


# EvoPath

Algoritmo genetico per la generazione di mappe  
dinamiche

 <https://github.com/Gianpyy/evopath>

Docente:

**Prof.**

**Fabio Palomba**

Studenti:

**Claudio Buono**

**Gianpio Silvestri**

ANNO ACCADEMICO 2023/2024

# Indice

<b>1</b>	<b>Introduzione</b>	<b>2</b>
<b>2</b>	<b>Definizione del problema</b>	<b>3</b>
2.1	Obiettivo . . . . .	3
2.2	Analisi del problema . . . . .	3
2.3	Formulazione PEAS . . . . .	5
2.4	Caratteristiche dell'ambiente . . . . .	5
<b>3</b>	<b>Tecnologie adottate</b>	<b>6</b>
<b>4</b>	<b>Soluzione del problema</b>	<b>7</b>
4.1	Tipo di IA . . . . .	7
4.2	Algoritmo Genetico . . . . .	8
4.2.1	Codifica degli individui . . . . .	8
4.2.2	Fitness . . . . .	9
4.2.3	Selezione . . . . .	10
4.2.4	Crossover . . . . .	11
4.2.5	Mutazione . . . . .	12
4.2.6	Budget di ricerca . . . . .	13
4.2.7	Ricerca del percorso . . . . .	13
<b>5</b>	<b>Risultati</b>	<b>14</b>
5.1	Parametri . . . . .	14
5.2	Analisi dei risultati . . . . .	14
5.2.1	Configurazione iniziale . . . . .	15
5.2.2	Operatori genetici . . . . .	16
5.2.3	Pesi della fitness . . . . .	18
<b>6</b>	<b>Considerazioni finali</b>	<b>20</b>

# 1 Introduzione

Nel contesto dei videogiochi di tipo tower defense<sup>1</sup>, la creazione di percorsi dinamici rappresenta una sfida cruciale per il design di livelli coinvolgenti e complessi. L'introduzione di un algoritmo genetico per la generazione di percorsi offre una soluzione innovativa, in grado di creare percorsi unici e complessi, senza doverli costruire manualmente.

L'utilizzo di un algoritmo genetico applicato alla generazione di percorsi in un gioco tower defense permette di esplorare una vasta gamma di possibilità, producendo percorsi che variano in termini di lunghezza, complessità e strategia richiesta per difenderli.

Inoltre grazie alla possibilità di poter cambiare i parametri dell'algoritmo, si può ottenere una mappa di gioco che si adatti alle esigenze richieste.

---

<sup>1</sup>Un genere di videogiochi strategici in cui il giocatore deve difendere un obiettivo dagli attacchi di nemici che seguono un percorso prestabilito.

## 2 Definizione del problema

In questo capitolo verranno esaminati gli obiettivi del progetto, analizzato il problema e presentata la formulazione PEAS.

### 2.1 Obiettivo

L'obiettivo di questo progetto è sviluppare un algoritmo genetico per generare mappe contenenti ostacoli e un percorso che va da un punto di ingresso a un punto di uscita, da utilizzare in un videogioco di tipo tower defense.

### 2.2 Analisi del problema

Per la realizzazione di questo progetto è stata presa come riferimento la ricerca [1], che esplora l'uso dei Chess Maze per la creazione di puzzle applicabili al game design.

Un Chess Maze è definito dalla disposizione, su una griglia, di un certo numero di pezzi degli scacchi. Questi pezzi non si muovono; piuttosto, coprono tutte le posizioni nelle quali potrebbero arrivare con una sola mossa valida per il tipo di pezzo.

Nel contesto del nostro problema a partire da un Chess Maze si può ricavare una mappa di gioco che contenga un percorso che colleghi un punto di ingresso ad un punto di uscita.

