Sl-AI-er

ALESSANDRO CATURANO

ACM Reference Format:
ALESSANDRO CATURANO. 2024. Sl-AI-er. 1, 1 (January 2024), 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Contents

Author's address: ALESSANDRO CATURANO.

XXXX-XXXX/2024/1-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUZIONE

Il progetto "Sl-AI-er" mira allo sviluppo di un agente intelligente in grado di vincere livelli del gioco del minigolf, implementato tramite il game engine "Unity". Più nello specifico si utilizza un algoritmo genetico, implementato nel linguaggio C#, per trovare soluzioni da migliorare ad ogni iterazione fino al raggiungimento di una configurazione di parametri che consenta all'IA di centrare la buca con la pallina.

I parametri in questione sono i seguenti:

- angolo: determina l'angolazione, in gradi, con cui sarà colpita la pallina;
- attesa: determina i secondi da attendere prima di colpire la pallina (utile vista la presenza nel gioco di ostacoli in movimento);
- potenza: determina la potenza da applicare alla pallina.

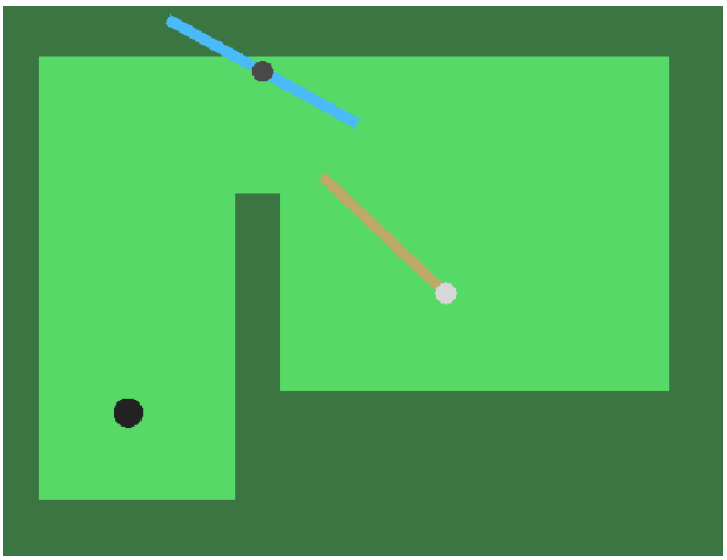

Fig. 1. Nell'immagine è riportato il Livello 1 del gioco. Si farà ulteriore riferimento a tale livello in seguito.

2 PERCHÉ L'ALGORITMO GENETICO?

Trovare la giusta configurazione di parametri che consenta di vincere un livello di minigolf è di per sé nu problema relativamente complesso. Una possibile alternativa sarebbe stata addestrare un modello di machine learning, tuttavia prima di tutto sarebbe necessario addestrarlo su numerosi livelli affinchè potesse essere efficace. Inoltre, poiché i livelli sono potenzialmente infiniti, l'agente avrebbe potuto riscontrare difficoltà nel risolverne di particolarmente complessi o diversi da quelli utilizzati nel training set.

Una soluzione più versatile è invece l'algoritmo genetico: questo parte da 0 per poi cercare passo dopo passo una soluzione, senza essere vincolato al livello del gioco in cui agisce. Supponiamo inoltre di voler suddividere i vari livelli del gioco in base a un indicatore di difficoltà: tramite algoritmo genetico si sarebbe in grado di classificare automaticamente i livelli, ad esempio valutandone la difficoltà sulla base del numero medio di generazioni necessarie per trovare una soluzione.

3 DESCRIZIONE DELL'AMBIENTE

Come detto, l'ambiente in cui l'agente opera è una simulazione in "Unity" del gioco del minigolf, dove il giocatore può regolare angolazione e forza con cui colpire la pallina, al momento desiderato: se questa non centra la buca il livello ricomincia da 0, in caso contrario è considerato vinto.

Gli indicatori PEAS che descrivono tale situazione sono i seguenti:

- Performance: la performance dell'agente viene valutata sulla base di quanto la pallina si avvicina alla buca durante un'iterazione;
- Environment: l'ambiente è costituito da:
  - Una pallina che il giocatore deve colpire
  - Una buca che la pallina deve centrare per superare il livello
  - Pareti che delimitano l'area di gioco e sulle quali la pallina può rimbalzare
  - Ostacoli in movimento;
- Actuators: l'agente può agire modificando l'ambiente tramite la regolazione dei tre parametri (angolazione, potenza, attesa): fatto ciò sarà in grado di avviare la simulazione;
- Sensors: quando il movimento della pallina colpita si arresta, l'agente intelligente ottiene informazioni sulla distanza tra pallina e buca, espressa in unità di distanza di Unity, e sull'eventuale presenza di pareti che si frappongono tra i due oggetti.

