UNIVERSITÀ DI PISA

Scuola di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in Ingegneria Informatica

TESI DI LAUREA

REALIZZAZIONE PYTHON DI UN AMBIENTE DI ESPLORAZIONE SPAZIALE PER DRONI ORIENTATO ALL'APPRENDIMENTO PER RINFORZO

RELATORI:

Prof. Mario G.C.A. Cimino

Prof. Gigliola Vaglini

Dott. Federico A. Galatolo

CANDIDATO:

Diego Casu

ANNO ACCADEMICO 2018/2019

ABSTRACT

L’impiego dell’intelligenza artificiale è ormai affermato e radicato: medicina, robotica, economia e ricerca scientifica beneficiano costantemente dell’utilizzo e dei progressi di questa disciplina. La combinazione di apprendimento automatico e reti neurali artificiali — il cosiddetto apprendimento profondo — rappresenta lo strumento principe per la risoluzione di vaste ed eterogenee categorie di problemi, grazie alla varietà di paradigmi di apprendimento applicabili. In particolare, il paradigma di apprendimento per rinforzo consente un approccio intuitivo verso tutte le classi di problemi in cui l’adattamento ad un ambiente mutevole è fondamentale: la distribuzione di una ricompensa, basata su una valutazione del comportamento, permette di organizzare algoritmi potenti e con forti analogie rispetto al processo di apprendimento umano legato alle esperienze vissute.

L’obiettivo della tesi è il completamento della progettazione e della scrittura di un ambiente di simulazione per sciami di droni, il cui compito è la ricerca di generici bersagli all’interno di una mappa bidimensionale: essi comunicano mediante la stigmergia, ovvero attraverso il rilascio di feromoni digitali, con un comportamento analogo a quello di alcune classi di esseri viventi (ad es. le formiche).

Sulla scia dello standard imposto da OpenAI Gym, il simulatore è un ambiente utilizzabile per l’apprendimento per rinforzo e sarà alla base di future ricerche nell’ambito dell’addestramento di reti neurali stigmergiche.

INDICE

1 INTRODUZIONE

Il problema di partenza, da cui scaturisce la necessità del simulatore, è il seguente: data una mappa bidimensionale, si vuole impiegare uno sciame di droni per la ricerca di obiettivi basandosi su meccanismi di intelligenza di sciame (*swarm intelligence*). A ciascun drone è assegnato un compito molto semplice: esplorare la mappa, riconoscere la presenza di eventuali obiettivi e comunicarla al resto dello sciame tramite il rilascio di un feromone nella posizione presunta, ovvero tramite il meccanismo di comunicazione detto stigmergia.

Un approccio classico al problema porterebbe alla definizione di un insieme di algoritmi di comportamento individuali tali da permettere:

- la sopravvivenza nell'ambiente, con opportuni cambi di rotta al fine di evitare collisioni con ostacoli naturali o droni dello stesso sciame;
- il rilascio di feromone (fisico o digitale) nell'eventualità sia riconosciuto un obiettivo;
- un'esplorazione della mappa che tenga conto della presenza di tracce stigmergiche e che sia in grado di indirizzare il drone verso la zona a priorità maggiore.

L'individuazione e la codifica delle politiche sarebbero compiti svolti da un programmatore umano, a seguito di un preciso studio preliminare.

Un approccio innovativo, che si trova alla base di questo lavoro, consiste nel lasciare che sia una rete neurale artificiale a dedurre i comportamenti: mediante un addestramento basato sul paradigma di apprendimento per rinforzo (*reinforcement learning*), la rete affronta un insieme di esperienze di ricerca di obiettivi, in cui le sue azioni sono valutate opportunamente tramite un meccanismo ricompensa-penalità.

L'addestramento della rete è efficace se è disponibile un'adeguata "palestra" in cui eseguire le esercitazioni: la tesi si prefigge come obiettivo il completamento e l'ampliamento del lavoro di progettazione e scrittura in linguaggio Python di un ambiente di simulazione per sciami di droni, adatto all'apprendimento per rinforzo; il precedente contributo è opera di Filippo Minutella (Minutella 2018/2019a) (Minutella 2018/2019b).

Le caratteristiche principali del simulatore sono due: è scritto in algebra tensoriale, ovvero impiega matrici multidimensionali (tensori) e la loro efficiente implementazione fornita dal modulo NumPy; adotta lo standard di interfaccia definito da OpenAI Gym, il quale si sta affermando nella definizione scientifica di ambienti per l'apprendimento per rinforzo.

La prima parte della tesi è dedicata all'introduzione ai concetti di intelligenza di sciame, intelligenza artificiale e apprendimento per rinforzo, nonché alla libreria NumPy e al toolkit OpenAI Gym. La seconda parte presenta l'ambiente di simulazione e il suo funzionamento, con particolare attenzione verso la documentazione dei metodi che lo compongono: essendo uno strumento di base, risulta fondamentale facilitare futuri interventi di manutenzione, ampliamento e integrazione.

2 INTELLIGENZA DI SCIAME

Definiamo intelligenza di sciame "l'intelligenza collettiva emergente di un gruppo di semplici agenti" (Bonabeau, Dorigo e Theraulaz 1999). Il concetto di intelligenza di sciame nasce in seguito all'osservazione del comportamento di numerose classi di esseri viventi all'interno di un ambiente: un collettivo, composto da individui che svolgono singolarmente una mansione semplice e limitata, mostra all'esterno un comportamento complesso, apparentemente inspiegabile. Tale comportamento è detto comportamento emergente, in quanto non è presente nessun controllo centrale che guidi le attività delle singole entità che compongono l'insieme.

Un semplice esempio è dato dalla condotta adottata dalle colonie di formiche alla ricerca di cibo presentata in Figura 1, dove N rappresenta il nido (*nest*) e F rappresenta il cibo (*food*).

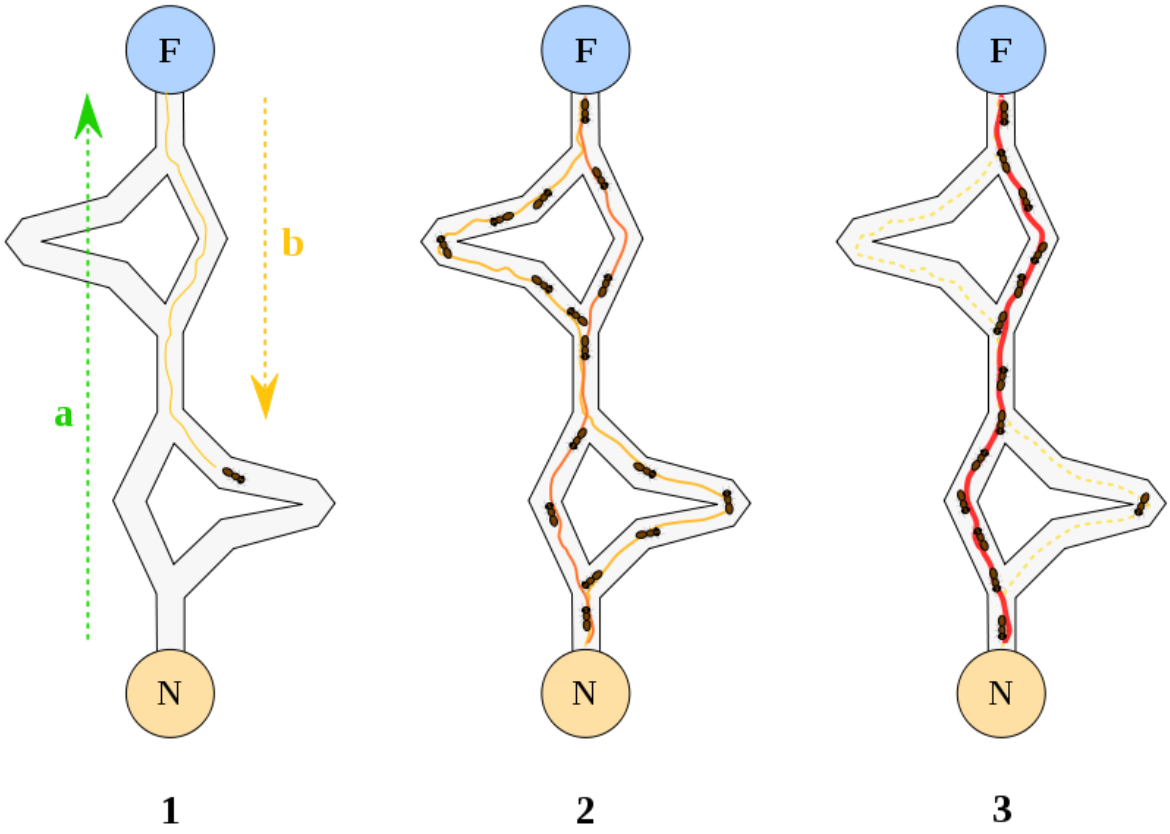

Figura 2.1 Ricerca del percorso più breve da parte di una colonia di formiche (Dréo 2006).