Dunque, gli elementi di un Chess Maze posso essere convertiti in elementi del nostro problema nella seguente maniera:

- I pezzi disposti sulla scacchiera e le loro possibili mosse costituiscono degli ostacoli
- Le caselle rimanenti costituiscono degli spazi vuoti

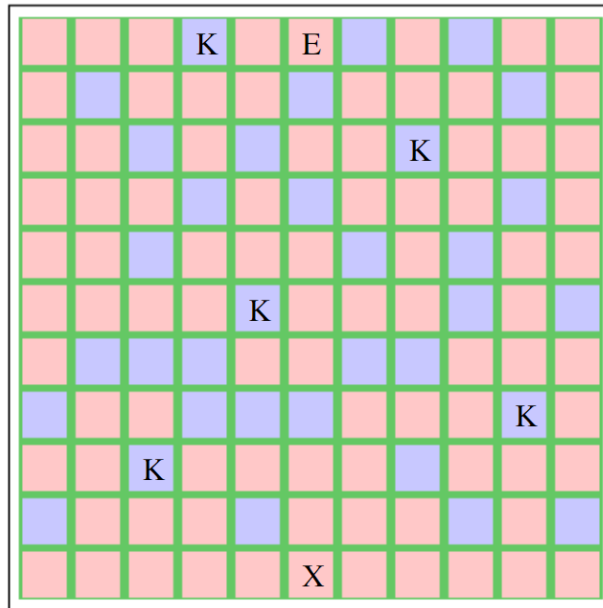


Figura 2.1: Esempio di Chess Maze

Nella griglia di sopra le **caselle blu** indicano la presenza di un ostacolo, mentre le **caselle rosse** indicano le caselle libere. La lettera al di sopra della casella indica cosa rappresenta, e può essere:

- [**K**]: Knight (Cavallo) posizionato sulla griglia
- [**X**]: Ingresso del percorso
- [**E**]: Uscita del percorso

Dati questi elementi, sarà possibile andare a costruire un percorso che parta da un punto di ingresso e arrivi ad un punto di uscita, percorrendo le caselle libere ed evitando gli ostacoli disposti.

## 2.3 Formulazione PEAS

Di seguito è riportata la formulazione P.E.A.S. adattata al nostro problema:

- **Performance.** La misura di prestazione adottata prevede la massimizzazione della lunghezza del percorso, del numero di ostacoli e del numero di curve (rientrando comunque in un determinato range di curve)
- **Environment.** L'ambiente in cui opera l'agente è costituito dallo stato attuale della mappa, ossia dalla posizione attuale degli ostacoli nella mappa
- **Actuators.** L'agente agisce sull'ambiente andando a modificare la posizione degli ostacoli e di conseguenza il percorso
- **Sensors.** L'agente percepisce l'ambiente prendendo lo stato attuale della mappa e valutando la posizione degli ostacoli su di essa

## 2.4 Caratteristiche dell'ambiente

Di seguito sono riportate le caratteristiche dell'ambiente:

- **Completamente osservabile.** In ogni momento l'agente conosce tutta la mappa tra cui l'inizio e la fine del percorso e i vari ostacoli presenti
- **Stocastico.** L'evoluzione è influenzata da elementi casuali generati durante il processo, data la natura degli operatori genetici
- **Sequenziale.** La soluzione finale è data dal raffinamento sequenziale di quelle precedenti
- **Dinamico.** La mappa cambia ad ogni generazione
- **Discreto.** Il numero di percezioni dell'agente è limitato
- **Singolo agente.** Data la natura del problema vi sarà un unico agente a cui sarà affidato il compito di generare una mappa

### 3 Tecnologie adottate

Per lo sviluppo è stato scelto di utilizzare Godot, un game engine che sta guadagnando crescente popolarità negli ultimi anni. Il linguaggio di programmazione utilizzato per l'implementazione dell'algoritmo genetico è stato C#, date le competenze pregresse e la buona integrazione con Godot, mentre GDScript<sup>2</sup> è stato utilizzato per gestire l'interfaccia utente, data la piena compatibilità con il game engine e facilità di integrazione.

I dati generati dalle diverse configurazioni dei parametri dell'algoritmo genetico sono stati salvati in Excel per facilitare l'analisi dei risultati.

---

<sup>2</sup>Linguaggio di scripting specifico per Godot Engine, progettato per essere semplice, flessibile e ben integrato con le funzionalità di Godot.

