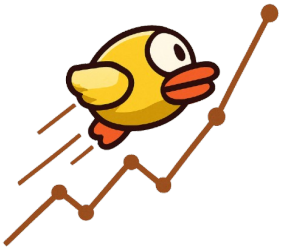


FlappyAI

Progetto di Fondamenti di Intelligenza Artificiale

Università degli Studi di Salerno

Anno Accademico 2024/2025

Carmelo Cappiello, Antonio Ceruso

Abstract

FlappyAI è un progetto il cui obiettivo è progettare e implementare due diverse soluzioni di Intelligenza Artificiale in grado di giocare autonomamente al noto gioco **Flappy Bird**. Le due soluzioni si distinguono per l'approccio adottato:

- La prima utilizza un algoritmo di ricerca locale classico, in particolare l'algoritmo di *Hill Climbing*.
- La seconda si basa su tecniche di Reinforcement Learning, e più nello specifico su modelli di *Deep Learning*.

Nel presente documento verranno descritte in dettaglio le metodologie e le tecniche impiegate nello sviluppo del progetto, con un focus particolare sugli aspetti ingegneristici. In particolare, verranno approfonditi il modello **CRISP-DM** e le **specifiche PEAS**, utilizzati per strutturare e guidare l'intero processo di progettazione delle due intelligenze artificiali.

1 Introduzione

FlappyAI nasce dalla curiosità di vedere come due approcci di training differenti permettano a un'intelligenza artificiale di giocare in modo perfetto a un videogioco: **Flappy Bird**.
Il codice sorgente del progetto è disponibile alla seguente repository Github:
`https://github.com/HEROCRA/FlappyBirdAI`

1.1 Cos'è Flappy Bird

Flappy Bird è uno dei giochi più storici, famosi e iconici dello scorso decennio. Consiste in un gioco a scorrimento orizzontale, dove lo scopo è fare più punti. L'uccellino dovrà attraversare dei tubi senza scontrarsi con essi; ogni tubo che attraversa corrisponde a un punto ottenuto, l'obiettivo è accumulare quanti più punti possibili. Se si scontra con un tubo o con i bordi del gioco, la partita termina.

2 Approccio classico con Hill Climbing

L'algoritmo Hill Climbing è un tipico algoritmo di ricerca locale che si basa fortemente sulla **struttura dei vicini (neighborhood)**. Questo significa che, una volta individuata una soluzione al problema, l'algoritmo la confronterà con le soluzioni vicine con l'obiettivo di trovarne una migliore. Nel caso in cui una soluzione migliore venga trovata, l'algoritmo si sposterà verso di essa e ripeterà nuovamente il processo.
Tuttavia, questo algoritmo non garantisce il raggiungimento della soluzione globalmente ottima, poiché tende a esplorare soluzioni che portano a risultati solo localmente ottimi. Questo comporta il rischio di rimanere intrappolati in **massimi/minimi locali** o di non riuscire a superare ostacoli della funzione di valutazione, come *ridges* o *plateau* , che possono ostacolare la ricerca di soluzioni migliori.

2.1 Specifica PEAS

- **Performance:**
  - Numero di tubi superati (*score*).
  - Massimizzazione della sopravvivenza dell'uccello.
  - Penalizzazione elevata in caso di morte (valutazione: -1000).
  - Visualizzazione dei punteggi nel tempo tramite grafico in tempo reale.
- **Environment:**
  - Ambiente simulato in `Pygame`.
  - Componenti: uccello, tubi (generati proceduralmente e con caratteristiche casuali), terreno.
  - Nessuna interazione umana; controllo interamente autonomo.
- **Actuators:**
  - Azione `[1]`: *flap* (salto verso l'alto).
  - Azione `[0]`: nessuna azione (gravità agisce sull'uccello).
  - Un'azione è eseguita per ciclo di gioco.
- **Sensors:**
  - Posizione verticale dell'uccello (`rect.y`).

  - Velocità verticale dell'uccello.
  - Posizione e altezza dei tubi.
  - Stato di collisione (con tubi o terreno).

2.2 CRISP-DM: Sviluppo Hill Climbing

2.3 1. Business Understanding

L'obiettivo del progetto è realizzare un agente autonomo in grado di giocare a **Flappy Bird** e superare quanti più ostacoli possibile, massimizzando il punteggio finale. L'approccio scelto è la **ricerca locale Hill Climbing**, per confrontare le prestazioni con tecniche più complesse come il Deep Learning in fasi successive.

