UniBa | UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO

Pathfinder^2

Algoritmo genetico di pathfinding per Pathfinder RPG

Laurea magistrale in Data Science
Modelli Decisionali e Ottimizzazione

Ivan Diliso

Dipartimento di Informatica
Università degli studi di Bari Aldo Moro

Indice

1 Introduzione

L'obiettivo del progetto è definire un algoritmo genetico di pathfinding per il gioco di ruolo Pathfinder. Pathfinder è un gioco da tavolo con componente ruolistica e di combattimento in cui diversi i giocatori impersonano un avatar, il loro personaggio (da ora in poi chiamato PG), all'interno di una storia narrata da un giocatore chiamato "Dungeon Master" (da ora in poi chiamato DM). Il gioco implementa svariate regole che permettono di definire le dinamiche di combattimento dei PG contro i nemici, comandati dal DM. Il combattimento avviene su mappa quadrettata (griglia) dove il DM disegna le diverse stanze dei "Dungeon", luoghi che i PG devono esplorare combattendo i nemici al loro interno e raggiungendo l'obiettivo definito dal DM.

1.1 Contesto di applicazione

Lo specifico dominio di applicazione del progetto è il seguente: Si immagini di essere un ladro (personaggio molto bravo a nascondersi e non farsi individuare dai nemici) e di dover raggiunre un obiettivo all'interno della stanza di un dungeon, questa stanza potrebbe avere molti pericoli, quali nemici, terreno scomodo (ad esempio un punto in cui il pavimento ha ceduto, rendendo difficile e scomodo il movimento), ostacoli (quali tavoli, sedie, letti, o qualsiasi elemento che può ostacolare i movimenti del personaggio) e ovviamente la presenza di nemici che pattugliano la stanza. Il ladro non conosce la posizione dell'obietivo, ne la conformazione della stanza. Ci chiediamo quindi se sia possibile trovare un percorso che sia:

- Il più breve possibile, che cerchi quindi di evitare ostacoli e terreno scomodo
- Minimizzi la probabilità di essere percepiti dal nemico

In questo constesto il ladro NON attaccherà i nemici, e cercherà soltanto di raggiungere l'obiettivo senza farsi scoprire e nel minor tempo possibile, questa è infatti una dinamica molto ricorrente in qualsiasi sessione di gioco di Pathfinder.

1.2 Regole di Pathfinder

Pathfinder ha una infinità di regole che gestiscono movimenti, attacchi, mosse speciali, etc etc. In questo caso però ci limiteremo ad usare le regole relative al movimento e una parte delle regole relative alla percezione dei nemici (abilità di vedere cose nascoste) e alla furtività del personaggio (abilità di nascondersi alla vista dei nemici).

1.2.1 Movimento

Pathfinder si gioca su una griglia quadrettata, i personaggi si muovo di quadretto in quadretto. Ogni quadretto sulla mappa ha le dimensioni canoniche di 1.5m x 1.5m, ogni giocatore ha un massimo numero di quadretti che può effettuare in un round di combattimento, ogni movimento da un quadretto ad un altro costa 1.5m, quindi se ad esempio un giocatore effettua un movimento di 4 quadretti a forma di L. Il movimento costerà 7.5m. Nel nostro contesto siamo in una situazione chiamata "off-combat" in cui i personaggi non hanno un movimento massimo ma devono comunque calcolare il movimento totale utilizzato nei loro spostamenti. Il costo di un movimento varia in base alla tipologia del terreno:

- Terreno scomodo: Il movimento costa doppio
- Ostacoli da superare: Il movimento costa triplo

Queste regole servono a trasformare in costo di movimento azioni come "devo passare sotto il tavolo, quindi dovò muovermi di più" oppure "devo camminare nel pantano, mi servirà più sforzo per andare avanti". Queste regole verranno utilizzate per definire il costo del percorso definito dall'algoritmo.

1.2.2 Percezione

Senza entrare nei dettagli del funzionamento delle regole della percezione e furtività, queste possono essere riassunte in questo modo: Più un personaggio si avvicina al nemico, più sarà facile per questo percepirlo. Un ladro dovrà quindi cercare di tenersi il più lontano possibile dai nemici, e passarci vicino solo se le strade alternative gli farebbero perdere troppo tempo (sperando di essere abbasstanza bravo nella furtività da non essere percepito pur passando accanto al nemico). Ulteriore regola, un nemico non può percercire un personaggio se c'è un ostacolo (come un muro) che ostacola la vista, un ladro può quindi pensare di sfruttare questi elementi per muovesi all'interno della mappa. Queste regole verranno utilizzare per definire una funzione che maggiora il costo di un movimento, sulla base della probabilità di essere percepito.