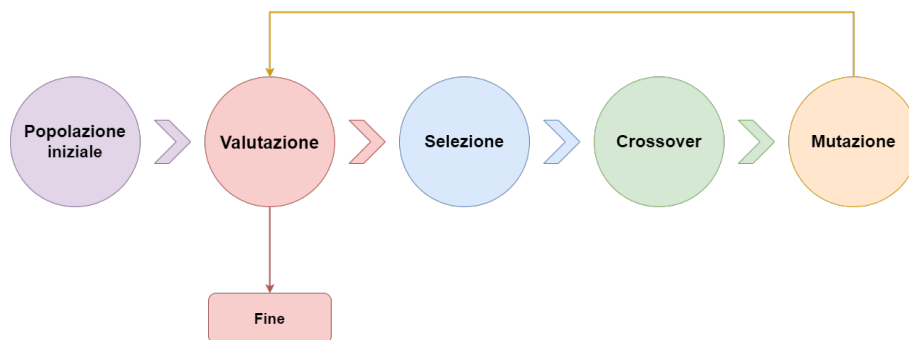
## 4 Soluzione del problema

In questo capitolo verrà analizzata la tipologia di IA scelta per la soluzione del problema e verranno esaminate le tecniche implementate.

### 4.1 Tipo di IA

Il problema richiede che le mappe generate siano sempre diverse tra loro. Gli algoritmi genetici, grazie alla loro capacità di esplorare ampi spazi di soluzioni e di mantenere una popolazione diversificata, risultano essere una buona soluzione a questo tipo di problema.

Gli algoritmi genetici sono una classe di algoritmi di ottimizzazione ispirati ai principi dell'evoluzione naturale. Questi algoritmi operano attraverso una popolazione di soluzioni candidate, evolvendole nel tempo tramite operazioni di selezione, crossover e mutazione.



Utilizzare un algoritmo genetico per risolvere questo problema permette di avere sempre delle mappe diverse in base alla configurazione adottata per la generazione della mappa.

Una preoccupazione comune con l'uso di algoritmi genetici è il loro tempo di esecuzione, che può essere significativo soprattutto per problemi complessi. Tuttavia, in un contesto videoludico, questo problema può essere mitigato efficacemente: durante il caricamento di un nuovo livello l'algoritmo genetico può essere eseguito per generare la mappa successiva e mascherare il tempo di calcolo.



## 4.2 Algoritmo Genetico

Di seguito saranno dettagliati gli elementi che andranno a costituire l'algoritmo genetico.

### 4.2.1 Codifica degli individui

Un individuo è rappresentato da una mappa candidata, una struttura dati che verrà utilizzata nelle varie operazioni dell'algoritmo genetico. Una mappa candidata contiene:

- **Griglia della mappa.** Una matrice che conterrà gli elementi che costituiranno la mappa
- **Ostacoli.** Un array booleano i cui elementi saranno *true* nelle caselle che corrispondono ad un ostacolo nella griglia; la scelta dell'array è dovuta ad una più agevole implementazione delle operazioni di crossover e mutazione.
- **Percorso.** La lista con le coordinate delle caselle della griglia che andranno a formare il percorso
- **Ingresso e uscita.** Le coordinate delle caselle di ingresso e uscita della mappa.

E	E	0	0	0	E	E	0	0	E
E	0	E	0	E	0	E	E	E	E
0	E	E	0	0	0	E	0	0	0
0	0	R	R	R	0	0	E	E	E
0	R	R	0	R	R	R	R	0	E
R	R	0	E	0	0	0	R	0	X
S	0	0	0	0	0	E	R	R	R
0	E	E	0	0	0	E	0	E	0
0	0	0	E	E	0	0	E	E	E
E	E	E	E	E	0	E	0	E	0

Figura 4.1: Esempio di codifica di un individuo

### 4.2.2 Fitness

La funzione di fitness è fondamentale per determinare la qualità di un individuo, influenzando la sua probabilità di essere selezionato durante l'operazione di selezione. Nel nostro problema, la funzione di fitness deve essere calcolata tenendo conto di obiettivi multipli, i quali sono:

1. **Lunghezza del percorso:** La lunghezza totale del percorso dal punto di ingresso al punto di uscita
2. **Numero di ostacoli:** La quantità di ostacoli presenti sulla mappa
3. **Numero di curve:** Il numero di curve presenti nel percorso, che dovranno rientrare nel range  $[CornerMin, CornerMax]$

Per ottimizzare la funzione di fitness, saranno utilizzati dei pesi, ossia dei parametri che assegnano un'importanza relativa a ciascuno degli obiettivi.