2.4 2. Data Understanding

Il sistema non utilizza un dataset predefinito ma genera dati in tempo reale attraverso simulazioni autonome. I dati utili per prendere decisioni includono:

- Altezza attuale dell'uccello.
- Distanza verticale dal centro del gap tra i tubi.
- Posizione orizzontale dei tubi rispetto all'uccello.
- Stato dell'uccello (in volo, in collisione, a terra).

2.5 3. Data Preparation

La preparazione dei dati è implicita nella simulazione:

- I dati sensoriali sono letti direttamente dagli oggetti Pygame.
- La funzione `evaluate_state()` valuta lo stato corrente dell'uccello, penalizzandolo ogni qualvolta la sua posizione verticale si allontana dal centro del gap.
- Nessuna normalizzazione o preprocessamento necessario.

2.6 4. Modeling

L'agente adotta una strategia di **Hill Climbing a orizzonte simulato**:

- Per ogni ciclo di gioco, vengono simulate due alternative: `flap` o `no flap`.
- Ogni simulazione viene eseguita per 35 passi virtuali.
- La simulazione che porta al miglior stato finale (valutato con `evaluate_state()`) determina l'azione reale.
- Lo stato viene clonato per evitare effetti collaterali sulla simulazione.

2.7 5. Evaluation

Il sistema viene valutato attraverso:

- **Punteggio** medio per partita.
- **Massimo punteggio** ottenuto.
- **Distribuzione dei punteggi**.
- Robustezza e consistenza tra più run.

2.8 6. Deployment

Attualmente il progetto è in fase sperimentale, ma include:

- Simulazione automatica di partite.
- Grafico in tempo reale delle performance.
- Interfaccia visiva con grafica accattivante.

3 Modello: `Hill_Climbing_Flappy`

Il modello `Hill Climbing` è un agente che utilizza un approccio di ricerca locale per ottimizzare le azioni in un contesto di gioco. In questo caso, l'agente gioca al gioco "Flappy Bird" cercando di massimizzare il punteggio mediante una strategia di ricerca del miglior passo da compiere in un dato orizzonte temporale.

Architettura

Il modello non si basa su una rete neurale, ma piuttosto su una strategia di hill climbing che valuta lo stato del gioco attraverso la funzione `evaluate_state`, quindi seleziona l'azione migliore (saltare o non saltare) per ottimizzare il punteggio. La procedura è la seguente:

- Per ogni possibile azione (0 = non saltare, 1 = saltare), il gioco viene "clonato" e simulato per un numero di passi (`horizon`).
- Ogni simulazione è valutata utilizzando la funzione di valutazione dello stato che tiene conto della distanza tra l'uccello e il centro del gap tra i tubi.
- L'azione che porta alla valutazione migliore (ovvero alla posizione ottimale dell'uccello) viene scelta come l'azione successiva.

Funzioni Principali

- `reset()`: Inizializza il gioco, ripristinando le variabili di stato e il punteggio.
- `choose_action(horizon)`: Simula l'effetto di ciascuna possibile azione per un dato orizzonte temporale e sceglie quella che ottimizza il punteggio.
- `evaluate_state(game_state)`: Calcola una valutazione dello stato del gioco basata sulla distanza tra l'uccello e il gap tra i tubi.
- `game_step(action)`: Esegue un passo del gioco, aggiornando la posizione dell'uccello, dei tubi e del terreno, e controllando la fine del gioco.
- `spawn_pipe()`: Genera nuovi tubi a intervalli regolari.

Per ciascuna possibile azione, il modello simula il gioco fino a un orizzonte definito (`horizon`), in cui l'azione selezionata viene applicata nel primo passo e le azioni successive sono mantenute neutre (non flap). Il gioco viene eseguito in parallelo per entrambe le azioni, e la funzione di valutazione misura la distanza dell'uccello dal centro del gap tra i tubi. L'azione che porta al punteggio migliore è scelta come quella da eseguire.

Input del Modello

Il modello si basa sui seguenti stati e variabili:

- Posizione verticale dell'uccello
- Velocità dell'uccello
- Stato precedente della velocità dell'uccello
- Se l'uccello sta sbattendo le ali
- Posizione dei tubi (altezza e distanza dall'uccello)
- Posizione del terreno