Una formica esplora l'ambiente alla ricerca di cibo: quando ne trova una sorgente, lo preleva e ritorna al nido, lasciando dietro di sé una traccia di feromone. Le altre formiche in esplorazione, se si trovano nel raggio di percezione del feromone, seguono la traccia, prelevano il cibo e rientrano al nido, rilasciando anch'esse del feromone. Con il passare del tempo, la traccia di feromone tende a rinforzarsi lungo il percorso più breve e a guidare naturalmente le formiche: le eventuali tracce più deboli, essendo parte di un percorso non utilizzato, non saranno rinforzate e scompariranno progressivamente a causa dell'evaporazione naturale. Terminata la fonte di cibo, anche la traccia principale evaporerà nel tempo, esaurendo il suo ruolo attrattore; ne segue che le formiche ricominceranno spontaneamente ad esplorare l'ambiente.

Le azioni compiute da ciascuna formica sono semplici e indipendenti, ma danno luogo ad una strategia collettiva di ricerca mediante percorsi minimi complessa e priva di controllo centralizzato. Ciò che permette il sopraggiungere del comportamento emergente è il rilascio di feromone nell'ambiente: un meccanismo di comunicazione siffatto è detto stigmergia.

"La stigmergia è una forma di comunicazione che avviene alterando lo stato dell'ambiente in un modo che influenzerà il comportamento degli altri individui per i quali l'ambiente stesso è uno stimolo." (Kennedy e Eberhart 2001)

Il rilascio del feromone ha un effetto modificativo temporaneo sull'ambiente e fornisce una strategia di coordinamento indiretta. Una formica compie azioni differenti in base allo stimolo chimico ricevuto, ovvero al feromone incontrato, il quale può essere attrattivo (comunica di dirigersi in una direzione) o repulsivo (comunica di allontanarsi da una direzione, ad esempio a causa di un pericolo).

Un meccanismo così semplice e potente è alla base di diversi algoritmi di ottimizzazione: tra i più famosi e pionieristici troviamo *Stochastic Diffusion Search, Ant Colony Optimization e Particle Swarm Optimization*; la nuova sfida è rappresentata dall'approssimare i meccanismi stigmergici di comportamento emergente tramite reti neurali artificiali.

3 RETI NEURALI E APPRENDIMENTO PER RINFORZO

La disciplina dell'intelligenza artificiale racchiude al suo interno innumerevoli ambiti di studio: la riproduzione dell' "abilità di un sistema di interpretare correttamente dati forniti dall'esterno, di imparare da tali dati, e di usare questi insegnamenti per raggiungere obiettivi e compiti specifici attraverso un adattamento flessibile" (Kaplan e Haenlein 2019) fornisce risposte a problemi vasti ed eterogenei. Tale estensione si riflette nell'articolata serie di sottodiscipline che la compongono, ognuna avente caratteristiche, metodi di analisi e di risoluzione distinti.

Tra le branche dell'intelligenza artificiale, una delle più famose e dinamiche è quella dell'apprendimento automatico (*machine learning*), che si pone come obiettivo il dotare i sistemi della capacità di apprendere e migliorare in autonomia, senza un lavoro di programmazione esplicito. Se il sistema in questione è formato da un insieme di neuroni artificiali, si parla di apprendimento profondo (*deep learning*): "un insieme di tecniche basate su reti neurali artificiali organizzate in diversi strati: ogni strato calcola i valori per quello successivo, in modo da elaborare l'informazione in maniera sempre più completa." (Redazione Osservatori Digital Innovation 2019).

Dopo aver introdotto il concetto di neurone artificiale e i principali paradigmi di apprendimento, ci soffermeremo su un esempio di algoritmo di apprendimento per rinforzo: il Q-learning.

3.1 Struttura di un neurone artificiale

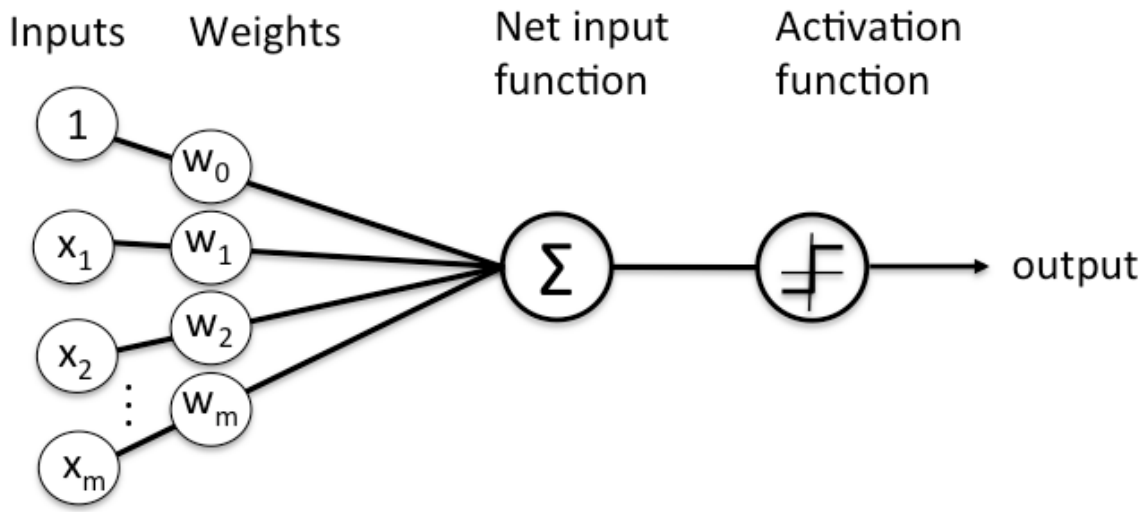

Figura 3.1.1 Modello matematico di un neurone artificiale (Nicholson s.d.).

Un neurone artificiale, così chiamato in quanto ispirato alla struttura di un neurone naturale, è un modello matematico caratterizzato da:

- un certo numero di collegamenti di ingresso (*inputs*), che indicheremo con x_i;
- un insieme di pesi (*weights*) associati agli ingressi, che indicheremo con w_i;
- una funzione di rete che caratterizza il nodo (*net input function*), nel nostro caso la funzione somma;
- una funzione di attivazione (*activation function*), che determina quando il neurone deve produrre un'uscita e che indicheremo con $f(\,)$;
- un collegamento di uscita (*output*), su cui è presente il valore da trasmettere all'esterno.

L'uscita del neurone è determinata dalla combinazione delle componenti in ingresso, secondo la formula:

$$output = f\left(\sum_{i=0}^{m} w_i \cdot x_i\right)$$

La funzione di attivazione dipende dal comportamento che si vuole stabilire per il neurone; esempi comuni di funzioni di attivazione sono la funzione sigmoidea e la funzione gradino.

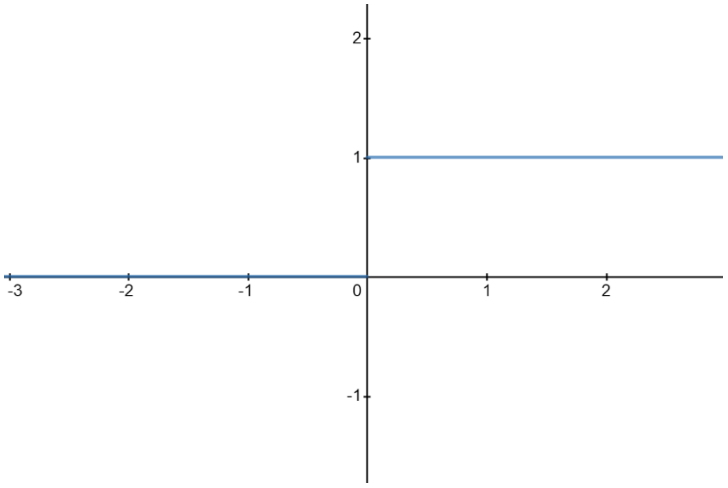

Figura 3.1.2 Funzione gradino.

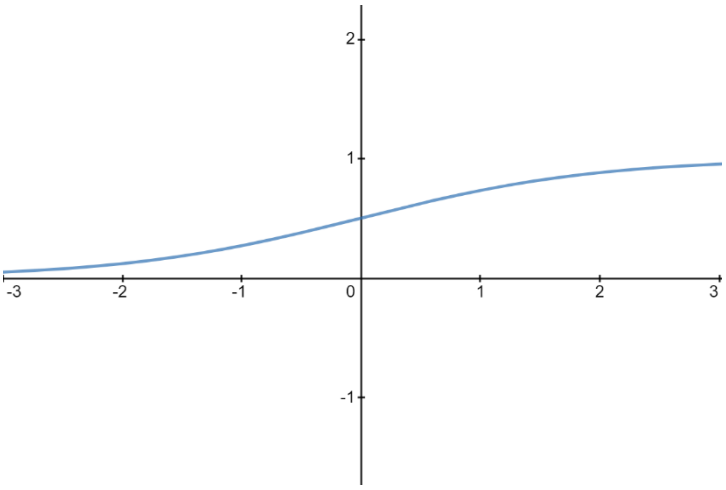

Figura 3.1.3 Funzione sigmoidea.

