo, il cui problema di ricerca era immediatamente definibile in fase di *Analisi preliminare*, dovrà occuparsi della ricerca (propriamente detta) del giocatore all'interno della stanza. Una volta trovato, Laertes ingaggerà il giocatore in combattimento. Prima di procedere col semplice algoritmo implementato da tale modulo in virtù della rappresentazione atomica di cui ai punti precedenti, analizziamo innanzitutto un'euristica interessante.

Nelle sezioni seguenti, saranno descritte alcune euristiche le quali hanno a che vedere con il concetto di *Swarm intelligence* (tipicamente tradotto per estensione con Mente a Scia-me). Ciò indica l'esistenza di algoritmi che hanno lo scopo di emulare il comportamento che avrebbero le singole istanze dell'agente se esse ragionassero come parte di un'entità più grande. Naturalmente la costruzione di algoritmi per *Swarm intelligence* va oltre lo scopo del progetto, e tuttavia alcune tra le euristiche che saranno presentate (marcate con l'icona ) vorrebbero presentare un accenno di tali idee. Si osservi che a prescindere dalla loro definizione categorica, tali euristiche saranno in ogni caso estremamente utili per la progettazione dell'agente intelligente.

Concetti di carattere generale e di forte ispirazione per le suddette euristiche (e ulteriori dettagli implementativi legati ad esse) sono discusse nell'articolo *Swarm Intelligence: Concepts, Models and Applications* [1] di H. Ahmed e J. Glasgow.

**Euristica** : posizione stimata del giocatore

L'euristica presa in considerazione è piuttosto semplice nella sua definizione. Considerando un'istanza di Laertes, se quest'ultimo non ha ricevuto una *segnalazione* allora è *probabile* che questi non si trovi nei quadranti in cui sono presenti ulteriori istanze di Laertes.

#### Ricerca best-first con euristica

L'algoritmo prescelto per risolvere il problema di ricerca proposto è un tipo di ricerca best-first con euristica, simile nella struttura al ben noto algoritmo A\*. La funzione di valutazione  $f(n)$ ,  $\forall n$  nodo (i.e. un quadrante della stanza) sarà definita come segue:

$$f(n) = g(n) + ch(n)$$

Dove  $g(n)$  rappresenta il costo di cammino precedentemente descritto, e  $h(n)$  la funzione euristica, a sua volta definita come:

$$h(n) = \text{distanza in quadranti dall'agente più vicino}$$

Il fattore  $c$  che pesa l'euristica è direttamente correlato al numero di agenti nella stanza: intuitivamente infatti, più popolata è la stanza, più incisiva diventa l'euristica (e meno determinante è invece il costo di cammino).

Le distanze in quadranti sono calcolate come segue. Siano  $(x_1, y_1)$  le coordinate del quadrante di partenza e  $(x_2, y_2)$  le coordinate di quello di arrivo, tale distanza in vale:

$$\text{dist}(q_1, q_2) = |x_1 - x_2| + |y_1 - y_2|$$

Una nota di curiosità: è comodo, per convincersi della veridicità di tale valore di distanza, pensare alla stanza come ad una matrice di quadranti piuttosto che ad un grafo. La semantica della rappresentazione atomica non varia, ma si ottengono informazioni circa le coordinate, descritte dagli indici di posizione di ogni quadrante nella matrice. L'obiettivo dell'algoritmo sarà la massimizzazione di tale funzione  $f(n)$ , cosicché ad ogni quadrante e sulla base della posizione degli altri agenti, Laertes decida quale nodo visitare per massimizzare la *copertura* della stanza.

In tal senso, la strategia impiegata dall'agente risponde all'aleatorietà dei movimenti del giocatore, con il raggiungimento della massima copertura possibile della stanza di gioco, con una forma embrionale di anticipazione delle mosse dei suoi compagni. La divisione in quadranti non dipende da FOV e FOA ed è dunque condivisa tra ogni istanza di Laertes.

Un'idea simile, legata l'adattamento dell'algoritmo A\* ad ambienti stocastici e parzialmente osservabili, seppure sia chiaro, per un problema differente, è discusso in *Pathfinding in stochastic environments: learning vs planning* di A. Skrynnik et al [2], ed è stato di particolare ispirazione per l'algoritmo di ricerca best-first qui presentato.