La funzione di fitness,  $F$ , può essere espressa come una combinazione lineare dei vari obiettivi:

$$F = w_1 \cdot L + w_2 \cdot O + w_3 \cdot C$$

dove:

- $L$  rappresenta la lunghezza del percorso
- $O$  rappresenta il numero di ostacoli
- $C$  rappresenta il numero di curve
- $w_1, w_2, w_3$  sono i pesi associati a ciascun obiettivo

Il valore di  $C$  sarà:

- il numero di curve, se  $C \in [CornerMin, CornerMax]$
- il numero di curve in eccesso, se  $C > CornerMax$
- il valore di  $CornerMin$ , se  $C < CornerMin$

$C$  assume valore **negativo** se non rientra nel range in modo tale da penalizzare gli individui che non rispettano l'obiettivo di massimizzare le curve.

### 4.2.3 Selezione

La fase di selezione determina quali individui della popolazione corrente verranno scelti per la riproduzione, contribuendo alla generazione successiva. Di seguito verranno elencate le strategie di selezione implementate.

#### Roulette Wheel Selection

La *Roulette Wheel Selection* è una tecnica che assegna una probabilità di selezione a ciascun individuo in proporzione al suo valore di fitness. Questa metodologia favorisce la selezione degli individui con una fitness più alta, ma permette comunque una certa diversità dando una possibilità di selezione anche agli individui con fitness inferiore.

Ad ogni individuo viene assegnata una probabilità  $p_i$  di essere selezionato calcolata in base alla formula

$$p_i = \frac{f_i}{\sum_{j=1}^N f_j}$$

Dopodiché, vengono selezionati casualmente  $n$  individui. Nel nostro caso vengono selezionati 2 individui.

#### Truncation Selection

La *Truncation Selection* è una strategia più semplice e deterministica. Questa metodologia è meno probabilistica rispetto alla *Roulette Wheel Selection* e tende a ridurre la diversità genetica più rapidamente, poiché si focalizza solo sugli individui con la fitness più alta.

Gli individui vengono ordinati in base al loro valore di fitness, e solo una percentuale dei migliori individui viene selezionata per la riproduzione.

Nel nostro caso, come per la *Roulette Wheel Selection*, verranno selezionati solo 2 individui.

#### 4.2.4 Crossover

Il crossover combina le informazioni di due genitori per generare nuovi individui, detti figli. Esistono diverse tecniche di crossover, ciascuna con caratteristiche e vantaggi specifici. Nella nostra soluzione, tutte le strategie di crossover verranno applicate sull'array degli ostacoli, dove ogni elemento dell'array rappresenta un gene. Questa scelta è dovuta dalla necessità di creare nuovi spazi nella mappa che permetteranno all'algoritmo la creazione di un nuovo percorso. Di seguito sono elencate le strategie di crossover implementate.

##### Single Point Crossover

Il *Single Point Crossover* sceglie casualmente un punto di crossover all'interno dei genitori e vengono scambiate le parti dei genitori dopo il punto, andando a creare due nuovi individui.

##### Uniform Crossover

Il *Uniform Crossover* differisce dal single point crossover in quanto non dipende da un singolo punto di crossover. Invece, ogni gene del figlio viene scelto indipendentemente da uno dei due genitori con una probabilità prefissata, che nel nostro caso sarà del 50%.