3.2 Reti neurali e paradigmi di apprendimento

Un neurone artificiale, preso singolarmente, non mostra caratteristiche di particolare rilievo; il maggior pregio di questo modello emerge in seguito all'interconnessione di più neuroni in una struttura a strati: una formazione siffatta è detta rete neurale artificiale e si comporta come un approssimatore universale di funzioni.

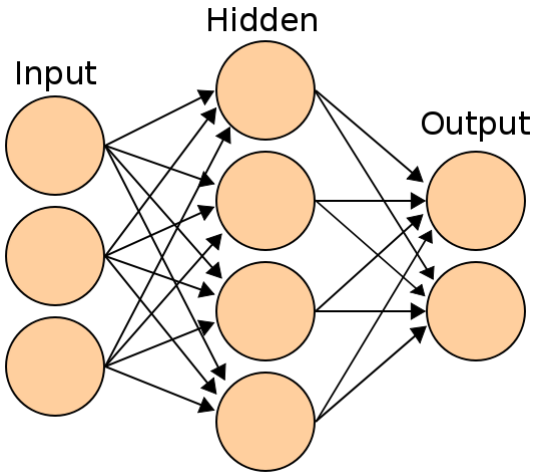

Figura 3.2.1 Rete neurale a 3 strati, con uno strato intermedio nascosto (Burnett 2006).

Esistono diverse configurazioni con cui è possibile costruire reti neurali (ad es. è possibile introdurre anelli di retroazione o meno), ciascuna adatta ad una particolare classe di problemi. Ciò che accomuna tutte le reti è la necessità di definire l'uscita di ogni singolo neurone in modo tale che sia verificata la corrispondenza ingresso-uscita voluta: il processo consiste nella calibrazione dei pesi associati ad ogni collegamento di ingresso e prende il nome di apprendimento (o addestramento). Il processo di apprendimento non è altro che un processo di riduzione dell'errore; esistono tre paradigmi di apprendimento:

- apprendimento supervisionato (*supervised learning*), in cui si dispone di un insieme di coppie ingresso-uscita iniziali da fornire alla rete; se l'addestramento ha successo, la rete deduce la relazione che le lega ed è capace di produrre un'uscita corretta anche nel caso di un ingresso mai visto prima. È tipicamente utilizzato nei problemi di classificazione (ad es. riconoscimento immagini);
- apprendimento non supervisionato (*unsupervised learning*), in cui non è disponibile nessuna coppia ingresso-uscita di riferimento ed è la rete stessa a dover determinare gli opportuni raggruppamenti. È tipicamente utilizzato nei problemi di clustering;

- apprendimento per rinforzo (*reinforcement learning*), in cui la rete viene istruita secondo un meccanismo di ricompensa-penalità, assegnando una valutazione ad ogni azione compiuta; l'apprendimento ha come obiettivo la massimizzazione della ricompensa ottenuta.

3.3 Apprendimento per rinforzo

L'apprendimento per rinforzo è un processo di apprendimento automatico che si basa sulla distribuzione di una ricompensa, detta rinforzo. Solitamente la rete assume il ruolo di un'entità, detta agente, immersa in un ambiente responsivo, ovvero capace di restituire feedback prestazionali in seguito al compimento di un'azione: la rete apprende come comportarsi esplorando progressivamente le combinazioni stato-azione permesse e le relative ricompense ricevute (premi o penalità), avendo come obiettivo la massimizzazione della ricompensa ricevuta. Tale processo di apprendimento mostra forti analogie con il processo di apprendimento umano, che permette di scegliere l'azione migliore in un certo contesto in base alle esperienze pregresse e alle conseguenze vissute.

I maggiori pregi di questo paradigma sono:

- la semplicità e l'intuitività con cui è possibile affrontare la risoluzione di un problema;
- la capacità di fornire alla rete un insieme di strumenti di sopravvivenza e adattamento ad un ambiente mutevole. Ne è un esempio l'insieme di risultati ottenuti dal gruppo DeepMind, i cui agenti addestrati hanno dimostrato di possedere la capacità di giocare ai videogiochi della console Atari 2600 ad un livello comparabile o superiore a quello umano (Silver 2016).

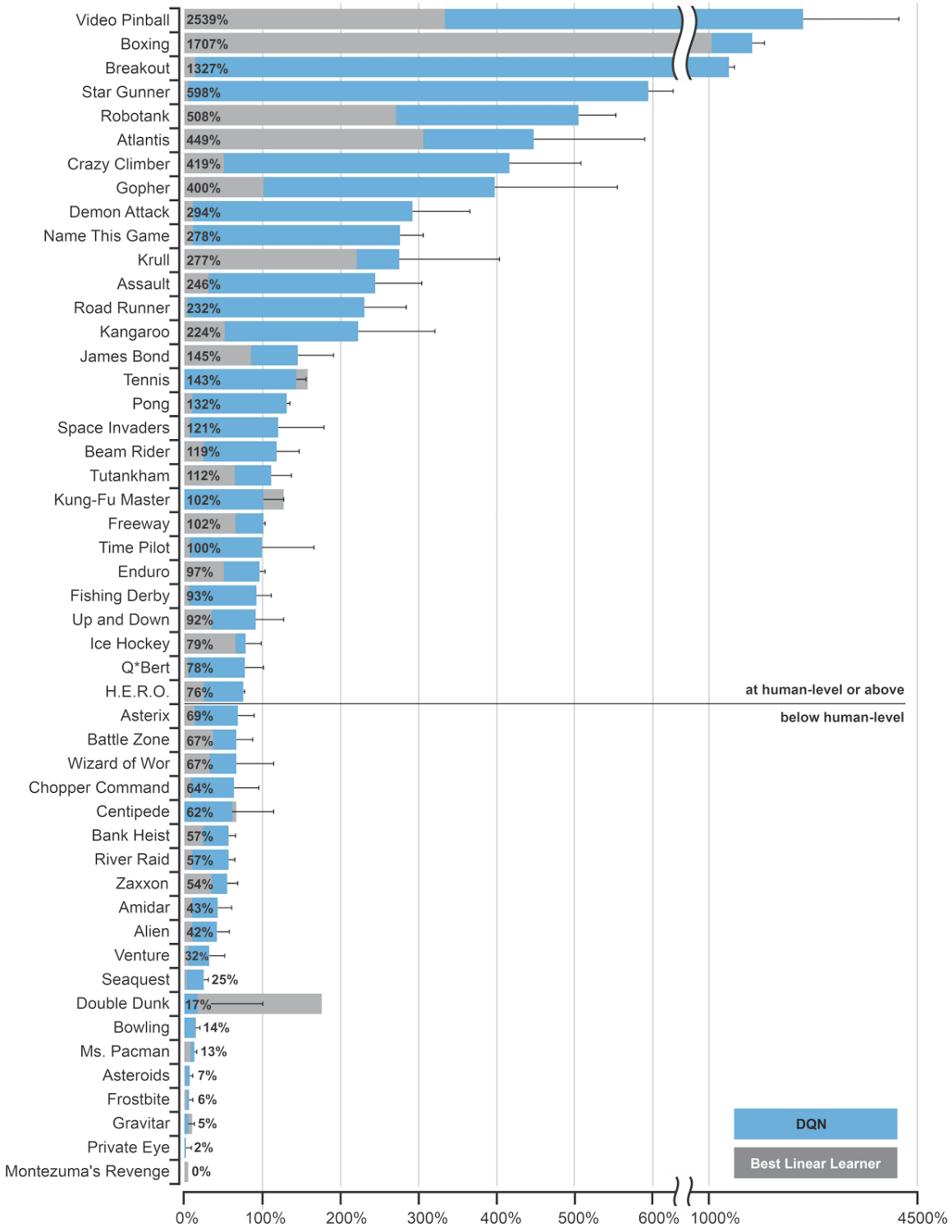

Figura 3.3.1 Risultati dell'algoritmo DQN nella risoluzione di livelli di videogiochi Atari 2600 (Silver 2016).

3.3.1 Modello matematico

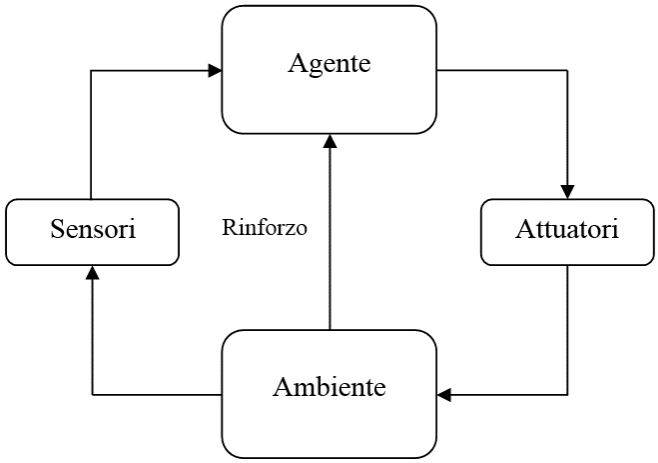

Figura 3.3.1.1 Schema di interazione di un agente nell'apprendimento per rinforzo (Lanzi s.d.).