2 Modello del problema

La tipologia di algoritmo scelto per trovare il percorso migliore è quella degli algoritmi genetici, abbiamo bisogno quindi di definire una funzine di costo associata ai possibili movimenti sulla mappa e una funzione di fitness che permetta di valutare la bontà di una possibile soluzione. Iniziamo formalizzando l'idea di mappa di gioco, percorso e funzione costo associata ad un percorso. Nella sezione successiva, questa verrà ulteriormente modificata per permettere all'algoritmo di convergere ad una soluzione ottimale.

2.1 Mappa e regole del gioco

Siano N,M dimensioni della mappa di gioco, ogni mappa conterrà i seguenti elementi:

(a) Elementi della mappa

Simbolo	**Significato**
#	Muro
O	Personaggio
Y	Obiettivo
Z	Nemico
X	Terreno scomodo
T	Ostacolo

(b) Esempio mappa

```
# # # # # # # # # # # #
#                 T   #
#                 T   #
#     #   Y   #       #
#     #       #       #
#   # #       # #     #
#     #       #       #
# X   # # # # #       #
# X X                 #
#         O           #
# # # # # # # # # # # #
```

(c) Esempio percorso

```
# # # # # # # # # # # #
#             ↓ ← T   #
#       ↓ ← ← ↑ ←     #
#     # → Y   # ↑ ← #
#     #       #   ↑ #
#   # #       # # ↑ #
#     #       # → ↑ #
# X   # # # # # ↑ ← #
# X X → → → → → → ↑ #
#     ↑ ← ←         #
# # # # # # # # # # # #
```

Un percorso sarà quindi rappresentato da una serie punti all'interno della mappa che sono stati attraversati dal personaggio. Un percorso deve soddisfare i seguenti vincoli:

- Un percorso non deve contenere le coordinate di un muro
- Un percorso non deve contenere le coordinate di un nemico

Sia quindi $Map = \{(i,j) \in \mathbb{N}^2 \mid 0 \leq i \leq N, 0 \leq j \leq M\}$ insieme delle posssibili coordinate associate ad un punto della mappa e sia f_{type} la funzione che associa ad ogni posizione la tipologia associata. Sia quindi $P = \{(x_1,y_1),(x_2,y_2)\dots(x_k,y_k)\} \subset Map$ un percorso di k punti. Si assume che il percorso sia continuo, cioè non contenga segmenti separati (la formulazione della soluzione dell'algoritmo genetico assicura l'impossibilità di generare percorsi spezzati). Posso ora formalizzare i vincoli con:

$$\forall (x,y) \in P \quad f_{type}(x,y) \neq \text{"}muro\text{"} \wedge f_{type}(x,y) \neq \text{"}nemico\text{"}$$

2.2 Costo associato al movimento

Il costo totale del movimento sarà la somma dei costi associati ad ogni quadretto della mappa. Definiamo una funzione che associa ad ogni coordinata il rispettivo costo:

$$f_{cost}(x,y) = \begin{cases} 1.5 & \text{se } f_{type}(x,y) = \text{"libero"} \\ 3 & \text{se } f_{type}(x,y) = \text{"scomodo"} \\ 4.5 & \text{se } f_{type}(x,y) = \text{"ostacolo"} \\ 0 & \text{altrimenti} \end{cases}$$

Prendendo quindi un percorso che rispetta i vincoli $P = \{(x_1,y_1),(x_2,y_2)\dots(x_k,y_k)\}$ la distanza totale del percorso è pari a:

$$\sum_{k}^{i=1} f_{cost}(x_i,y_i)$$

2.3 Costo associato ai nemici

Per tener conto contermporanteamente sia della distanza totale di un percorso, sia della probabilità di essere individuati da un nemico ho deciso di modellare la probabilità di essere percepito come una funzione di penalizzazione che va a moltiplicare il costo di ogni quadretto della mappa. In questo modo camminare nel territorio di un nemico costerà di più che camminare in un terriorio non pattugliato. L'idea generale è la seguente:

- Se un quadretto non è nella visuale del nemico (è presente un muro tra il quadretto e il nemico) il costo del quadretto rimane invariato
- Se un quadretto è nella visuale del nemico, il costo del quadretto sarà moltiplicato per un valore di penalizzazione che sarà tanto più grande quanto il quadretto è vicino al nemico.
- Se un quadretto è nella visuale del nemico, ma è abbastanza distante (distante almeno 6 quadretti) il costo resta invariato. Questo perchè ad una distanza di 6 quadetti il nemico non ha alcun bonus per percepire il PG.