[DEFINIRE COSTANTE C PER EURISTICA]

### 3.2.2 Hamlet: Il modulo genetico

*I must be cruel only to be kind; Thus bad begins, and worse remains behind.*  
*Hamlet - Act III Scene IV*

Il modulo genetico, il quale consente di generare ad ogni stanza, una nuova e più abile generazione di nemici, è descritto nella sezione presente. Naturalmente si procederà analizzando le caratteristiche che è necessario definire per incorniciare il problema all'interno della meta-euristica fornita dagli algoritmo genetico.

Si osservi innanzitutto che è necessario esplicitare la codifica degli individui e la funzione (d'ora in avanti il *fitness*) mediante la quale gli individui suddetti saranno valutati. Si procederà infine con la caratterizzazione dei vari parametri legati alla genetica, insieme con il razionale delle scelte (ed eventuali euristiche applicate) per ciascuna di esse.

#### Codifica degli individui

Ciascun individuo della popolazione sarà rappresentato dalla seguente stringa: I geni che concorrono alla codifica di ciascun individuo sono la velocità di movimento, l'arma equipaggiata dall'individuo e l'avatar ad esso associato, che ne determina anche inerzia, resistenza ai colpi e hitbox. Si osservi che ulteriori caratteristiche derivate, quali ad esempio, il danno per colpo, sono desumibili a partire dai geni sopraelencati.

Pertanto, detta  $v$  la velocità di movimento,  $A$  l'arma equipaggiata e  $P$  l'avatar, la codifica dell'individuo  $L$  è descritta dalla seguente rappresentazione vettoriale:

$$L = [v, A, P]$$

La quale è osservabile in semplice formato di stringa, concatenando i geni:

$$L \equiv v, A, P$$


### Funzione di fitness

Si indichi con  $f$  la funzione di fitness. Sia  $r$  un qualunque individuo della popolazione all' $i$ -esima generazione  $P_i$ , allora  $f$  è esplicitata come segue:

$$f(r) = 2d + p + l$$

Dove  $d$  è il danno,  $p$  la precisione e  $l$  il lifespan, così come definiti nella specifica PEAS alla sezione 2 del presente documento. Naturalmente tali valori potrebbero essere soggetti a processi di *tuning*

Un approccio alternativo al fitness così presentato può consistere nella modellazione del problema mediante il concetto di Fronte di Pareto (e dunque come un problema multi-obiettivo).

**Euristica** : parametro di fitness prioritario

Osservando la presenza di più individui, ciascuno implementante il modulo operativo *Laertes*, sembra evidente che il parametro  $d$ , i.e. il danno, debba pesare in maggior parte, in termini della valutazione del fitness. Intuitivamente infatti, l'individuo che infligge molto danno per breve tempo apporta un significativo beneficio alla popolazione stessa. Un'argomentazione opposta potrebbe tuttavia sostenere che un individuo con un lifespan molto alto è in grado di infliggere più danno in seguito, essendo capace di sopravvivere più a lungo.

Per risolvere il dilemma in mancanza di dati concreti ed in virtù del Rasoio di Occam, si decide, fino a nuovo ordine, di privilegiare il parametro di danno rispetto al lifespan: tuttavia il lifespan sarà la misura di performance valutata per applicare *l'elitismo* nel contesto di tale algoritmo.

Infine un breve cenno di progettazione di dettaglio e implementazione: si prevede di realizzare un sotto-modulo statistico, atto a misurare i parametri di fitness di ciascun individuo presentati in precedenza. Ciò consente di definire una struttura *spider* per il modulo genetico, che utilizzi moduli ulteriori dalle funzionalità coese, ma ben suddivise.

Procediamo ora nella valutazione delle scelte di design effettuate rispetto all'impostazione del GA presentato:

- **Size della popolazione**

Il numero di elementi nella popolazione è variabile. Tuttavia, non è previsto che l'incremento della difficoltà sia dovuto all'incremento del numero di individui, quanto piuttosto al miglioramento iterativo degli stessi. Conseguentemente, sembra ragionevole utilizzare una successione aritmetica di ragione  $r$ , cosicché, denotando con  $|P_i|$  la taglia della popolazione  $i$ -esima si abbia:

$$|P_{i+1}| = |P_i| + r$$

Si scelgono, su pura base intuitiva,  $|P_1| = 5$  ed  $r = 1$ , come taglia della prima popolazione e ragione della successione; tali valori saranno naturalmente ridefiniti in seguito alle procedure di testing.