Un agente, ovvero un'entità che interagisce con l'ambiente di addestramento, è composto da tre sistemi:

- un sistema sensoriale, che fornisce informazioni sull'ambiente;
- un sistema decisionale, che decide quale azione intraprendere sulla base delle informazioni ottenute dal sistema sensoriale;
- un sistema attuativo, che traduce l'azione scelta dal sistema decisionale.

L'ambiente è solitamente assunto come descrivibile da un Processo di Decisione Markoviano, in cui il passaggio ad uno stato del sistema dipende solamente dallo stato precedente (dinamica ad un passo); esso è formalmente definito da:

- un insieme finito di stati S;
- un insieme finito di azioni A;
- una funzione di transizione $T: S \times A \rightarrow \Pi(S)$, che associa ad ogni coppia stato-azione una distribuzione di probabilità su S (uno stato);
- una funzione di rinforzo (*reward*) $R: S \times A \times S \rightarrow \mathbb{R}$, che associa un valore numerico reale ad ogni possibile transizione.

L'obiettivo dell'agente è quello di massimizzare la quantità di rinforzo; la formulazione più comune è data dal modello ad orizzonte finito scontato, con il rinforzo da massimizzare espresso come

$$E\left[\sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k+1}\right] \qquad 0 < \gamma \leq 1$$

dove γ è detto fattore di sconto e determina quali rinforzi sono più importanti nel processo di apprendimento (quelli iniziali se $\gamma \to 0$, quelli futuri se $\gamma \to 1$). Il comportamento dell'agente, ovvero il suo processo decisionale, dipende da una politica $\pi: S \to \Pi(\mathrm{A})$, che associa ad uno stato una distribuzione di probabilità su A (un'azione).

Durante il processo di apprendimento per rinforzo, l'agente apprende:

- una funzione valore di stato $V: S \to \mathbb{R}$, che associa ad uno stato il valore del rinforzo atteso in quello stato;
- una funzione valore di azione $Q: S \times A \to \mathbb{R}$, che associa ad una coppia stato-azione il valore del rinforzo atteso se, in tale stato, si compie quell'azione. (Lanzi s.d.)

3.3.2 Un semplice esempio: il Q-learning

Un esempio di apprendimento per rinforzo è dato dall'algoritmo di addestramento noto come Q-learning (Watkins 1989): l'obiettivo è quello di ricavare la funzione di valore di azione attraverso l'uso di una matrice Q che mette in relazione tutti i possibili stati con tutte le possibili azioni.

La matrice Q è inizializzata con un valore arbitrario iniziale; successivamente, ad ogni instante t e fino ad un certo stato finale, l'agente seleziona un'azione a_t, riceve un rinforzo r_t, entra nello stato successivo s_{t+1} e procede all'aggiornamento della stessa mediante la relazione

$$Q^{new}(s_t, a_t) = (1 - \alpha) \cdot Q(s_t, a_t) + \alpha \cdot (\, r_t + \gamma \cdot \max_a Q(s_{t+1}, a)\,)$$

dove $0 < \alpha \leq 1$ è detto tasso di apprendimento, γ è il fattore di sconto e il valore massimo è la stima del valore ottimo futuro.

Una volta costruita la matrice, l'agente può utilizzarla per decidere, in ogni stato, l'azione ottima da intraprendere.

4 IL TOOLKIT OPENAI GYM

OpenAI Gym è un toolkit, distribuito sotto forma di libreria open-source, per lo sviluppo di algoritmi di apprendimento per rinforzo: attraverso un'interfaccia semplice e intuitiva, sono messi a disposizione dell'utente un certo numero di ambienti, nei quali è possibile simulare la risoluzione di specifici problemi. Gym nasce nella speranza di fornire ambienti di test migliori in varietà e semplicità d'uso, nonché di standardizzare la struttura degli ambienti stessi utilizzati nelle pubblicazioni scientifiche e nella ricerca sull'apprendimento per rinforzo.

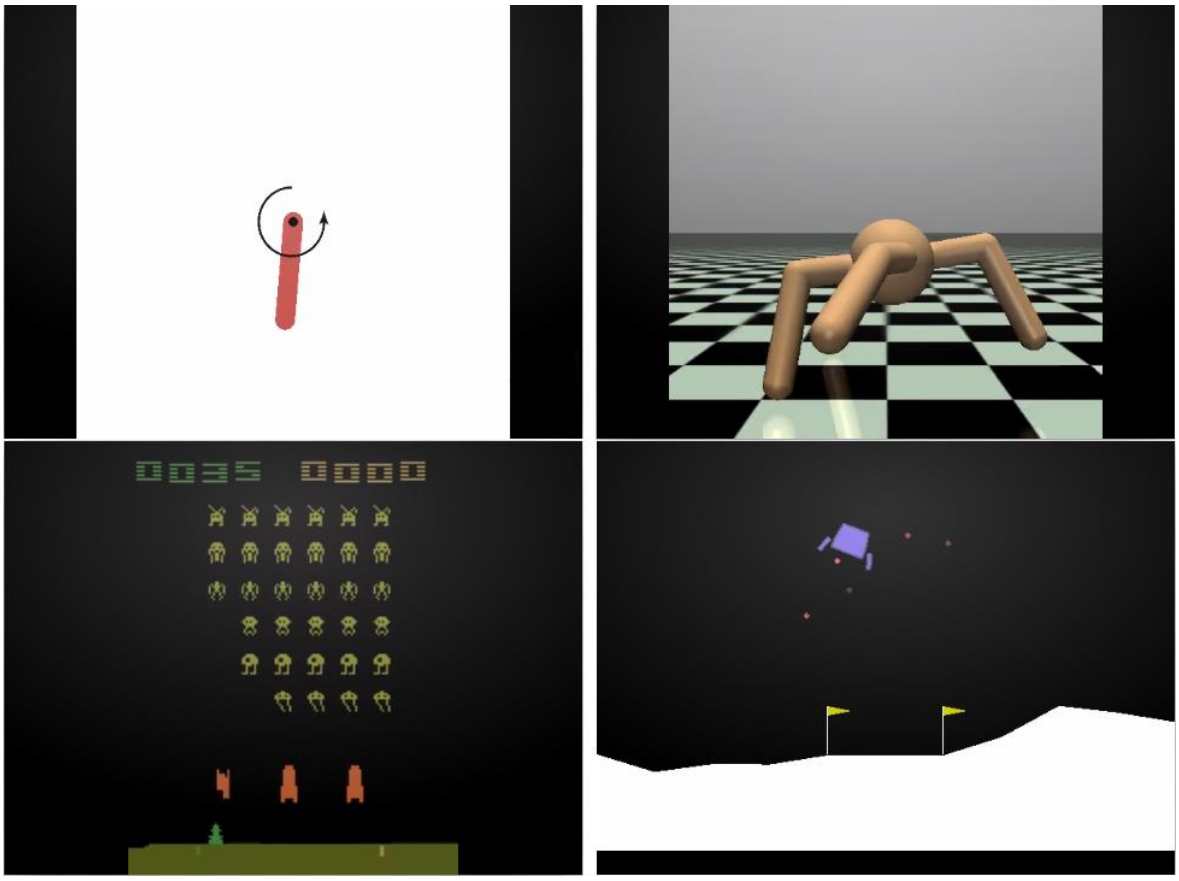

Figura 4.1 Da destra a sinistra, gli ambienti Pendulum-v0, Ant-v2, SpaceInvaders-v0 e LunarLander-v2 (OpenAI s.d.)

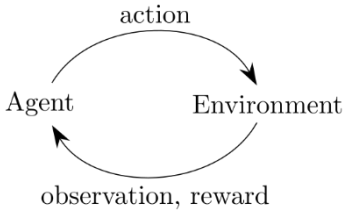

Figura 4.2 Il processo di apprendimento per rinforzo in OpenAI Gym (OpenAI s.d.).

In Gym, il processo di apprendimento per rinforzo ruota intorno alle seguenti variabili:

- observation : *object*

  Rappresenta l'osservazione che l'agente ha compiuto nell'ambiente. È un oggetto specifico dell'ambiente e tali sono le informazioni che può contenere;

- reward : *float*

  Rappresenta il rinforzo ottenuto in seguito all'ultima azione compiuta. L'obiettivo dato è la massimizzazione della ricompensa totale;

- done : *bool*

  Flag che indica, se pari a True, la necessità di resettare l'ambiente. È utile per determinare il raggiungimento dello stadio conclusivo della simulazione;

- info : *dict*

  Insieme di informazioni sulla simulazione, utili per il processo di debug. Non ne è consentito l'utilizzo all'interno del processo di apprendimento.

I principali metodi che compongono l'interfaccia di Gym sono:

- step(), il cui compito è eseguire un passo della simulazione in accordo ad una azione scelta. Restituisce le relative variabili observation, reward, done e info;
- reset(), il cui compito è reinizializzare l'ambiente;
- render(), il cui compito è il disegno a video dello stato della simulazione.