Per calcolare il segmento che unisce due punti all'interno della mappa, viene utilizzata la **linea di Bresenham**, algoritmo di rasterizzazine della linea che connette due punti all'interno di una matrice. Si assume da ora in poi che i punti nella forma $a = (x,y) \in Map$ appartengano alla mappa. Sia quindi $p = (x,y)$ generico punto, $e = (x_e, y_e)$ posizione di un nemico e $L_{bh}(p,e) = \{(x_j,y_j)\}$ insieme dei punti appartenenti alla linea di Bresenham che unisce il punto (x,y) e (x_e,y_e) Allora possiamo definire la funzione distanza dal nemico nel seguente modo:

$$D_{enemy}(p,e) = \begin{cases} \infty & \text{se } \exists u \in L_{bh} \ni f_{type}(u) = \text{"muro"} \\ \#L_{bh}(p,e) & \text{altrimenti} \end{cases}$$

In questo modo abbiamo che la distanza da un quadretto al nemico è uguale all numero di quadretti tra il nemico e il quadretto secondo la linea di Bressenham (se la linea non contiene un muro, altrimenti abbiamo distanza ∞). Definiamo ora la funzione di penalizzazione basasata sul valore di distanza per una certa distanza $d \in \mathbb{N}$:

$$E_{penality}(d) = \begin{cases} 3 & \text{se } d = 1 \\ 3 - (d/3) & \text{altrimenti} \end{cases}$$

Possiamo quindi infire definire la funzione di penalizzazione associata al punto $p = (x,y)$ rispetto al nemico $e = (x_e, y_e)$ come:

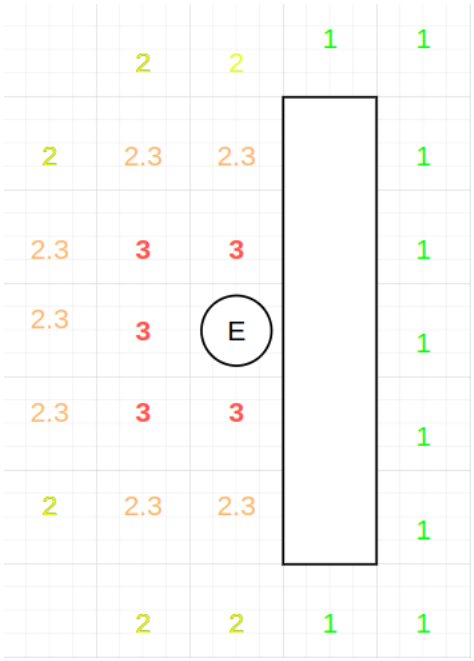
$$Penality(p,e) = E_{penality}(min(6, D_{enemy}(p,e))$$

Per fare maggiore chiarezza mostro la tabella dei risultati per ogni valore di distanza

(a) Funzione di penalizzazione

Distanza	Penalità
1	3
2	2.3
3	2
4	1.6
5	1.3
6	1
7	1
Infinite	1

(b) Esempio grafico



Con questa definizione abbiamo quindi una funzione crescente al diminuire della distanza, che non modifica il costo di un quadretto per ogni valore di distanza maggiore o uguale a 6 e che non modifica il costo di un quadretto se questo non è visibile al nemico. Quindi ad esempio, muoversi in un quadretto accanto al nemico, costerà 3 volte tanto il costo base del quadretto. Questa penalizzazione viene applicata per ogni nemico, quindi camminare in un quadretto accanto a due nemici costerà 6 volte il costo base.

2.4 Costo totale movimento penalizzato

Definiamo quindi il costo del movimento penalizzato (questa non è la fitness, ma una parte di essa, questa porzione tiene conto solo del costo del movimento all'interno della mappa di gioco). Sia quindi $P = \{(x_1, y_1), (x_2, y_2) \dots (x_k, y_k)\} = \{p_1, p_2 \dots p_k\}$ una soluzione che rispetta i vincoli e sia $E = \{e = (i, j) \in Map \mid f_{type}(e) = \text{"enemy"}\}$ insieme dei punti che rappresentano un nemico sulla mappa. Il costo totale del percorso sarà:

$$P_{cost} = \sum_{i=1}^{k} \sum_{e \in E} Penality(p_i, e) * f_{cost}(p_i)$$

3 Algoritmo genetico

3.1 Soluzione candidata

Definiamo una soluzione candidata come un insieme di movimenti $S = \{u, l, l, \dots, r, d\} \in \{u, l, r, d\}^k$ movimenti su, giù destra e sinistra all'interno della mappa di gioco. Ad ogni insieme di movimenti è possibile associare un percorso $P_S =$, un insieme di posizioni successive ottenute effettuando il movimento a partire da una posizione di partenza. Una soluzione candidata deve rispecchiare i seguenti vincoli:

- Una soluzione candidata non deve avere collisioni con muri o nemici
- Una soluzione candidata non deve poter effettuare backtracking (tornare sui propri passi, fare tre passi in avanti per poi tornare indietro)
- Una soluzione candidata non può continuare a fare movimenti se nel suo percorso ha già incontrato l'obiettivo
- Una soluzione candidata non può attraversare due volte lo stesso quadretto (non ci possono essere cicli all'interno del percorso)

Le soluzioni della popolazione iniziale verranno generate randomicamente, creando una sequenza randomica di movimenti, questo tipo di generazione (unito alle soluzioni generate tramite crossover o modificate tramite mutazione) possono generare delle soluzioni che non rispecchiano i vincoli imposti, vengono quindi applicati ulteriori due operatori di **Sanificazione** e **Estensione**.