In generale, possiamo definire l'ambiente come:

- Parzialmente osservabile: l'agente conosce solo la misura di prestazione relativa alla popolazione presa in considerazione durante un'iterazione.
- Deterministico: lo stato successivo del gioco sarà determinato esclusivamente dalla prossima azione dell'agente.
- Episodico: potremmo considerare ogni tentativo dell'agente di fare buca come un episodio. D'altronde, l'IA non eseguirà altre azioni finchè la pallina non si sarà fermata.
- Dinamico: tranne nel caso di livelli privi di ostcoli in movimento, l'ambiente varia nel tempo.
- Continuo: si tratta di una simulazione di un ambiente tridimensionale in cui, ad esempio, le angolazioni con cui si può colpire la palla sono infinite, così come i percorsi che questa può seguire e i punti in cui si può fermare.
- Singolo: dal momento che non vi sono altri agenti, l'ambiente è ad agente singolo.

4 IMPLEMENTAZIONE

Nella seguente sezione si riporta il processo che ha portato all'implementazione e al progressivo miglioramento dell'algoritmo genetico.

4.1 Versione iniziale

Una prima implementazione dell'algoritmo genetico presenta caratteristiche implementative generiche e prende numerose decisioni basate sulla casualità. Si noti che questa è solo una versione iniziale dell'algoritmo, e ne fungerà da base sulla quale operare per ottenere prestazioni migliori. Ad ogni modo, vale comunque la pena osservarne il comportamento per avere una misura concreta dei successivi progressi.

- Codifica individuo: un individuo è caratterizzato dai tre parametri precedentemente descritti: angolazione, potenza, attesa;
- Dimensioni popolazione: Inizialmente, la popolazione presenta 4 individui, la mating pool avrà le stesse dimensioni;

- Funzione di fitness: la funzione di fitness iniziale è inversamente proporzionale alla distanza a cui la pallina si ferma dalla buca, e si ottiene tramite la seguente formula:

$$F = 10/d$$

  dove d è la distanza pallina-buca e 10 è un valore di normalizzazione;
- Popolazione iniziale: la popolazione iniziale è caratterizzata da 4 individui di cui angolazione, potenza e attesa sono determinati casualmente;
- Selezione: si utilizza il metodo "roulette wheel": ogni individuo ha una probabilità di essere selezionato nuovamente proporzionale alla sua fitness;
- Crossover: Si itera su tutta la popolazione scambiando angolazione, potenza o attesa (il gene è scelto casualmente) per ogni coppia di individui;
- Mutazione: Si seleziona un parametro casuale: questo viene variato di una quantità casuale compresa tra due estremi:
  - angolazione: $\pm 10°$
  - potenza: ± 0.3 (dove la potenza è un valore compreso tra 0 e 2)
  - attesa: $\pm 0.5s$ (su un massimo di 4s);

Vediamo un esempio di esecuzione di questo algoritmo sul Livello 1. La seguente tabella riporta gli individui della popolazione iniziale a cui è stato assegnato, al termine della simulazione, un valore di fitness ottenuto tramite la formula riportata sopra:

	angolo	attesa	potenza	fitness
Individuo 0	101,2013	3,041242	0,802838	0,8523427
Individuo 1	305,3118	3,329434	0,2638977	1,181628
Individuo 2	56,78854	1,450229	1,985107	0,9013949
Individuo 3	255,748	3,575829	1,489817	5,413477

La tabella successiva riporta la popolazione dopo che è stata eseguita la selezione: come avremmo potuto intuire dall'elevato valore di fitness, l'individuo 3 nella tabella precedente è stato selezionato numerose volte.

	angolo	attesa	potenza	fitness
Individuo 0	255,748	3,575829	1,489817	5,413477
Individuo 1	255,748	3,575829	1,489817	5,413477
Individuo 2	56,78854	1,450229	1,985107	0,9013949
Individuo 3	255,748	3,575829	1,489817	5,413477

Per completezza si riporta la popolazione dopo l'esecuzione di crossover e mutazione:

	angolo	attesa	potenza
Individuo 0	255,748	3,128424	1,489817
Individuo 1	260,3835	3,575829	1,489817
Individuo 2	248,8245	1,450229	1,985107
Individuo 3	56,78854	3,672463	1,489817

Nel caso riportato, l'algoritmo è stato in grado di trovare un individuo con una fitness elevata già nella generazione iniziale (considerata generazione 0). Di conseguenza, è stata trovata una soluzione appena alla generazione 1, ovvero l'individuo 1 nell'ultima tabella.