##### Block Swap Crossover

Il *Block Swap Crossover* prevede lo scambio di blocchi di geni tra i genitori. Nel contesto del nostro problema un blocco è costituito da  $n$  ostacoli consecutivi, per un valore massimo di  $maxBlockSize$ . La probabilità che due blocchi vengano scambiati è data dalla seguente formula

$$\frac{fitness1}{fitness1 + fitness2}$$

dove:

- *fitness1* rappresenta il valore di fitness del genitore 1
- *fitness2* rappresenta il valore di fitness del genitore 2

Questo ci permette di dare una probabilità minore di scambio al genitore con un valore di fitness maggiore, in modo tale da preservarne i geni.

### Heuristic Crossover

L'*Heuristic Crossover* salva i geni del genitore con il valore di fitness maggiore in entrambi i figli. Ne consegue che i due figli saranno dei gemelli.

## 4.2.5 Mutazione

La mutazione è utilizzata per mantenere la diversità genetica all'interno della popolazione ed evitare che l'algoritmo si blocchi in ottimi locali. Anche in questo caso, come per il crossover, andiamo ad applicare le tecniche di mutazione sull'array degli ostacoli, mutando uno o più elementi da *true* a *false* e viceversa. Le motivazioni che ci portano a questa scelta sono identiche a quelle espresse per il crossover. Di seguito sono descritte le tecniche di mutazione implementate.

### Bit-flip Mutation

Nel *Bit-flip Mutation* ogni elemento ha una probabilità di essere mutato determinata dal valore di *mutationRate*.

### Block Mutation

La *Block Mutation* consiste nel selezionare e mutare dei blocchi di ostacoli contigui. Il numero di ostacoli che verranno mutati per ogni blocco è uguale alla metà del numero di ostacoli presente in un blocco.

### Simulated Annealing Mutation

La *Simulated Annealing Mutation* valuta tutti gli elementi dell'array e li muta con una probabilità pari a *currentTemperature*, che va diminuendo ad ogni mutazione, sia essa effettuata o fallita.

### 4.2.6 Budget di ricerca

Per il budget di ricerca si è optato per una stopping condition ibrida basata su:

- **Numero di generazioni.** Un valore fissato a priori che andrà a definire il numero massimo di generazioni che l'algoritmo potrà generare.
- **Assenza di miglioramenti.** Un limite entro il quale dovrà essere prodotta una generazione con un individuo con un valore di fitness maggiore rispetto al miglior individuo di sempre.

### 4.2.7 Ricerca del percorso

Per la ricerca del percorso dal punto di ingresso al punto di uscita è stato utilizzato l'algoritmo di ricerca  $A^*$ , che restituirà le coordinate delle caselle della griglia che formeranno il percorso.

Data la presenza di ostacoli sulla mappa, non è detto che  $A^*$  sia in grado di trovare con successo un percorso che vada dal punto di ingresso al punto di uscita. Per questa ragione, ad ogni individuo di una generazione verrà applicato l'algoritmo *Repair* che rimuove ostacoli in maniera casuale finché non verrà trovato un percorso. In questo modo si avranno sempre individui con un percorso che vada dal punto di ingresso al punto di uscita, ottenendo così sempre delle soluzioni ammissibili al problema.

# 5 Risultati

In questo capitolo verranno analizzati i dati ottenuti applicando le tecniche descritte nel capitolo precedente, con l'obiettivo di trovare una configurazione ottimale.

## 5.1 Parametri

Data la necessità di un algoritmo che fosse il più flessibile possibile, in modo tale da poter avere delle mappe che si adattassero ad esigenze diverse, si sono raggruppati i parametri in categorie, così da configurare al meglio l'algoritmo. Dunque, i parametri dell'algoritmo sono stati raggruppati in tre categorie:

- **Pesi.** I pesi associati ai parametri della funzione di fitness
- **Impostazioni algoritmo genetico.** I parametri associati alla configurazione della mappa ed alla configurazione dell'algoritmo genetico
- **Operatori.** Gli operatori genetici implementati

## 5.2 Analisi dei risultati

In questa sezione saranno esplorate diverse configurazioni con l'obiettivo di migliorare la fitness media della miglior mappa e il tempo di esecuzione, tenendo comunque conto degli altri parametri considerati. Ogni configurazione è stata analizzata tramite l'esecuzione di 100 iterazioni dell'algoritmo genetico per valutarla secondo la media ed il valore massimo delle seguenti metriche: generazioni usate, fitness miglior mappa, lunghezza percorso, numero curve, numero ostacoli, tempo di esecuzione.