Un semplice programma di test ha la seguente forma (OpenAI s.d.):

```
1.  import gym
2.  env = gym.make('CartPole-v0')
3.  for i_episode in range(20):
4.      observation = env.reset()
5.      for t in range(100):
6.          env.render()
7.          print(observation)
8.          action = env.action_space.sample()
9.          observation, reward, done, info = env.step(action)
10.         if done:
11.             print("Episode finished after {} timesteps".format(t+1))
12.             break
13. env.close()
```

5 ALGEBRA TENSORIALE: LA LIBRERIA NUMPY

L'algebra tensoriale è la branca dell'algebra che si occupa delle proprietà e delle operazioni che riguardano i tensori, oggetti matematici che rappresentano una generalizzazione di scalari e vettori. Un tensore è caratterizzato da un rango (o ordine), pari al numero di dimensioni che lo caratterizzano e lo descrivono: uno scalare è un tensore di rango zero, mentre un vettore è un tensore di rango uno.

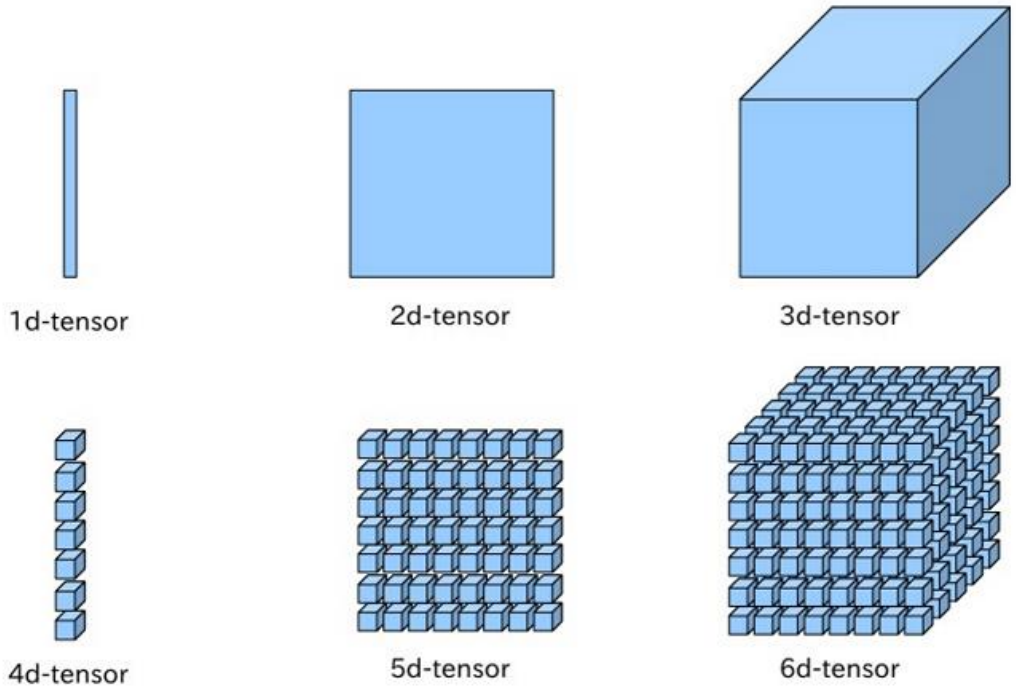

Figura 5.1 Esempi di tensori (Araujo dos Santos s.d.).

Il concetto matematico di tensore è centrale in molte discipline scientifiche, tra cui la termodinamica, la meccanica e l'elettronica (DoITPoMS s.d.), ma può essere generalizzato in modo tale da rappresentare un contenitore multidimensionale di informazioni. Il vantaggio di questo approccio è la possibilità di sfruttare le operazioni su tensori per manipolare dati generici in modo parallelo e scalabile su grandi dimensioni.

Una libreria Python che permette di operare su array N-dimensionali secondo entrambi gli approcci è NumPy: oltre a rappresentare un'ampia e versatile collezione di metodi, il modulo si distingue per l'estrema efficienza con cui sono stati implementati i suoi meccanismi.

In NumPy, un ndarray (array N-dimensionale) è descritto dai dati che contiene e da un insieme di informazioni aggiuntive, tra cui il numero di dimensioni, la forma e il tipo dei dati stessi. A differenza delle strutture di base fornite da Python (ad es. le liste), un ndarray è contenuto in blocchi contigui e omogenei di memoria RAM; questa impostazione permette alla libreria di:

- implementare in linguaggi efficienti e a "basso livello", in particolare il C, le operazioni sugli elementi dell'array, in quanto è possibile sfruttare l'aritmetica dei puntatori ed ottenere accessi in tempo costante;
- sfruttare maggiormente la cache CPU secondo il principio di località spaziale;
- sfruttare le istruzioni vettorizzate delle moderne CPU (Intel SSE e AVX, AMD XOP). Ad esempio, molteplici numeri in virgola mobile possono essere caricati in registri da 128, 256 o 512 bit e processati simultaneamente, se contigui;
- essere compatibile con librerie algebriche altamente ottimizzate, librerie per la parallelizzazione multithread e compilatori JIT (Numba) (Rossant s.d.).

6 L'AMBIENTE DI SIMULAZIONE PER DRONI

6.1 Visione funzionale

Data una mappa bidimensionale caratterizzata da obiettivi, ostacoli e non-ostacoli[1], il simulatore permette a uno sciame di droni di muoversi e rilasciare tracce stigmergiche.

I droni che compongono la formazione sono tutti uguali e possono essere calibrati in:

- numero;
- dimensione;
- raggio di osservazione dell'ambiente circostante;
- inerzia nel movimento.

Il movimento è permesso nelle due direzioni secondo il sistema di riferimento matriciale righe-colonne e ad ogni passo può assumere un valore arbitrario positivo o negativo (spostamento in avanti o indietro); l'effettivo spostamento dipende dalla velocità attuale del drone e dall'inerzia applicata al modello.

È possibile impostare un arbitrario numero di feromoni rilasciabili: il tipo del feromone (attrattivo o repulsivo) è noto esclusivamente al programmatore e non è salvato in nessun parametro del simulatore; l'intensità del rilascio è sempre uguale per tutti i droni, mentre può variarne il raggio. Ad ogni passo della simulazione, tutte le tracce stigmergiche presenti subiscono gli effetti del meccanismo di evaporazione, che ne riduce l'intensità fino a causarne l'annullamento.

Grazie ai contenitori multidimensionali offerti da NumPy, è possibile eseguire in parallelo più simulazioni indipendenti caratterizzate dagli stessi parametri di base (mappa, caratteristiche dei droni e dei feromoni), con il solo vincolo di programmazione dovuto alla gestione multidimensionale delle informazioni scambiate attraverso le funzioni di interfaccia. Nel caso si scelga di lavorare su una simulazione singola, è possibile abilitare il disegno a video della stessa.

L'interfaccia disponibile è composta in accordo alle principali funzioni disponibili in OpenAI Gym: in particolare, il rinforzo distribuito è stabilito in base al numero di obiettivi trovati, alla distanza media da tutti gli obiettivi non ancora trovati e alle collisioni occorse.

[1] Ad esempio, un insieme di perimetri visuali in cui ci si aspetta che i droni permangano, ma che non sono fisicamente presenti.

6.2 La classe DroneSimulator

Il simulatore è scritto in Python 3.7 ed è contenuto all'interno di una singola classe chiamata DroneSimulator, dove tutti i parametri calibrabili sono passati in fase di inizializzazione; le funzioni di interfaccia utilizzabili dal programmatore sono step(), reset() e render(). Il codice sorgente è disponibile su GitHub (Casu 2019).

6.2.1 Librerie di supporto

```
1.  from __future__ import division
2.  from pyqtgraph.Qt import QtCore, QtGui
3.  from PIL import Image
4.  import numpy as np
5.  import pyqtgraph as pg
6.  import sys
7.  import random
8.  import threading
9.
10. np.set_printoptions(threshold=sys.maxsize)
11. pg.setConfigOptions(imageAxisOrder='row-major')
12. pg.setConfigOption('background', 'w')
```

Nello specifico, si osserva che:

- la libreria PyQtGraph è usata in abbinamento a PyQt4;
- la libreria per la manipolazione delle immagini è Pillow, nonostante sia importata con l'alias PIL.

L'impostazione np.set_printoptions(threshold=sys.maxsize) è necessaria per stampare a video i valori elaborati da NumPy senza troncamento.

L'impostazione pg.setConfigOptions(imageAxisOrder='row-major') è necessaria per evitare il comportamento di default di PyQtGraph, in cui l'immagine è mostrata a video trasposta.

L'impostazione pg.setConfigOption('background', 'w') stabilisce il colore di background nella finestra di disegno a video: il colore bianco permette un maggiore contrasto con il colore nero del terreno nella mappa.