- **Inizializzazione della popolazione  $P_1$**

La popolazione iniziale non sarà generata, bensì ne saranno fissati i singoli individui. Ciò consente di evitare un eccessivo sbilanciamento di difficoltà iniziale, sia esso difettivo o eccessivo. Nello specifico, riferendosi alla codifica osservata in precedenza, si

congettura (a monte dei collaudi), una popolazione iniziale tale da contenere almeno un genoma di ciascun tipo.

- **Selezione e size del mating pool**

La strategia di selezione è mista. Innanzitutto, in virtù delle ragioni espresse in precedenza, sarà ammesso nella popolazione successiva l'individuo con il lifespan più elevato.

L'algoritmo di selezione propriamente detto segue il paradigma *steady-state*. Il razionale dietro tale scelta prevede che, mediante una crescita dal tasso più lento, si possa offrire al giocatore una più pura esperienza di difficoltà incrementale, avanzando man mano nelle singole scene. In particolare, se  $P$  è la popolazione corrente,  $\lceil \frac{|P|}{3} \rceil$  sarà il numero di individui che, avendo i più bassi valori di fitness nella popolazione  $P$ , saranno scartati.

Come descrivono J. Smith e F. Vavak in *Replacement Strategies in Steady State Genetic Algorithms: Static Environments* [3], strategie del tipo Replace-Worst, per quanto preservino gli individui migliori (a differenza di tecniche differenti, quali le selezioni Replace-Eldest), possono soffrire di convergenza prematura. Si osservi che tale articolo è legato principalmente all'analisi di ambienti statici, tuttavia presenta delle interessanti conclusioni sulle strategie di selezione sopracitate i cui vantaggi e svantaggi possono essere estesi agli ambienti dinamici. Nel nostro caso, la pressione di selezione sarà mitigata dall'introduzione di componenti casuali tra le operazioni di crossover e mutazione.

L'insieme di individui ammessi alla riproduzione, i.e. il mating pool, risulteranno essere i rimanenti  $|P| - \lceil \frac{|P|}{3} \rceil = \lfloor \frac{2|P|}{3} \rfloor$ . Naturalmente, tali individui dovranno ripristinare adeguatamente la taglia della popolazione, ricordando che è necessario generarne una successiva di dimensioni  $|P| + 1$ .

- **Crossover**

Rispetto a quanto osservato circa la selezione, è chiaro che l'algoritmo di crossover dovrà ovviare agli individui scartati nella fase precedente. Pertanto, sarà necessario generare un totale di  $\lceil \frac{|P|}{3} \rceil$  individui.

E' dunque necessario selezionare due individui genitori  $\lceil \frac{|P|}{3} \rceil$  volte. La scelta dei suddetti è casuale tra tutti coloro che hanno passato la selezione (incluso l'individuo elitario). Tale strategia è di semplice comprensione ed ha l'ulteriore vantaggio di essere applicata a valle di una selezione steady-state, in cui individui pessimi sono già stati scartati.

Dal momento che ciascuna coppia di genitori dovrà generare due figli, sarà applicato un algoritmo 3-point crossover ai cromosomi dei genitori, sicché:

- La velocità della prole sarà scelta casualmente tra quelle dei genitori;
- L'arma della prole sarà l'arma del genitore il cui danno è maggiore;
- L'avatar della prole sarà l'avatar del genitore il cui lifespan è maggiore.

Ciò consente di applicare il paradigma steady-state anche in presenza di una struttura di geni più complessa.

- **Mutazione**

Tra gli individui ammessi alla popolazione successiva, ma già presenti in quella considerata (i.e. non considerando gli *offspring* generati in fase di crossover), il 25% degli individui con fitness più basso subirà una mutazione alla velocità di  $\pm v$ , per un  $v$  opportuno. Inoltre, come di seguito specificato:

Sia  $d$  il danno effettuato dall'individuo ed  $l$  il lifespan, sarà applicata una strategia di random resetting del gene:

- Arma, se  $d < l$ , cosicché il parametro di danno possa migliorare;
- Avatar, se  $l \leq d$ , cosicché la resistenza ai colpi e, si presume, il lifespan possa migliorare.