### 5.2.1 Configurazione iniziale

La configurazione iniziale sulla quale saranno basati i primi risultati è la seguente:

#### **Pesi**

- Tutti i pesi hanno valore 1

#### **Impostazioni algoritmo genetico**

- **Dimensione popolazione:** 20
- **Limite generazioni:** 30
- **Probabilità crossover:** 100
- **Probabilità mutation:** 5
- **Larghezza mappa:** 10
- **Altezza mappa:** 10
- **Numero di Knight piazzati:** 5
- **Numero massimo generazioni senza miglioramenti:** 5

#### **Operatori genetici**

- **Selection:** Roulette Wheel
- **Crossover:** Single Point
- **Mutation:** Bit-flip



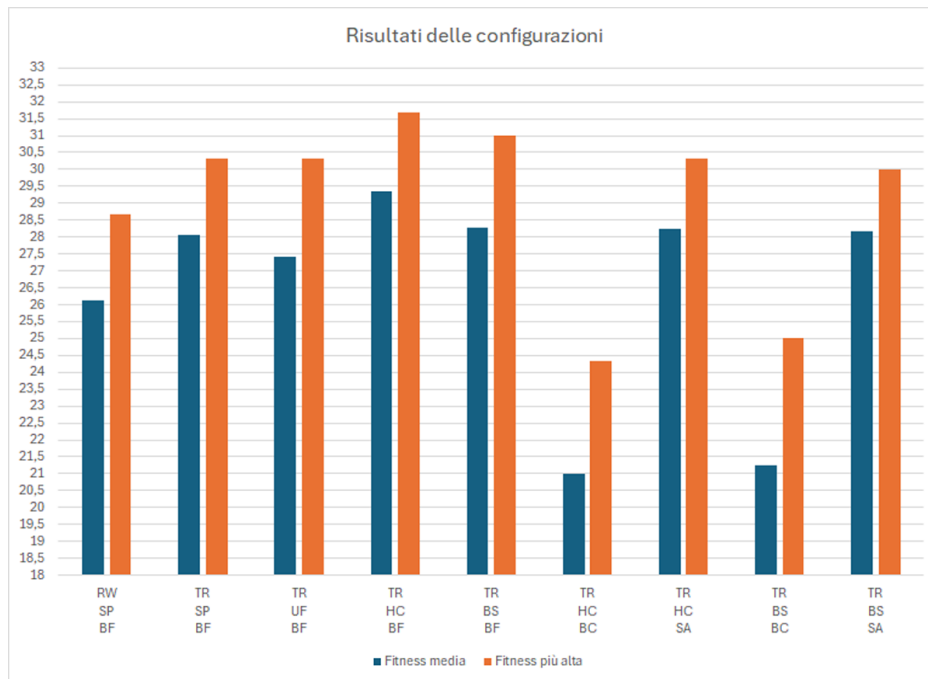
Tabella risultati

	Media	Massimo
<b>Generazioni usate</b>	13.16	29
<b>Fitness miglior mappa</b>	25.16	28
<b>Lunghezza percorso</b>	22.24	31
<b>Numero curve</b>	10.62	12
<b>Numero ostacoli</b>	42.64	57
<b>Tempo di esecuzione</b>	00:11141	00:22811

Dai risultati si evince che il numero di generazioni usate in media (13.16) è molto inferiore rispetto al limite di generazioni per questa configurazione (30). Pertanto, è stato impostato il numero massimo di generazioni senza miglioramenti a 10 per le prossime configurazioni.

## 5.2.2 Operatori genetici

Successivamente sono state provate diverse combinazioni di operatori genetici tra quelli descritti nel capitolo 4, così da trovare la combinazione che ci dia il valore più alto di fitness media. Di seguito è riportato il grafico con tutte le combinazioni provate e la loro fitness media e più alta.