6.2.2 Attributi

La classe si compone dei seguenti attributi:

- __bitmap : *str*
  Contiene il percorso della mappa bidimensionale (bitmap) in cui verrà effettuata la simulazione.
- __batch_size : *int*
  Numero di simulazioni parallele da avviare. Il parametro deve assumere un valore maggiore o uguale a 1.
- __drone_size : *int*
  Ampiezza di un drone: un drone è rappresentato come un quadrato di raggio pari a drone_size. Il parametro deve assumere un valore maggiore o uguale a 0.
- __observation_range : *int*
  Ampiezza dello spazio di osservazione di un drone: un drone è capace di vedere lo spazio circostante, a partire da ogni sua estremità, per un raggio pari a observation_range. Il parametro deve assumere un valore maggiore o uguale a 0.
- __amount_of_drones : *int*
  Numero di droni che partecipano a ciascuna simulazione. Il parametro deve assumere un valore maggiore o uguale a 1.
- __stigmergy_evaporation_speed : *ndarray (1D)*
  Array monodimensionale contenente le velocità di evaporazione di ciascun livello stigmergico, uguali per ogni simulazione parallela. Il numero dei livelli stigmergici — uno per ciascun tipo di feromone che è possibile rilasciare — è automaticamente considerato come pari alla dimensione dell'array, ovvero al numero di velocità fornite. Le velocità devono essere numeri interi maggiori o uguali a 0: in caso siano inseriti numeri reali, essi saranno approssimati all'intero più vicino. Per maggiori informazioni sul significato della velocità di evaporazione, si veda la funzione __stigmergy_evaporation().
- __stigmergy_colours : *ndarray (2D)*
  Array contenente i colori con cui saranno visualizzate le tracce stigmergiche: ogni colore è una terna RGB sotto forma di ndarray ed è associato ad un singolo feromone. Il numero di colori fornito devo corrispondere al numero di livelli

stigmergici dichiarato tramite il parametro __stigmergy_evaporation_speed; ad esempio, per tre livelli stigmergici, si potrebbe avere:

```
1.  stigmergy_evaporation_speed = numpy.array([20, 30, 40])
2.  stigmergy_colours = numpy.array([[255,64,0],[255,128,0],[255,255,0]])
```

- __inertia : *float*

  Rappresenta l'inerzia che caratterizza il movimento dei droni.

- __collision_detection : *ndarray (1D)*

  Array monodimensionale di booleani che determina quali livelli della mappa, oltre al livello degli obiettivi, sono da considerare abilitati alle collisioni: un'entrata pari a True abilita le collisioni, un'entrata pari a False disabilita le collisioni. La dimensione dell'array deve essere pari al numero dei livelli presenti nella mappa, escluso il livello degli obiettivi.

- __max_steps : *int*

  Indica il numero massimo di step che possono essere effettuati in ciascuna simulazione parallela: raggiunto tale numero, risulta necessario il reset del simulatore. Il parametro deve essere maggiore o uguale a 1.

- __current_steps : *int*

  Tiene traccia del numero di step effettuati in ciascuna simulazione.

- __rendering_allowed : *bool*

  Determina se il simulatore deve effettuare il rendering della simulazione, ovvero visualizzare a video l'andamento della stessa. L'opzione è disponibile solamente se batch_size è pari a 1: in caso contrario, sarà generata un'eccezione.

- __drone_colour : *ndarray (1D)*

  Array contenente il colore con cui saranno visualizzati i droni: il colore è una terna RGB sotto forma di ndarray ed è uguale per ogni drone.

```
1.  drone_colour= numpy.array([255, 255, 255])
```

- __targets : *ndarray (2D)*

  Array bidimensionale avente come dimensioni le dimensioni della mappa fornita. Un elemento pari a 1 indica la presenza di un obiettivo nella posizione; in caso contrario, l'elemento è pari a 0.

- __collision : *ndarray (2D)*

  Array bidimensionale avente come dimensioni le dimensioni della mappa fornita. Un elemento pari a 1 indica la presenza di un elemento con cui il drone può collidere (un ostacolo); in caso contrario, l'elemento è pari a 0.

- __no_collision : *ndarray (2D)*

  Array bidimensionale avente come dimensioni le dimensioni della mappa fornita. Un elemento pari a 1 indica la presenza di un elemento con cui il drone non può collidere (utile per la visualizzazione di informazioni aggiuntive); in caso contrario, l'elemento è pari a 0.

- __environment_bitmap : *ndarray (3D)*

  Matrice bidimensionale, avente come dimensioni le dimensioni della mappa fornita, in cui ogni elemento è una terna RGB, la quale determina il colore dell'elemento contenuto nella posizione. Contiene la mappa iniziale e le sue informazioni fisse (terreno, obiettivi e ostacoli); gli elementi dinamici (droni e tracce stigmergiche) sono aggiunti in fase di rendering senza modificarne il contenuto.

- __drones_position_float : *ndarray (3D)*

  Array tridimensionale che, per ogni simulazione, contiene le coordinate dei droni nella mappa, salvate come float. Sono utilizzate solamente nel calcolo dello spostamento con velocità.

- __drones_position : *ndarray (3D)*

  Array tridimensionale che, per ogni simulazione, contiene le coordinate dei droni nella mappa, approssimate come interi. Sono utilizzate in tutti i calcoli e nel disegno della simulazione a video, ad eccezione del calcolo dello spostamento con velocità.

- __drones_velocity : *ndarray (3D)*

  Array tridimensionale che, per ogni simulazione, contiene le componenti di velocità dei droni nelle due direzioni della mappa, salvate come float.

- __drawn_drones : *ndarray (4D)*

  Array che, per ogni simulazione, contiene il disegno di ciascun drone su una matrice bidimensionale di dimensioni pari alle dimensioni della mappa. Nello specifico, per ogni drone è mantenuta una matrice in cui gli elementi pari a 1 indicano la posizione del drone nella mappa, in accordo alle coordinate e alla dimensione dello stesso.

- __targets_achieved : *ndarray (4D)*

  Array che mantiene, per ogni simulazione, gli obiettivi trovati da ciascun drone durante la simulazione. In particolare, per ogni drone è mantenuta una matrice, di dimensioni pari alle dimensioni della mappa, in cui gli elementi pari a 1 indicano gli obiettivi trovati.
- __stigmergy_space : *ndarray (4D)*

  Array che mantiene, per ogni simulazione, lo stato dello spazio stigmergico. Nello specifico, lo spazio stigmergico di una simulazione è composto da un numero di matrici bidimensionali pari al numero di feromoni fornito: ogni matrice ha dimensione pari alla dimensione della mappa e presenta elementi maggiori di 0 in corrispondenza delle posizioni in cui i droni hanno rilasciato tracce stigmergiche. Il valore dell'elemento determina l'intensità della traccia stigmergica presente.
- __image : *ndarray (3D)*

  Matrice bidimensionale, avente come dimensioni le dimensioni della mappa fornita, in cui ogni elemento è una terna RGB, la quale determina il colore dell'elemento contenuto nella posizione. Contiene la mappa iniziale, le sue informazioni fisse (terreno, obiettivi e ostacoli) e gli elementi dinamici (droni e tracce stigmergiche); viene utilizzata per la visualizzazione a video quando il rendering è abilitato.
- __image_semaphore : *thread.lock*

  Lock utilizzato per l'accesso concorrente all'attributo __image.

6.2.3 Metodi

__init()__

```
1.  def __init__(self, bitmap, batch_size, drone_size, observation_range,
2.                amount_of_drones, stigmergy_evaporation_speed, stigmergy_colours,
3.                inertia, collision_detection, max_steps, rendering_allowed=False,
4.                drone_colour=[255, 255, 255]):
5.
6.      self.__init_simulator_parameters(bitmap, batch_size, drone_size,
7.                                       observation_range, amount_of_drones,
8.                                       stigmergy_evaporation_speed,
9.                                       stigmergy_colours, inertia,
10.                                      collision_detection,  max_steps,
11.                                      rendering_allowed, drone_colour)
12.
13.     self.__init_environment_parameters()
14.     self.__parse_bitmap()
15.
16.     self.__init_drones_parameters()
17.     self.__init_drones()
18.
19.     self.__init_stigmergy_space()
20.     self.__init_rendering_parameters()
```

Inizializza un oggetto DroneSimulator a partire dai parametri forniti dall'utente: per maggiori informazioni sui parametri in ingresso al metodo, si veda l'elenco degli attributi della classe. Se non specificato, il rendering è disabilitato (rendering_allowed = False); se non specificato, il colore dei droni è bianco (drone_colour=[255, 255, 255]).

__init_simulator_parameters()

```
1.  def __init_simulator_parameters(self, bitmap, batch_size, drone_size,
2.                                  observation_range, amount_of_drones,
3.                                  stigmergy_evaporation_speed, stigmergy_colours,
4.                                  inertia, collision_detection, max_steps,
5.                                  rendering_allowed, drone_colour):
6.
7.      self.__bitmap = bitmap
8.      self.__batch_size = batch_size
9.      self.__observation_range = observation_range
10.     self.__drone_size = drone_size
11.     self.__amount_of_drones = amount_of_drones
12.     self.__stigmergy_evaporation_speed = stigmergy_evaporation_speed.astype(int)
13.     self.__stigmergy_colours = stigmergy_colours
14.     self.__inertia = inertia
15.     self.__collision_detection = collision_detection
16.     self.__max_steps = max_steps
17.     self.__rendering_allowed = rendering_allowed
18.     self.__drone_colour = drone_colour
```

Inizializza i parametri del simulatore: per maggiori informazioni sui parametri in ingresso al metodo, si veda l'elenco degli attributi della classe.