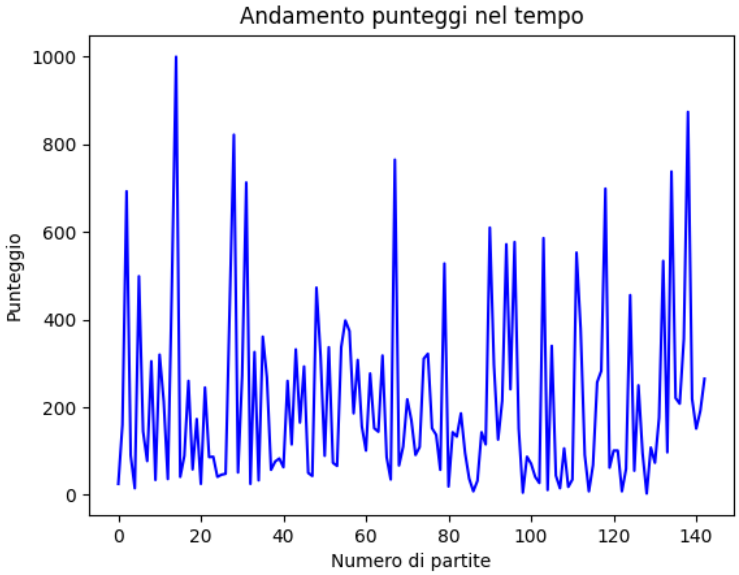
Esecuzione del Gioco

Durante l'esecuzione del gioco:

- Il gioco viene simulato ad ogni passo.
- Ogni volta che il gioco termina, viene registrato il punteggio corrente.
- Un grafico viene aggiornato in tempo reale per visualizzare l'andamento del punteggio.

4 Analisi Prestazioni

Il modello Hill Climbing mostra un comportamento altamente variabile ma potenzialmente molto efficace. Come visibile dal grafico dei punteggi, l'agente è in grado di raggiungere, occasionalmente, risultati estremamente elevati, anche superiori ai 1000 punti. Questo è reso possibile dalla natura del metodo: ad ogni iterazione il sistema esplora localmente due alternative e seleziona quella che, in una simulazione a breve termine, porta al miglior stato. Tuttavia, l'elevata oscillazione dei punteggi evidenzia una mancanza di stabilità. Il modello risulta particolarmente sensibile alla configurazione iniziale e agli elementi casuali della simulazione, con frequenti cadute precoci. Nonostante ciò, la semplicità dell'algoritmo e la sua capacità di ottenere prestazioni eccezionali in singole partite lo rendono un approccio sorprendentemente competitivo in determinati scenari.



5 Approccio Deep Learning e Reinforcement Learning

Il reinforcement learning è un paradigma dell'intelligenza artificiale in cui un agente apprende a compiere azioni in un ambiente al fine di massimizzare una ricompensa cumulativa. A differenza dell'apprendimento supervisionato, dove i dati contengono direttamente le risposte corrette, nel RL l'agente impara attraverso tentativi ed errori, ricevendo feedback sotto forma di ricompense o penalità. Questo approccio si ispira al modo in cui gli esseri viventi imparano comportamenti tramite l'esperienza.
Nel contesto di FlappyAI tale paradigma viene sviluppato in combinazione con un approccio Deep Learning.
Il Deep Learning è una branca del Machine Learning che utilizza reti neurali artificiali composte da molti strati (da cui il termine "deep") per apprendere rappresentazioni complesse dei dati. A differenza degli algoritmi tradizionali, che spesso richiedono un'intensa attività di progettazione manuale delle caratteristiche, le reti profonde apprendono in modo end-to-end, migliorando con l'aumento della quantità di dati e della potenza computazionale.

5.1 Specifica PEAS

- **Performance:**
  - Massimizzare lo score (*score*).
  - Minimizzare le collisioni.
  - Penalizzazione in caso di morte (-5).
  - Visualizzazione dei punteggi nel tempo tramite grafico in tempo reale.
  - Apprendere uno stile di gioco ottimale tramite reward e penalties.
  - Stabilità e consistenza nel tempo
- **Environment:**
  - Ambiente simulato in `Pygame`.
  - Componenti: uccello, tubi (generati proceduralmente e con caratteristiche casuali), terreno.
  - Nessuna interazione umana; controllo interamente autonomo.
  - Agenti fisici impattanti come forza di gravita e velocità.
- **Actuators:**
  - Azione `[0,1]`: *flap* (salto verso l'alto).
  - Azione `[1,0]`: nessuna azione (gravità agisce sull'uccello).
- **Sensors:**
  - Posizione verticale dell'uccello (`rect.y`).
  - Velocità verticale dell'uccello.
  - Posizione e altezza dei tubi.
  - Stato di collisione (con tubi o terreno).
  - Azione intrapresa dall'uccello (sta saltando oppure no)
  - Stato uccello in Pipe (si o no)

5.2 CRISP-DM: Sviluppo Deep Learning

5.3 Business Understanding

L'obiettivo del progetto è realizzare un agente intelligente in grado di apprendere autonomamente a giocare a **Flappy Bird** tramite **Deep Q-Learning**, massimizzando il punteggio attraverso l'esperienza diretta. A differenza di approcci basati su ricerca locale, questo metodo apprende una *policy* ottimale tramite interazioni ripetute con l'ambiente, sfruttando una rete neurale per stimare il valore delle azioni in ogni stato.