Si può dedurre dal grafico che la miglior combinazione di operatori genetici è la seguente:

- **Selection:** Truncation (TR)
- **Crossover:** Heuristic (HC)
- **Mutation:** Bit-flip (BF)

**Tabella risultati**

	<b>Media</b>	<b>Massimo</b>
<b>Generazioni usate</b>	26.61	29
<b>Fitness miglior mappa</b>	29.36	31.66
<b>Lunghezza percorso</b>	23.24	32
<b>Numero curve</b>	11.41	12
<b>Numero ostacoli</b>	53.45	67
<b>Tempo di esecuzione</b>	00:21308	00:30332

Dopo aver trovato questa configurazione si è provato a riabbassare il numero massimo di generazioni senza miglioramenti a 5 ed i risultati sono espressi nella seguente tabella:

**Tabella risultati**

	<b>Media</b>	<b>Massimo</b>
<b>Generazioni usate</b>	18.03	29
<b>Fitness miglior mappa</b>	28.26	32.33
<b>Lunghezza percorso</b>	23.19	30
<b>Numero curve</b>	11.42	12
<b>Numero ostacoli</b>	50.17	64
<b>Tempo di esecuzione</b>	00:16126	00:27629

Si nota che la fitness è leggermente più bassa rispetto alla tabella precedente, ma è anche significativamente più basso il tempo di esecuzione. Per cui, si è scelto di mantenere la configurazione con il minor tempo di esecuzione così da non avere un tempo di generazione delle mappe troppo elevato.

### 5.2.3 Pesì della fitness

A partire dalla configurazione precedente, è stato cercato di individuare la combinazione ottimale di pesi per la funzione di fitness, con l'obiettivo di bilanciarli e ottenere mappe che rispecchino le nostre aspettative in termini di lunghezza e numero di curve, mantenendo al contempo un valore di fitness elevato.

La combinazione di pesi che è risultata essere la migliore è la seguente:

- Peso della lunghezza del percorso: 5
- Peso degli ostacoli: 3
- Peso delle curve: 2

I risultati della configurazione con la combinazione di pesi appena espressa sono i seguenti:

**Tabella risultati**

	<b>Media</b>	<b>Massimo</b>
<b>Generazioni usate</b>	18.40	29
<b>Fitness miglior mappa</b>	29.14	32.50
<b>Lunghezza percorso</b>	25.34	34
<b>Numero curve</b>	11.14	13
<b>Numero ostacoli</b>	47.59	64
<b>Tempo di esecuzione</b>	00:16754	00:27992

A partire dalla configurazione appena trovata, è stato provato ad aumentare gradualmente la dimensione della popolazione. Questo ha portato ad un leggero aumento della fitness a discapito di un evidente aumento del tempo di esecuzione, per cui è stato ripristinato il valore iniziale di 20 individui.

Infine, si è tentato un valore della probabilità di crossover inferiore al 100%, ma questo è risultato in un abbassamento dei valori di fitness, per cui è stato mantenuto il valore iniziale.

Arrivati a questo punto, la configurazione con il miglior compromesso tra fitness e tempo di esecuzione che genera delle ottime mappe è la seguente:

#### **Operatori genetici**

- **Selection:** Truncation
- **Crossover:** Heuristic
- **Mutation:** Bit-flip

#### **Pesi della fitness**

- **Peso delle curve:** 2
- **Peso degli ostacoli:** 3
- **Peso della lunghezza del percorso:** 5

## 6 Considerazioni finali

In conclusione, gli obiettivi prefissati sono stati raggiunti, ottenendo risultati soddisfacenti che hanno permesso di esplorare le potenzialità e i limiti di un algoritmo genetico. Il prodotto finale rappresenta un'ottima base per futuri miglioramenti implementativi, rendendolo idoneo per l'utilizzo in un videogioco tower defense.

Questa esperienza ha migliorato le nostre competenze nel campo dell'intelligenza artificiale e nello sviluppo di videogiochi, oltre a rafforzare la capacità di coordinamento all'interno di un progetto di team.

# Bibliografia

- [1] Daniel Ashlock. Automatic generation of game elements via evolution. pages 289 – 296, 09 2010.