3.1.1 Sanificazione delle soluzioni

Per sanificazione si intende il processo di rimozione da una soluzione di movimenti che non rispecchiano i vincoli (collisione e backtracking) e troncatura (raggiungimento dell'obiettivo, quadretto già visitato). Nello specifico:

- *Collisione*: Vengono rimossi da una soluzione tutti i movimenti che non modificano la posizione del personaggio (sbattere contro un muro, provare ad entrare nel quadretto occupato da un nemico)
- *Backtracking*: Vengono rimossi da una soluzione tutti i movimenti che portano il personaggio ad tornare indietro alla posizione precedente.
- *Obiettivo*: Se all'interno di una soluzione una mossa porta la posizione del personaggio ad essere uguale a quella dell'obiettivo, il resto della soluzione viene troncato.
- *Cicli*: Se all'interno di una soluzione, alla posizione k, non esitono mosse legali che permettono di incrementare la lunghezza della sequenza, il resto delle mosse viene troncato.

Questo operatore definisce i criteri di creazione delle mosse iniziali. Verrà inoltre applicato agli offspring dell'operazione di crossover e dopo un operatore di mutazione, per assicurare che la soluzione rispecchi tutti i vincoli. Si specifica che l'operatore di sanificazione non assicura che la soluzione sia ottimale, o che raggiunga l'obiettivo, ma "pulisce" una sequenza di mosse da eventuali mosse illegali.

3.1.2 Estensione di soluzioni

Ogni soluzione avrà una dimensione variabile, e non tutte raggiungeranno l'obiettivo fin dalla prima generazione. Alcune soluzioni potrebbero ad esempio bloccarsi in un vicolo cieco fin da subito. Le operazioni di crossover e mutazione potrebbero "sbloccare" queste soluzioni, creando degli offspring che potenzialmente possono andare avanti nel loro percorso. L'operatore di estensione, da a queste soluzioni "sbloccate" la possibilità di proseguire con mosse randomiche nel percorso, fino a raggiungere nuovamente un criterio di stop (la soluzione potrebbe di nuovo bloccarsi in un vicolo cieco).

Esempio di estensione In questo esempio viene mostrata una soluzione che ha come punto finale un "vicolo cieco" (nella generazione randomica, la soluzione è capitata con un punto finale che sbatte contro un muro), fortunatamente l'operatore di mutazione ha casualmente trasformato l'ultimo movimento in un movimento verso l'alto, "sbloccando" la soluzione e permettendole di crescere tramite l'operatore di estensione (sfortunatamente, dato che l'estensione è casuale, la soluzione è andata nuovamente a scontrarsi contro un muro).

(a) Soluzione base

```
# # # # # # # # # # # #
#                     #
#         # # #       #
# T   #       # Z   #
# T   #       # → → #
# T   #   Y   # ↑   #
# T Z #       # ↑   #
# T   # # # # # ↑ Z #
#         → → → ↑   #
#         ↑         #
# # # # # # # # # # # #
```

(b) Mutazione (posizione finale)

```
# # # # # # # # # # # #
#                     #
#         # # #       #
# T   #       # Z   #
# T   #       # → ↑ #
# T   #   Y   # ↑   #
# T Z #       # ↑   #
# T   # # # # # ↑ Z #
#         → → → ↑   #
#         ↑         #
# # # # # # # # # # # #
```

(c) Estensione

```
# # # # # # # # # # # #
#                   ↑ #
#         # # #     ↑ #
# T   #       # Z ↑ #
# T   #       # → ↑ #
# T   #   Y   # ↑   #
# T Z #       # ↑   #
# T   # # # # # ↑ Z #
#         → → → ↑   #
#         ↑         #
# # # # # # # # # # # #
```

3.2 Fitness

La funzione di fitness permette di valutare la bontà di una soluzione, questa infatti sarà proprio la misura utilizzata dal modello per selezionare le soluzioni migliori nella popolazione. Da ora in poi si suppone che ogni soluzione sia stata sanificata e estesa. Sia quindi P_{cost} il costo del percorso P associato alla soluzione S, sia p_{start} posizione di partenza del personaggio e sia p_{obj} posizione dell'obiettivo del personaggio e p_k posizione finale del personaggio dopo aver effettuato tutti i movimenti. La funzione di fitness di basa sulla seguente idea:

- Una soluzione che raggiunge l'obiettivo è migliore di una soluzione che non lo raggiunge
- Una soluzione ferma in un vicolo cieco dovrebbe essere penalizzata
- Una soluzione troppo vicina al punto di partenza dovrebbe essere penalizzata
- La soluzione dovrebbe essere penalizzata in base alla distanza dall'obiettivo

Definiamo quindi una misura di distanza rispetto alla partenza e all'obiettivo, verrà usata la distanza di Manhattan (Norma 1) moltiplicata per 1.5, in quanto è la misura più vicina a simulare un movimento quadretto per quadretto all'interno della mappa.

$$d_{start} = \|p_{start} - p_k\|_1 * 1.5 \quad d_{obj} = \|p_{obj} - p_k\|_1 * 1.5$$

Definiamo inoltre una funzione di penalizzazione sulla base del raggiungimento o meno dell'obiettivo, questa funzione ci permetterà di "preferire" soluzioni non ottimali che raggiungono l'obiettivo, a soluzioni molto vicine all'obiettivo ma che non lo raggiungono (ad esempio se l'obiettivo è subito dopo un muro e una soluzione continua a "sbattere" contro il muro).

$$O_{penality}(p_k, p_{obj}) = \begin{cases} -10 & \text{se } p_k = p_{obj} \\ +20 & \text{altrimenti} \end{cases}$$

Definiamo quindi i seguenti iperparametri:

- δ_d: Peso associato alla distanza della soluzione
- δ_s: Peso associato alla distanza rispetto alla posizione iniziale
- δ_o: Peso associato alla distanza rispetto all'obiettivo
- δ_b: Peso associato al bonus assegnato in base alla posisizone finale

Definiamo la funzione di fitness finale associata al percorso P relativo alla soluzione S come:

$$Fitness(S) = \delta_d P_{cost} + \delta_s d_{start} - \delta_o d_{obj} + \delta_b O_{penality}$$

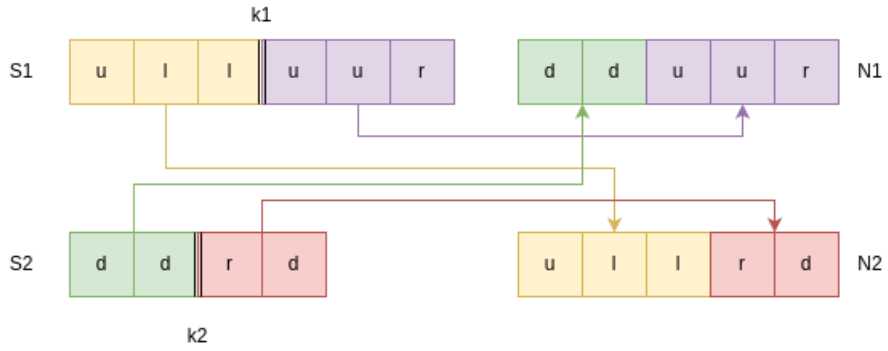
3.3 Tournament Selection

La tipologia di selezione scelta è la selezione tramite torneo, che consiste nel considerare *tourn_size* numero di soluzioni distinte che partecipano ad un torneo, in un torneo verrà selezionata la soluzione con il fit migliore con probabilità p, la seconda soluzione migliore con probabilità $p(1-p)$ e la k+1-esima soluzione migliore con probabilità $p(1-p)^k$. Vengono quindi definiti ulteriori iperparametri:

- *pop_size*: Grandezza della popolazione
- *selec_prob*: Probabilità di selezione della soluzione con miglior fitness
- *tourn_size*: Grandezza del sottoisiemi di soluzioni selezionata per il torneo

3.4 One Point Crossover

L'operazione di crossover sceltà è il one point crossover, avendo soluzioni di grandezza differente verranno scelti due cut-point randomici e verrano generati due nuovi offspring scambiando le rispettive metà di ogni soluzione:



Questa tipologia di crossover può generare soluzioni che non rispettano i vincoli del problema (anche nell'esempio in foto, la soluzione N1 non rispetta i vincoli avendo un movimento su, seguito da un movimento giu) vengono quindi applicati gli operatori di sanificazione per la rimozione delle mosse illegali, e di estensinoe per permettere alla soluzione di proseguire nel suo percorso se ne ha la possibilità. Unico iperparametro per il crossover è *cross_prob* probabilità di applicare l'operatore.