In molti altri casi tuttavia l'algoritmo raggiunge una convergenza prematuramente, bloccandosi su un singolo individuo corrispondente a una soluzione sub-ottimale. Di conseguenza, non è in grado di centrare la buca con la pallina.

4.2 Miglioramenti

Sfruttando la conoscenza sul problema, si possono implementare numerosi miglioramenti al fine di migliorare l'efficacia e l'efficienza dell'algoritmo.

Prima di tutto, per risolvere il problema della convergenza prematura che porta l'algoritmo a bloccarsi su configurazioni sub-ottime, è stato aggiunto un limite massimo di generazioni come stopping condition (pari a 12): se questo viene superato la soluzione viene considerata sub-ottima e pertanto non soddisfacente: non consente di vincere il livello! In questi casi, l'algoritmo ricomincia da capo re-iniziallizzando la popolazione.

Allo stato attuale, le prestazioni dell'algoritmo sono mediocri: su 50 soluzioni trovate nel Livello 1, 24 sono sub-ottime mentre le soluzioni ottime hanno richiesto in media 3,96 generazioni per essere trovate. In altre parole nel 52% dei casi l'algoritmo è stato in grado di fare buca in molto meno di 12 generazioni: appunto, 3,96 in media. Si potrebbe dunque pensare di diminuire il limite di generazioni, ma per il momento si preferisce concentrarsi sul migliorare l'effettivo algoritmo.

Seguono i miglioramenti apportati successivamente:

- Il fatto che gran parte delle soluzioni siano trovate in poche generazioni significa che la popolazione iniziale gioca un ruolo chiave: vale quindi la pena aumentarne le dimensioni, da 4 a 6 ad esempio (bisogna ricordare che trattandosi di una simulazione in tempo reale, aumentare troppo le dimensioni della popolazione equivarrebbe ad aumentare eccessivamente il tempo di elaborazione!).
- Inoltre, invece di generare angolazioni casuali, si possono inizializzare 6 direzioni tali che l'i-esimo individuo abbia un'angolazione compresa tra 60*i e 60*(i+1) gradi per aumentare la diversità ed esplorare maggiormente lo spazio delle soluzioni. Ciò è osservabile nella seguente tabella:

	angolo	attesa	potenza	fitness
Individuo 0	25,33942	0,7645559	1,417887	0,8028
Individuo 1	103,8874	0,7958865	1,972853	0,7588924
Individuo 2	152,3891	1,113277	0,4605322	1,61378
Individuo 3	193,6266	1,72595	1,247374	1,028153
Individuo 4	279,1908	3,454127	1,392084	0,8170456
Individuo 5	302,5711	0,5300827	1,592986	1,338265

- Per quanto riguarda la fitness, si potrebbe calcolare la distanza del percorso più breve tra palla e buca invece di utilizzare la distanza in linea d'area. Questo presuppone tuttavia l'impiego di algoritmi di ricerca, ad esempio A*. Si è deciso invece di applicare una soluzione più semplice che consiste nel penalizzare la fitness (sempre calcolata in base alla distanza in linea d'area) dividendola per 2, nel caso in cui vi siano pareti tra la pallina e la buca. In questo modo, non si ricorre ad una funzione multi-obbiettivo.
- In fine, si è pensato di introdurre una strategia di elitismo: ad ogni iterazione viene salvata la soluzione migliore, denominata best-fitting. Nella generazione successiva questa viene confrontata con la soluzione peggiore: se la fitness di quest'ultima è minore, best-fitting la sostituirà.

4.3 Versione finale

Avviando la simulazione del Livello 1, con lo stesso limite di 12 alle generazioni massime, le prestazioni dell'algoritmo risultano notevolmente migliorate: su 50 soluzioni, una sola è sub-ottima e il numero medio di generazioni necessarie per trovare una soluzione scende a 2,24. In altre parole le soluzioni ottimali trovate salgono all'98% (dal 52%) e la media delle generazioni scende di 1,72.

Abbiamo notato, e riteniamo interessante condividerlo, che l'elitismo ha avuto un grosso impatto sulle prestazioni: implementando soltanto gli altri miglioramenti la media di generazioni necessarie era di 2,7 e le soluzioni ottime erano 37 su 50, il 74%.

5 CONCLUSIONI

L'algoritmo implementato potrebbe ulteriormente essere migliorato, ad esempio migliorando la strategia di selezione, cambiando le probabilità che gli operatori genetici vengano applicati o implementare una stopping condition che valuta quando fermarsi in base ai miglioramenti ottenuti nelle ultime generazioni.

Tuttavia, i risultati sin qui ottenuti sono stati ritenuti soddisfacenti in relazione al problema: l'agente è in grado di vincere un livello del minigolf in relativamente poche generazioni. Pertanto si è deciso di considerare la versione attuale dell'algoritmo come definitiva.