5.4 Data Understanding

L'agente apprende tramite l'interazione continua con l'ambiente simulato. Ogni partita fornisce nuove esperienze sotto forma di transizioni:

- Stato corrente dell'ambiente.
- Azione eseguita (flap o non flap).
- Ricompensa ottenuta.
- Stato successivo risultante.
- Stato terminale (fine partita).

Tali esperienze sono memorizzate e utilizzate per l'addestramento del modello.

5.5 Data Preparation

La preparazione dei dati è parte integrante della simulazione:

- Gli input vengono **normalizzati** per migliorarne la gestibilità durante l'apprendimento.
- Gli **stati** vengono rappresentati come array numerici con 10 features continue.
- Le **azioni** sono codificate come vettori di dimensione 2.

5.6 Modeling

L'agente utilizza un'architettura di **Deep Q-Network (DQN)**:

- Rete neurale fully connected a 3 layer (`10 → 256 → 128 → 2`).
- Funzioni di attivazione `ReLU`.
- Loss Function: **MSELoss** tra il Q-value predetto e quello target.
- Strategia di esplorazione: **Epsilon-Greedy**, con decrescita dell'ϵ nel tempo.
- Algoritmo di ottimizzazione: `Adam` con learning rate dinamico.

5.7 Evaluation

Le prestazioni del modello vengono valutate tramite:

- **Punteggio medio** ottenuto per partita.
- **Record massimo** raggiunto.
- **Grafico delle performance** (score e mean score nel tempo).
- Verifica della stabilità nel comportamento e della convergenza della strategia.

5.8 Deployment

Il sistema supporta:

- Addestramento continuo con salvataggio automatico del miglior modello.
- Visualizzazione delle partite e delle metriche in tempo reale.

Modello: `Linear_QNet`

Il modello `Linear_QNet` è una rete neurale completamente connessa (feed-forward) composta da tre layer lineari con funzioni di attivazione ReLU intermedie. È progettato per stimare i valori Q in un contesto di apprendimento per rinforzo.

Architettura

Ha 10 nodi di input e 2 nodi di output, il modello ha la seguente struttura:

$$\begin{aligned} x_1 &= \mathrm{ReLU}(W_1 x + b_1), & W_1 &\in \mathbb{R}^{64\times 10} \\ x_2 &= \mathrm{ReLU}(W_2 x_1 + b_2), & W_2 &\in \mathbb{R}^{32\times 64} \\ \hat{Q} &= W_3 x_2 + b_3, & W_3 &\in \mathbb{R}^{2\times 32} \end{aligned}$$

dove ReLU è la funzione di attivazione rettificata definita da:

$$\mathrm{ReLU}(z) = \max(0, z)$$

ReLU svolge il compito di introdurre non linearità, permettendo alla rete di modellare funzioni complesse.
Il vettore $\hat{Q}$ rappresenta i valori Q predetti per ciascuna azione possibile nello stato corrente.

Training

L'addestramento della rete viene effettuato usando il metodo di Q-learning. Dato uno stato s, un'azione a, una ricompensa r, lo stato successivo s' e una variabile booleana `done` che indica se l'episodio è terminato, il trainer aggiorna i pesi per minimizzare la loss:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \left(Q^{(i)}_{\text{target}} - Q^{(i)}_{\text{pred}} \right)^2$$

Aggiornamento del Target

Per ciascun elemento del batch, si calcola:

$$Q_{\text{new}} = \begin{cases} r & \text{se done} = \text{True} \\ r + \gamma \max_{a'} Q(s', a') & \text{altrimenti} \end{cases}$$

Il valore Q_{new} sostituisce il valore predetto corrispondente all'azione presa a nel vettore target, mentre gli altri valori rimangono invariati.