3.5 Mutazione

La mutazione in questo dominio trasforma un movimento all'interno di una soluzione in un altro movimento casuale (diverso da quello di partenza). Questa operazione è risultata molto utile per portare soluzioni bloccate in dei vicoli ciechi a cambiare posizione ed ottenere la possibilità di essere nuovamente estese. La probabilità di mutazione *mutation_prob* non avrà un valore fisso ma varierà in base alla generazione corrente, in modo da:

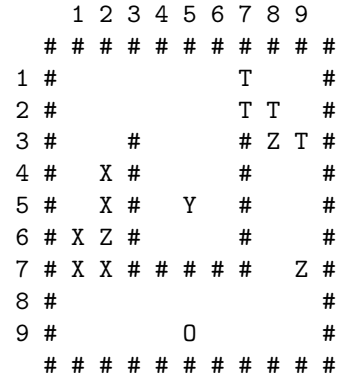
- Favorire l'esplorazione nelle fasi iniziali dell'algoritmo
- Non modificare troppo le soluzioni nelle generazioni più avanzate

In generale è stato deciso di ridurre di un valore *mutation_red* la probabilità di mutazione ogni 20 generazioni.

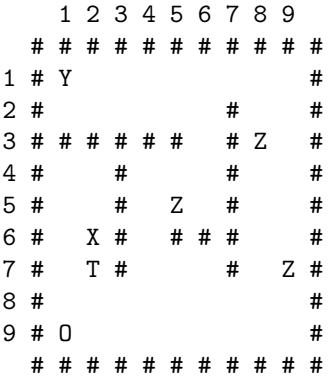
4 Risultati sperimentali

Il modello è stato testato su diverse mappe per valutare le prestazione in diversi contesti, verranno presentate tre mappe che mostrano i risultati salienti in diverse situazioni, per ogni mappa verranno indicati i risultati sperati e le strategie migliori che il modello dovrebbe mettere in atto per raggiungere la soluzione ottimale. Ogni mappa avrà le stesse dimensioni 9x9, questa risulta infatti essere una dimensione stereotipica di una stanza di un dungeon di Pathfinder RPG.

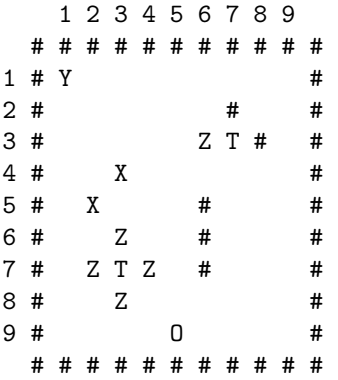
(a) Mappa 1



(b) Mappa 2



(c) Mappa 3



Per ogni mappa verranno usati i seguenti valori degli iperparametri di base:

Parametro	**Valore**
pop_size	50
selec_prob	0.80
tourn_size	5
cross_prob	1
mutation_prob	0.15 (decresce di 0.05 ogni 20 generazioni)
max_generations	100
δ_d	1
δ_s	5
δ_o	2
δ_b	1

4.1 Mappa 1: Il modello sceglie i percorsi giusti?

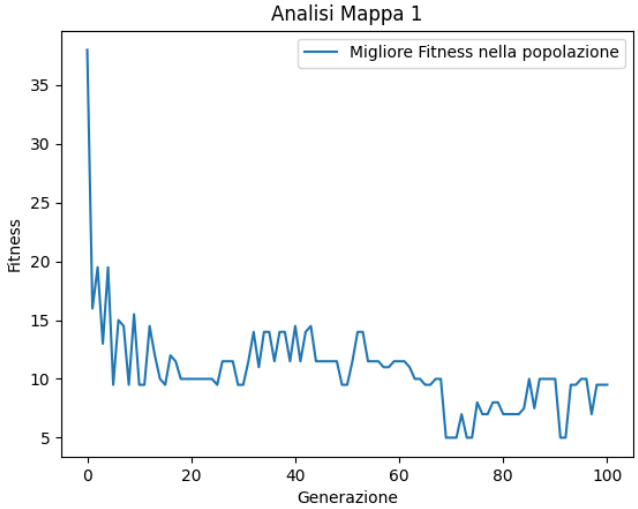
L'obiettivo della Mappa 1 è assicurarsi che il modello sia in grado di scegliere la strada corretta quando posto davanti a due strade con distanza fisica simile, ma con diverso numero di nemici.