__ init_environment_parameters()

```
1.  def __init_environment_parameters(self):
2.      self.__environment_bitmap = None
3.      self.__targets = np.array([])
4.      self.__collision = np.array([])
5.      self.__no_collision = np.array([])
```

Definisce i parametri dell'ambiente; essi saranno inizializzati nel metodo __parse_bitmap().

__parse_bitmap()

```
1.  def __parse_bitmap(self):
2.      input_array = np.asarray(Image.open(self.__bitmap))
3.      rgb_bit_array = np.unpackbits(input_array, axis=2)
4.      # rgb_bit_array is a matrix of pixels, where each cell (each pixel) is
5.      # a 24-bit array
6.
7.      collision = []
8.      no_collision = []
9.      level_founded = 0
10.
11.     for i in range(0, 24):
12.         level = rgb_bit_array[:, :, i]
13.         # Only levels with at least 1 item are inserted in the environment
14.         if np.any(level):
15.             if level_founded == 0:
16.                 # First level is composed of targets
17.                 self.__targets = np.asarray(level)
18.                 self.__environment_bitmap = np.full((self.__targets.shape[0],
19.                                                      self.__targets.shape[1], 3),
20.                                                     0)
21.             else:
22.                 if self.__collision_detection[level_founded - 1]:
23.                     collision.append(level)
24.                 else:
25.                     no_collision.append(level)
26.
27.             self.__environment_bitmap[level == 1, :] = (
28.                     self.__environment_bitmap[level == 1, :] + self.__get_colour(i))
29.             level_founded += 1
30.
31.     if not collision:
32.         collision = np.zeros((1, self.__targets.shape[0], self.__targets.shape[1]))
33.     else:
34.         collision = np.asarray(collision)
35.
36.     if not no_collision:
37.         no_collision = np.zeros((1, self.__targets.shape[0],
38.                                  self.__targets.shape[1]))
39.     else:
40.         no_collision = np.asarray(no_collision)
41.
42.     self.__collision = np.sum(collision, axis=0)
43.     self.__collision[self.__collision > 0] = 1
44.
45.     self.__no_collision = np.sum(no_collision, axis=0)
46.     self.__no_collision[self.__no_collision > 0] = 1
```

Preleva la mappa dal percorso specificato in fase di inizializzazione e la trasforma in una matrice di pixel manipolabile: per ogni livello individuato, vengono aggiornate le mappe degli obiettivi, delle collisioni e delle non-collisioni; inoltre, è costruita la mappa dell'ambiente con le informazioni fisse ed i colori associati in formato RGB.

I colori sono individuati da 24 bit, dove ogni bit rappresenta un livello: con quest'ultimo termine individuiamo un insieme di informazioni denotate dallo stesso colore (la mappa

sarà sempre in due dimensioni, a prescindere dal numero di livelli presenti); i livelli sono esaminati a partire dal bit più significativo. La scelta dei colori deve rispettare la seguente politica:

- il terreno è sempre nero, ovvero è individuato da soli 0;
- il primo livello è sempre quello degli obiettivi ed un singolo obiettivo occupa sempre una ed una sola casella (in caso siano occupate più caselle contigue, saranno riconosciuti obiettivi multipli);
- ogni colore utilizzato deve avere un solo bit a 1;
- l'unica eccezione alla regola precedente è data dal caso in cui due o più oggetti appartenenti a livelli differenti siano sovrapposti, ovvero debbano essere disegnati nella stessa casella. In questo caso, il colore della casella deve presentare più bit 1: nello specifico, deve presentare un bit 1 in corrispondenza di ogni livello sovrapposto.

Un esempio di utilizzo dei colori è il seguente:

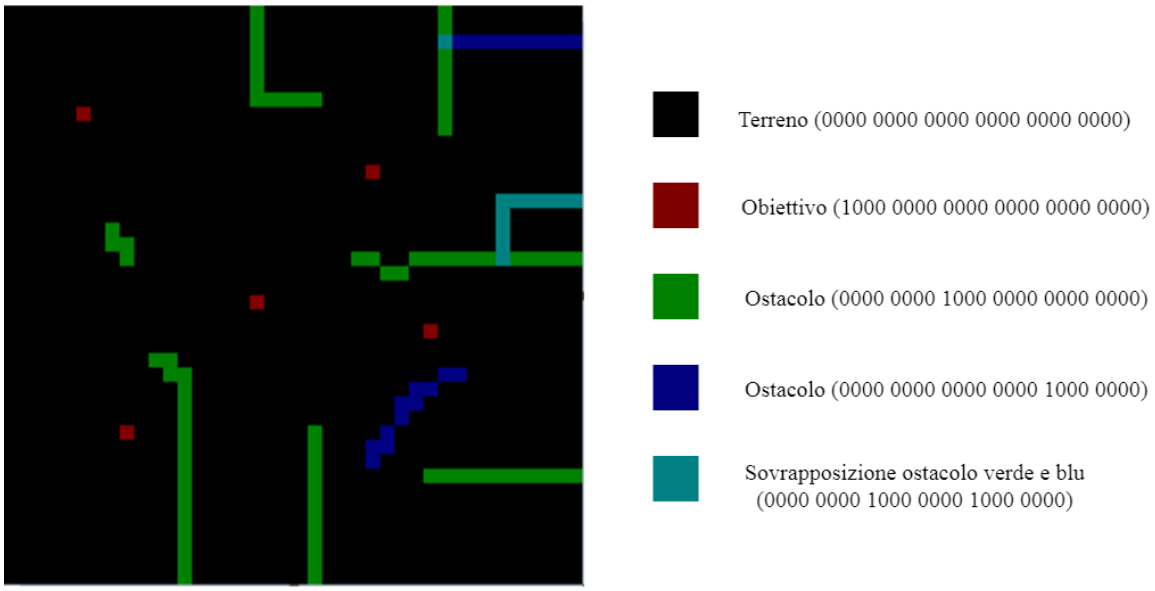

Figura 6.2.3.1 Un esempio di mappa bidimensionale e scelta dei colori.

__get_colour()

```
1.  def __get_colour(self, i):
2.      colour = np.zeros(shape=3)
3.      i = 24 - i - 1
4.
5.      colour[2 - i // 8] = 2 ** (i % 8)
6.      return colour
```

Funzione di utilità che restituisce un colore in formato RGB a partire dalla posizione del bit 1 nella rappresentazione binaria.

Parametri

- i : *int*

  Posizione del bit 1 nella rappresentazione binaria del colore.

Restituisce

- colour : *ndarray (1D)*

  Array monodimensionale contenente la rappresentazione RGB del colore.

__init_drones_parameters()

```
1.  def __init_drones_parameters(self):
2.      self.__drones_position_float = None
3.      self.__drones_position = np.full((self.__batch_size,
4.                                        self.__amount_of_drones, 2), -1)
5.
6.      self.__drones_velocity = np.zeros((self.__batch_size,
7.                                         self.__amount_of_drones, 2))
8.
9.      self.__drawn_drones = np.zeros((self.__batch_size,
10.                                     self.__amount_of_drones,
11.                                     self.__targets.shape[0],
12.                                     self.__targets.shape[1]))
13.
14.     self.__targets_achieved = np.zeros((self.__batch_size,
15.                                         self.__amount_of_drones,
16.                                         self.__targets.shape[0],
17.                                         self.__targets.shape[1]))
18.     self.__current_steps = 0
```

Definisce i parametri relativi ai droni; essi saranno inizializzati nel metodo __init_drones().

__init_drones()

```
def __init_drones(self):
    for batch_index in range(self.__batch_size):
        drone_index = 0
        while drone_index < len(self.__drones_position[batch_index]):
            self.__drones_position[batch_index, drone_index] = np.asarray([
                random.randint(0, self.__targets.shape[0] - 1),
                random.randint(0, self.__targets.shape[1] - 1)
            ])

            # A drone is correctly positioned if it's rendered completely inside
            # the map and it doesn't collide with environment or other drones
            if not self.__out_of_map(batch_index, drone_index, self.__drone_size):
                self.__draw_drone(batch_index, drone_index)
                if not self.__detect_collision(batch_index):
                    drone_index += 1

    self.__drones_position_float = np.copy(self.__drones_position).astype(float)
```

Inizializza i droni di tutte le simulazioni parallele, assegnando a ciascuno di essi una posizione casuale nella mappa: la posizione è valida solamente se il drone risulta completamente dentro la mappa e non collide con elementi dell'ambiente (tra cui eventuali altri droni).