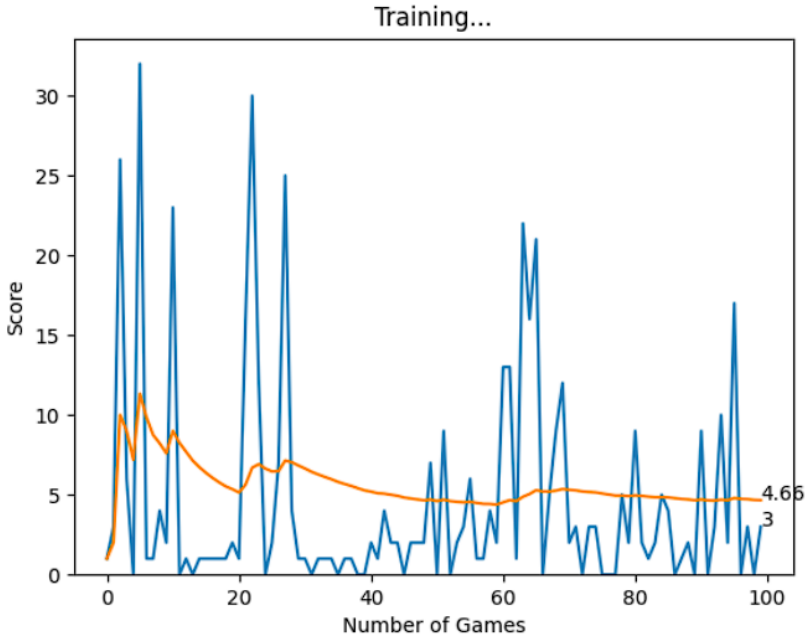
Input del modello

Per ogni stato del gioco, vengono estratti 10 parametri che rappresentano l'andamento corrente della partita:

- Posizione verticale dell'uccello
- Altezza ottimale dell'uccello
- Distanza dall'altezza ottimale
- Distanza orizzontale dal prossimo set di tubi
- Altezza tubo inferiore
- Altezza tubo superiore
- Boolean che indica se l'uccello si trova in un set di tubi
- Boolean che indica se l'uccello sta sbattendo le ali
- Velocità dell'uccello
- Velocità stato precedente dell'uccello

Analisi delle prestazioni

Il modello Deep Learning è stato allenato utilizzando un'esperienza graduale attraverso aggiornamenti continui della rete neurale. Il grafico delle performance mostra come l'agente, durante l'allenamento, attraversi una fase iniziale di alta variabilità nei punteggi, seguita da una progressiva stabilizzazione. La linea arancione, che rappresenta la media mobile, evidenzia un lento ma costante miglioramento nella capacità dell'agente di sopravvivere più a lungo. Sebbene il punteggio massimo raggiunto non sia paragonabile a quello ottenuto dal modello Hill Climbing, il comportamento dell'agente è più regolare e meno soggetto a fluttuazioni estreme. Questo suggerisce che il modello sta apprendendo una politica più generalizzabile, anche se ancora lontana dall'ottimale. La traiettoria suggerisce ampi margini di miglioramento con tempi di allenamento più lunghi e ottimizzazioni ulteriori nella rete neurale o nell'algoritmo di esplorazione.



Riferimenti e risorse utilizzate

In questa sezione sono raccolti tutti gli elementi di supporto al progetto: fonti consultate, materiali utilizzati, dispense di riferimento e altri contenuti accessori che hanno contribuito allo sviluppo di FlappyAI.

- **Rendering grafico e versione Human based del gioco:**
  - `https://github.com/MaxRohowsky/flappy-bird`
- **Guida di riferimento a Pygame e Reti Neurali applicati al gioco Snake**
  - `https://www.youtube.com/watch?v=L8ypSXwyBds&t`
- **Dispense teoriche di supporto a documentazione e processo di sviluppo**
  - Slide del corso di Fondamenti di Intelligenza Artificiale.
    Autore: Fabio Palomba

Conclusioni

Il progetto dimostra l'efficacia di un approccio semplice ma ben strutturato, come Hill Climbing, per la generazione di comportamenti intelligenti in un ambiente dinamico come Flappy Bird. L'utilizzo dei framework PEAS e CRISP-DM ha permesso una progettazione sistematica, seguendo una logica di sviluppo ben strutturata e coerente. Le prestazioni del modello basato su Hill Climbing mostrano un andamento altalenante nei punteggi. Tuttavia, questo comportamento è da considerarsi generalmente positivo, in quanto rappresenta una caratteristica intrinseca dell'algoritmo stesso, che esplora continuamente nuove soluzioni alla ricerca di miglioramenti locali. È evidente come il modello basato su Deep Learning presenti, nelle condizioni attuali, prestazioni inferiori rispetto all'algoritmo di Hill Climbing. Tuttavia, nonostante i risultati non ottimali, l'approccio basato sul Deep Learning ha dimostrato delle buone capacità di apprendimento e di azione. Con un periodo di addestramento più esteso e una progettazione più consapevole e competente, è plausibile attendersi un miglioramento significativo delle sue prestazioni. Si potrebbero esplorare ulteriori soluzioni con modelli basati su reti neurali convoluzionali (CNN), potenzialmente più adeguati e pertinenti rispetto alla natura del problema considerato.