(a) Migliore Generazione 0

```
Legal:     True
Complete:  True
Fitness:   42.0
    1 2 3 4 5 6 7 8 9
  # # # # # # # # # # #
1 # → ↓ → ↓     T     # 1
2 # ↑ → ↑ ↓     T T   # 2
3 # ↑ ← # ↓     # Z T # 3
4 # → ↑ # → ↓   #     # 4
5 # ↑ X #   Y   #     # 5
6 # ↑ Z #       #     # 6
7 # ↑ X # # # # #   Z # 7
8 # ↑ ↓ ← ↓ ← ← ←     # 8
9 # ↑ ← ↑ ← → → ↑     # 9
  # # # # # # # # # # #
    1 2 3 4 5 6 7 8 9
```

(b) Migliore Generazione 100

```
Legal:     True
Complete:  True
Fitness:   10.5
    1 2 3 4 5 6 7 8 9
  # # # # # # # # # # #
1 # → → ↓       T     # 1
2 # ↑   → → ↓   T T   # 2
3 # ↑   #   ↓   # Z T # 3
4 # ↑ X #   ↓   #     # 4
5 # ↑ X #   Y   #     # 5
6 # ↑ Z #       #     # 6
7 # ↑ X # # # # #   Z # 7
8 # ↑                 # 8
9 # ↑ ← ← ← ←         # 9
  # # # # # # # # # # #
    1 2 3 4 5 6 7 8 9
```



La soluzione trovata dall'algoritmo rispecchia le caratteristiche di un buon percorso, il modello infatti evita il percorso di destra, che penalizza molto la fitness a causa della presenza di molti nemici e degli ostacoli. Nel percorso scelto evita inoltre il più possibile il terreno scomodo e nella parte superiore della mappa, preferisce allungare un po' il percorso (allontandandosi dal nemico e diminuendo le probabilità di essere scoperto), prima di avvicinarsi all'obbiettivo, questo tipo di stragia è una dinamica colune nelle sessioni di combattimento di Pathfinder RPG.

4.2 Mappa 2: Il modello è in grado di superare minimi locali?

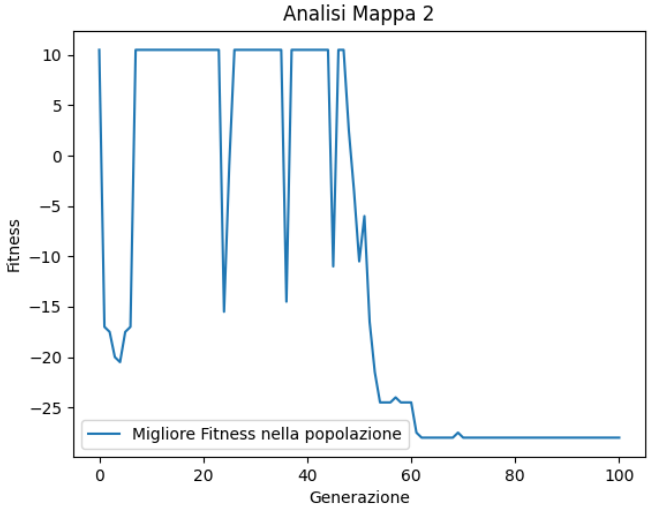
La Mappa 2 mira a valutare il seguente fattore nella soluzione finale del modello: In presenza di percorsi bloccati (percorso di sinistra) molto vicini alla soluzione, quindi con una fitness relativamente bassa, il modello riesce comunque a trovare una soluzione ottimale che raggiunge l'obiettivo superando il minimo locale?

(a) Migliore Generazione 0

```
Legal:     True
Complete:  False
Fitness:   10.5
    1 2 3 4 5 6 7 8 9
  # # # # # # # # # # #
1 # Y                 # 1
2 #             #     # 2
3 # # # # # # #   # Z   # 3
4 # →   #       #     # 4
5 # ↑ ← #   Z   #     # 5
6 # → ↑ #   # # #     # 6
7 # ↑ T #       #   Z # 7
8 # ↑                 # 8
9 # ↑                 # 9
  # # # # # # # # # # #
    1 2 3 4 5 6 7 8 9
```

(b) Migliore Generazione 100

```
Legal:     True
Complete:  True
Fitness:   -28.0
    1 2 3 4 5 6 7 8 9
  # # # # # # # # # # #
1 # Y                 # 1
2 # ↑ ← ← ← ← ← #     # 2
3 # # # # # # # ↑ # Z   # 3
4 #     # → → ↑ #     # 4
5 #     # ↑ Z   #     # 5
6 #   X # ↑ # # #     # 6
7 #   T # ↑     #   Z # 7
8 #   → → ↑             # 8
9 # → ↑                 # 9
  # # # # # # # # # # #
    1 2 3 4 5 6 7 8 9
```



Notiamo infatti che la soluzione migliore alla prima generazione si trova proprio nel punto di minimo locale, questo punto infatti è molto vicino alla soluzione, e porta la fitness a ridursi molto, rendendo diffile trovare una soluzione ottimale fin dalle prime generazioni. Notiamo infatti che rispetto alla mappa precendete la fitness della soluzione migliore tende a oscillare molto fino a molto avanti nelle generazioni, fino a stabilizzarsi intorno alla generazione 70.