Un tale algoritmo ci consente di introdurre una ragionevole diversità, senza deviare eccessivamente dal paradigma steady-state.

- **Condizione di terminazione**

L'evoluzione proseguirà fino alla sconfitta del giocatore. La condizione di terminazione è pertanto *esterna* al GA proposto ed è formalmente concretizzata all'iterazione  $\tau$ , all'istante di tempo nel quale il parametro di salute del giocatore scende a zero. La generazione  $P_\tau$  che ha sconfitto il giocatore è considerabile come il valore di restituzione del modulo genetico.

I punti precedenti consentono una caratterizzazione completa dell'algoritmo presentato. Come già osservato, sarà applicato il concetto di elitismo: sarà automaticamente promosso alla popolazione successiva l'individuo il cui parametro lifespan è il più elevato nella popolazione corrente.

Si precisa, in conclusione, che eventuali varianti considerate rispetto ai moduli Hamlet e Laertes saranno valutate in base a misure metriche quali lo *Average Fitness Value*. Il valore di fitness medio, considerate le scelte effettuate in precedenza e la contestualizzazione dell'agente nell'ambiente di gioco, rappresenta pertanto la difficoltà media incontrata dal giocatore e consente di interpretare un responso circa la crescita progressiva del suddetto livello di difficoltà, in concordanza con il livello di abilità raggiunto dalle istanze dell'agente intelligente.

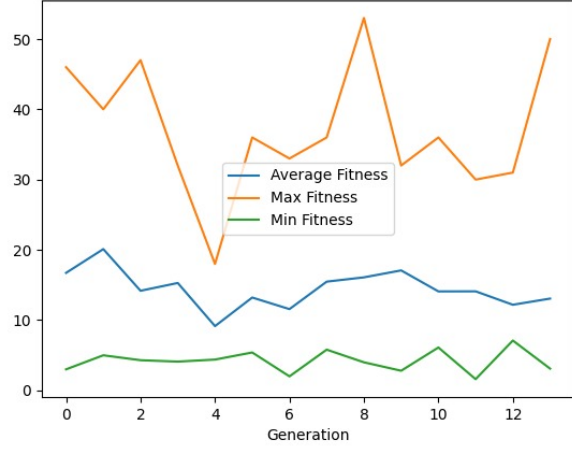
---

I risultati dell'esecuzione dell'algoritmo, senza mutazione, sono promettenti e tuttavia, ancora imperfetti. Come si osserva:

```

[11:08:15] Begin of generation 1 / #population = 5
UnityEngine.Debug.Log (object)
[11:08:29] End of generation 1 / AvgFitness = 9,412696
UnityEngine.Debug.Log (object)
[11:08:29] Begin of generation 2 / #population = 6
UnityEngine.Debug.Log (object)
[11:08:59] End of generation 2 / AvgFitness = 12,9399
UnityEngine.Debug.Log (object)
[11:08:59] Begin of generation 3 / #population = 7
UnityEngine.Debug.Log (object)
[11:09:14] End of generation 3 / AvgFitness = 16,34155
UnityEngine.Debug.Log (object)
[11:09:14] Begin of generation 4 / #population = 8
UnityEngine.Debug.Log (object)
[11:09:29] End of generation 4 / AvgFitness = 14,02271
UnityEngine.Debug.Log (object)
[11:09:29] Begin of generation 5 / #population = 9
UnityEngine.Debug.Log (object)
[11:09:44] End of generation 5 / AvgFitness = 15,42341
UnityEngine.Debug.Log (object)
[11:09:44] Begin of generation 6 / #population = 10
UnityEngine.Debug.Log (object)