4.3 Mappa 3: Il modello è in grado di sfruttare le coperture?

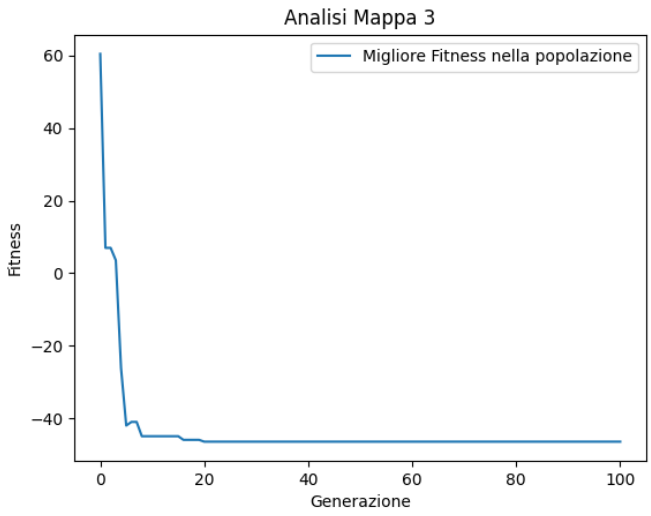
La mappa 3 valuta il fattore più importante nel movimento furtivo di Pathfinder: Il modello è in grado di sfruttare l'ambiente e le coperture presenti nella mappa, per raggiungere l'obiettivo senza essere scoperto? La mappa infatti è presa direttamente da una partita giocata recentemente dal mio gruppo di gioco, presenta in fatti una taverna, con un tavolo di 4 guardie, un separè centrale, e una colonna ad angolo con un ulteriore tavolo occupato da una guardia.

(a) Migliore Generazione 0

```
Legal:     True
Complete:  False
Fitness:   67.0
    1 2 3 4 5 6 7 8 9
  # # # # # # # # # # #
1 # Y                 # 1
2 #             #     # 2
3 #           Z T #   # 3
4 #     X             # 4
5 #   X       #       # 5
6 #     Z     # ↓ ←   # 6
7 #   Z T Z   #   ↑   # 7
8 #     Z       → ↑   # 8
9 #         → → ↑     # 9
  # # # # # # # # # # #
    1 2 3 4 5 6 7 8 9
```

(b) Migliore Generazione 100

```
Legal:     True
Complete:  True
Fitness:   -46.44444444444444
    1 2 3 4 5 6 7 8 9
  # # # # # # # # # # #
1 # Y ← ← ← ← ← ← ← ← # 1
2 #               #   ↑ # 2
3 #           Z T # ↑ # 3
4 #     X           ↑ # 4
5 #   X       #     ↑ # 5
6 #     Z     #     ↑ # 6
7 #   Z T Z   #     ↑ # 7
8 #     Z           ↑ # 8
9 #         → → → → ↑ # 9
  # # # # # # # # # # #
    1 2 3 4 5 6 7 8 9
```



Il modello riesce correttamente a sfruttare l'ambiente per effettuare un percorso che per la maggior parte lo rende invisibile ai nemici. Spostarsi prima nell'angolo in basso a destra permette al modello di rendersi invisibile alla guardia presente al tavolo singolo, proseguire dritto permette poi di restare inosservato fino all'angolo in alto a destra.

5 Conclusioni e sviluppi futuri

Il modello ha performato in maniera ottimale sulle mappe presentate, ulteriori test su altre mappe hanno mostrato risultati simili a quelli ottenuti, le tre mappe presentate permettono però di sottolineare gli aspetti cruciali dell'algoritmo di ottimizzazione e rispondere alle domande di ricerca. Voglio sottolineare che l'algoritmo non rispecchia perfettamente la realtà della furtività in pathfinder, questa infatti consiste in una "prova contrapposta" tra la furtività del personaggio e la percenzione del nemico, i nemici hanno un bonus in questa prova in base alla distanza del personaggio, questo bonus è stato modellato nel problema come penalizzazione crescente più il personaggio si avvicina al nemico, ma, volendo rendere l'algoritmo specializzato su uno specifico personaggio, si potrebbe basare la funzione di penalizzazione anche sull'abilità di furtività del personaggio (non possiamo tener conto dell'abilità di percezione del nemico dato che questa non è nota ai personaggi ma solo al DM). In questo modo :

- Personaggi con alta furtività potrebbero preferire percorsi che passano in mezzo ai nemici, essendo sicuri di non essere individuati in ogni caso
- Personaggi con bassa furtività potrebbero preferire percorsi più "sicuri" evitando il più possibile i nemici.