```

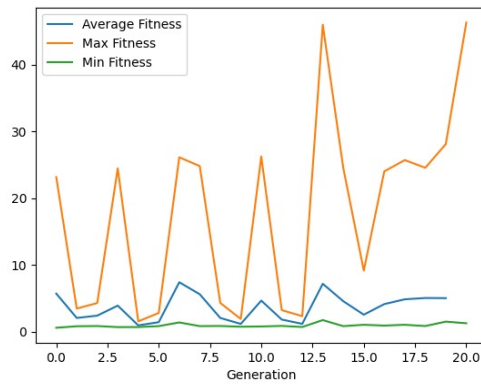


Il fitness medio resta pressoché invariato, ma la cosa non sorprende: con il progredire delle generazioni il numero di individui aumenta! La deduzione logica è che il fitness totale deve proporzionalmente aumentare, cosicché:

$$AvgFitness_{P_i} = \frac{\sum_{j=1}^{|P_i|} fitness(I_j)}{|P_i|} \approx \frac{\sum_{j=1}^{|P_{i-1}|} fitness(I_j)}{|P_{i-1}|} = AvgFitness_{P_{i-1}}$$

Dove  $I_j$  descrive il j-esimo individuo della popolazione considerata e, a runtime,  $i \rightarrow \infty$ .

E' molto interessante osservare che, con l'introduzione della mutazione, la musica cambia radicalmente, aggiungendo l'inclinazione voluta alla curva del fitness medio (il che implica che il fitness totale aumenta significativamente, rendendo l'algoritmo ulteriormente efficace). Come riporta il grafico seguente:

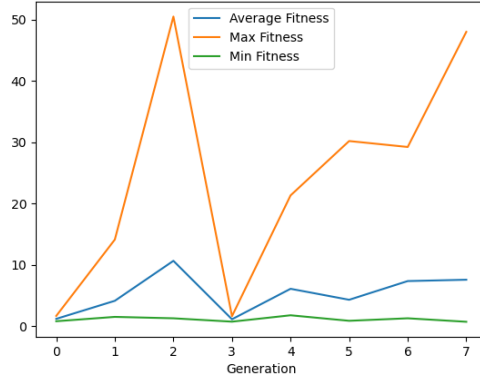


E' curioso come la crescita descriva ampi "saliscendi" con picchi di crescita e decrescita del fitness massimo e medio. Intuitivamente, l'entropia del sistema è aumentata con l'introduzione della mutazione. Tuttavia, si osservi come ciascun picco massimo sia superiore al precedente.

Si rende dunque necessaria una fase di *tuning* in cui modulare dati parametri del GA proposto. Innanzitutto, il confronto tra danno e lifespan, rispetto alla fase di valutazione del fitness e mutazione non può essere condotto direttamente. I due hanno unità di misura differenti, in intervalli differenti, e devono pertanto essere normalizzati rispetto ad un intervallo comune. Applicando la min-max normalization nell'intervallo  $[0, 1]$  e comparando i valori normalizzati si ottiene una mutazione più stabile.

Ora che la mutazione è stata stabilizzata, è possibile aggiungere diversità considerando più individui. Ricordiamo che il 25% proposto è calcolato senza tener conto della prole generata in fase di crossover, né dell'individuo elitario. Moduliamo tale valore lasciandolo salire al 33%. La crescita risulta ora più lenta, e soprattutto più lineare, come voluto per fornire al giocatore la difficoltà incrementale.

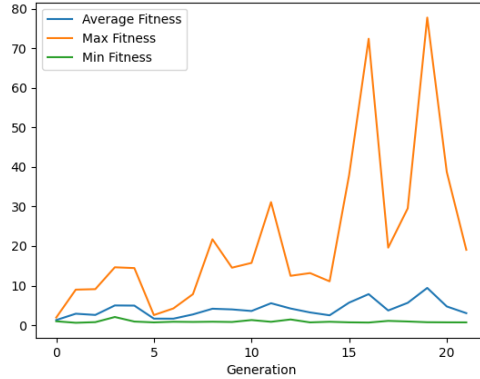
Lasciamo inoltre invariata la popolazione iniziale, cosicché i vari possibili genomi continuino ad essere presenti sin dall'inizio. Osserviamo cosa accade al grafico di crescita del fitness;



Infine, il fitness può ora essere modulato. Osserviamo cosa accade tramutando l'espressione del fitness in:

$$f(r) = \frac{3}{2}d + p + l$$

In particolare, ora il danno pesa il 150% del lifespan. Il grafico mostra:





Rammentando la dinamicità e stocasticità dell'ambiente derivante dalla presenza di un giocatore umano, è ragionevole osservare la presenza di alcuni picchi, specialmente per quanto concerne il fitness massimo. Tuttavia, si osservi nuovamente come ciascun picco sia in genere superiore al precedente, e come l'abilità del giocatore possa essere osservata in funzione di essi.

Infine, si noti come il *tuning* dell'algoritmo ha portato ad una crescita sicuramente più lineare del fitness medio, ottenendo la tanto agognata difficoltà incrementale derivante dall'abilità crescente dei singoli individui.

### 3.2.3 Ophelia: Il modulo di apprendimento

*'Tis in my memory lock'd, and you yourself shall keep the key of it.*

*Ophelia - Act I Scene III*

Ophelia è il nostro modulo di apprendimento. Presentato sin dalla descrizione architetturale dell'agente, esso è inteso nel sistema come una potenziale espansione futura ed ha il particolare compito di istruire il modulo genetico rispetto alla valutazione del fitness degli individui.

## 4 Glossario

<b>Robomaquia</b>	Il nome del progetto discusso nella presente. Descrive un videogioco integrato con più moduli di intelligenza artificiale. E' una macedonia di parole ed è traducibile come La Corrida dei robot!
<b>Laertes</b>	Il modulo operativo del progetto. E' implementato da ciascuno degli individui della popolazione ed ha l'obiettivo di individuare ed attaccare il giocatore.
<b>Hamlet</b>	Il modulo risultante dall'implementazione di un GA. E' responsabile dell'evoluzione della popolazione di nemici secondo i criteri di fitness.
<b>Ophelia</b>	Il modulo di apprendimento del progetto. E' definito come una potenziale espansione futura.
<b>Precisione</b>	Il rapporto tra i colpi nemici andati a segno contro il giocatore ed i colpi totali tentati.
<b>Lifespan</b>	La somma degli intervalli di tempo in cui l'agente è rimasto in vita, ingaggiando in combattimento il giocatore.
<b>Campo visivo (FOV)</b>	La regione dello spazio nella quale un'istanza dell'agente vede il giocatore. L'obiettivo dell'algoritmo di ricerca che esplora la stanza è far sì che il giocatore rientri in tale campo.
<b>Campo di attacco (FOA)</b>	La regione dello spazio nella quale un'istanza dell'agente attacca il giocatore.
<b>Stanza/Scena</b>	Lo spazio di gioco. Il giocatore può uscire dalla stanza quando ha sconfitto ciascuna delle istanze dell'agente, accedendo alla stanza successiva.
<b>Quadrante</b>	Una suddivisione della stanza di gioco. Tale astrazione torna utile nel contesto dell'algoritmo di ricerca.
<b>Swarm intelligence</b>	Un paradigma che prende a piene mani dal comportamento socio-biologico delle colonie di animali. Tale idea è utilizzata, seppur solo spiritualmente nell'ambito di alcune euristiche applicate dai moduli del progetto.
<b>Avatar</b>	Descrive l'aspetto di ciascun individuo e dunque lo sprite ad esso associato. Con riferimento al modulo Hamlet, è usato per determinare la resistenza ai colpi, l'inerzia al movimento e la hitbox (i.e. la taglia).

## Bibliografia

- [1] Hazem Ahmed - Janice Glasgow. *Swarm Intelligence: Concepts, Models and Applications*. URL: [https://www.researchgate.net/publication/264457775\\_Swarm\\_Intelligence\\_Concepts\\_Models\\_and\\_Applications](https://www.researchgate.net/publication/264457775_Swarm_Intelligence_Concepts_Models_and_Applications).
- [2] Alexey Skrynnik - Anton Andreychuk - Konstantin Yakovlev - Aleksandr Panov. *Pathfinding in stochastic environments: learning vs planning*. URL: <https://peerj.com/articles/cs-1056/>.
- [3] Jim Smith - Frank Vavak. «Replacement Strategies in Steady State Genetic Algorithms: Static Environments». In: *Intelligent Computer System Centre - University of the West of England* ().