__draw_drone()

```
def __draw_drone(self, batch_index, drone_index):
    radius = self.__drone_size
    position_axis_0 = self.__drones_position[batch_index, drone_index, 0]
    position_axis_1 = self.__drones_position[batch_index, drone_index, 1]
    interval_axis_0, interval_axis_1 = self.__drawing_boundaries(position_axis_0,
                                                                 position_axis_1,
                                                                 radius)

    drone_level = np.zeros((self.__targets.shape[0], self.__targets.shape[1]))
    drone_level[interval_axis_0, interval_axis_1] = 1
    self.__drawn_drones[batch_index, drone_index] = drone_level
```

Disegna un drone appartenente ad una simulazione: nel caso in cui il drone sia parzialmente o completamente fuori dalla mappa, sarà disegnata solamente la parte interna a quest'ultima.

Parametri

- batch_index : *int*

  Indice della simulazione parallela su cui effettuare l'operazione.

- drone_index : *int*

  Indice del drone da disegnare.

__drawing_boundaries()

```
def __drawing_boundaries(self, position_axis_0, position_axis_1, radius):
    start_point_axis_0 = position_axis_0 - radius
    end_point_axis_0 = position_axis_0 + radius + 1
    start_point_axis_1 = position_axis_1 - radius
    end_point_axis_1 = position_axis_1 + radius + 1

    if position_axis_0 - radius < 0:
        start_point_axis_0 = 0
    elif position_axis_0 - radius >= self.__targets.shape[0]:
        start_point_axis_0 = self.__targets.shape[0]

    if position_axis_0 + radius < 0:
        end_point_axis_0 = 0
    elif position_axis_0 + radius >= self.__targets.shape[0]:
        end_point_axis_0 = self.__targets.shape[0]

    if position_axis_1 - radius < 0:
        start_point_axis_1 = 0
    elif position_axis_1 - radius >= self.__targets.shape[1]:
        start_point_axis_1 = self.__targets.shape[1]

    if position_axis_1 + radius < 0:
        end_point_axis_1 = 0
    elif position_axis_1 + radius >= self.__targets.shape[1]:
        end_point_axis_1 = self.__targets.shape[1]

    return slice(start_point_axis_0, end_point_axis_0),
           slice(start_point_axis_1, end_point_axis_1)
```

Per ogni asse, calcola gli intervalli in cui un elemento rettangolare è interno alla mappa.

Parametri

- position_axis_0 : *int*

  Posizione dell'elemento sull'asse 0 (righe della matrice).

- position_axis_1: *int*

  Posizione dell'elemento sull'asse 1 (colonne della matrice).

- radius : *int*

  Raggio dell'elemento.

Restituisce

- interval_axis_0 : *slice object*

  Intervallo sull'asse 0 in cui l'elemento è interno alla mappa, rappresentato nel formato slice(start, stop).

- interval_axis_1 : *slice object*

  Intervallo sull'asse 1 in cui l'elemento è interno alla mappa, rappresentato nel formato slice(start, stop).

__detect_collision()

```
1.  def __detect_collision(self, batch_index):
2.      collision_level = self.__collision[np.newaxis, ...]
3.      collision_detection = np.append(self.__drawn_drones[batch_index],
4.                                      collision_level, axis=0)
5.      collision_detection = np.sum(collision_detection, axis=0)
6.
7.      if np.any(collision_detection > 1):
8.          return True
9.
10.     return False
```

Verifica se è presente una qualsiasi collisione tra droni oppure tra droni e livelli della mappa abilitati alle collisioni. Il metodo viene usato dopo lo spostamento di un singolo drone: se la situazione di partenza è priva di collisioni, è univocamente determinabile quale drone ha colliso senza l'utilizzo di indici identificativi.

Parametri

- batch_index : *int*

  Indice della simulazione parallela su cui effettuare l'operazione.

Restituisce

- collision_detected : *bool*

  Determina se è stata individuata una collisione (True) o meno (False).

__out_of_map()

```
def __out_of_map(self, batch_index, drone_index, radius):
    position_axis_0 = self.__drones_position[batch_index, drone_index, 0]
    position_axis_1 = self.__drones_position[batch_index, drone_index, 1]

    if (position_axis_0 - radius < 0 or
          position_axis_0 -radius >= self.__targets.shape[0] or
            position_axis_0 + radius < 0 or
              position_axis_0 + radius >= self.__targets.shape[0]):
        return True

    if (position_axis_1 - radius < 0 or
          position_axis_1 - radius >= self.__targets.shape[1] or
            position_axis_1 + radius < 0 or
              position_axis_1 + radius >= self.__targets.shape[1]):
        return True

    return False
```

Verifica se un drone si trova al di fuori dei confini della mappa (parzialmente o completamente).

Parametri

- batch_index : *int*

  Indice della simulazione parallela su cui effettuare l'operazione.

- drone_index : *int*

  Indice del drone su cui effettuare la verifica.

- radius : *int*

  Raggio del drone.

Restituisce

- out_of_map : *bool*

  Determina se il drone si trova parzialmente o completamente fuori dalla mappa (True) oppure se si trova completamente dentro la mappa (False).

__init_stigmergy_space()

```
def __init_stigmergy_space(self):
    self.__stigmergy_space = np.zeros((self.__batch_size,
                                       self.__stigmergy_evaporation_speed.shape[0],
                                       self.__targets.shape[0],
                                       self.__targets.shape[1]),
                                       int)
```

Inizializza lo spazio stigmergico: gli elementi che compongono le mappe stigmergiche sono interi, in modo tale da evitare problemi di visualizzazione dovuti all'imprecisione della rappresentazione dei float (in particolare, nel meccanismo di evaporazione).

__init_rendering_parameters()

```
def __init_rendering_parameters(self):
    self.__image = np.full((self.__targets.shape[0], self.__targets.shape[1], 3), 0)
    self.__image_semaphore = None

    if self.__rendering_allowed:
        if self.__batch_size > 1:
         raise ValueError("Rendering is allowed only when batch_size is equal to 1")

        self.__image_semaphore = threading.Lock()
        rendering = threading.Thread(target=self.__init_rendering)
        rendering.start()
```

Inizializza i parametri necessari per il rendering e genera un thread che disegna a video.

Lancia

- ValueError

  L'eccezione è lanciata nel caso in cui il numero delle simulazioni parallele sia maggiore di 1 e il rendering sia abilitato.

__init_rendering()

```
def __init_rendering(self):
    app = QtGui.QApplication([])

    w = pg.GraphicsView()
    w.show()
    w.showMaximized()
    w.setWindowTitle('Drone simulator')

    view = pg.ViewBox()
    view.invertY()
    view.setAspectLocked(True)
    w.setCentralItem(view)

    self.__image_semaphore.acquire()
    img = pg.ImageItem(self.__image)
    view.addItem(img)
    self.__image_semaphore.release()

    timer = QtCore.QTimer()
    timer.timeout.connect(lambda: self.__update_rendering(view))
    timer.start(100)

    app.instance().exec()
```

Inizializza la finestra in cui avverrà il rendering e predispone un timer di aggiornamento dello stesso.

__update_rendering()

```
def __update_rendering(self, view):
    self.__image_semaphore.acquire()

    view.clear()
    img = pg.ImageItem(self.__image)
    view.addItem(img)

    self.__image_semaphore.release()
```

Aggiorna l'immagine mostrata a video: l'immagine cambia effettivamente se, in precedenza, è stata chiamata la funzione render().

Parametri

- view : *ViewBox*

  Elemento dell'interfaccia grafica su cui effettuare l'aggiornamento dell'immagine.

step()

```
def step(self, drones_actions, stigmergy_actions):
    self.__current_steps += 1
    if self.__current_steps > self.__max_steps:
        return None, None, True, "Maximum number of steps reached"

    observation_dimension = 2*(self.__drone_size + self.__observation_range) + 1
    observations_table = np.zeros((self.__batch_size, self.__amount_of_drones,
                                   observation_dimension, observation_dimension))

    rewards_table = np.zeros((self.__batch_size, self.__amount_of_drones, 1))

    self.__stigmergy_evaporation()
    self.__update_stigmergy_space(stigmergy_actions)
    self.__update_drones(drones_actions, rewards_table, observations_table)

    return observations_table, rewards_table, False, self.__environment_info()
```

Il metodo compie un passo della simulazione, ovvero effettua nell'ordine:

- l'evaporazione delle tracce stigmergiche presenti;
- il rilascio di nuove tracce stigmergiche (nella posizione non aggiornata dei droni, ovvero quella risultante dallo step precedente);
- lo spostamento dei droni, con il calcolo della ricompensa e dello spazio di osservazione per ciascun drone.

Parametri

- drones_actions : *ndarray (3D)*

  Array contenente gli spostamenti che i droni devono effettuare in ciascuna simulazione parallela.

  L'array fornito deve essere di dimensioni pari a shape = (batch_size, amount_of_drones, 2), dove il 2 è dovuto al fatto che il comando da impartire esprime uno spostamento bidimensionale: il primo elemento si riferisce all'asse 0 (righe), il secondo si riferisce all'asse 1 (colonne).

  Ad esempio, si potrebbe avere:

```
# batch_size = 1, amount_of_drones = 3
shape = (1, 3, 2)
actions = np.array([[[1,0],[0,1],[-3,5]]])
```

```
# batch_size = 4, amount_of_drones = 10
shape = (4, 10, 2)
actions = np.random.randint(7, size=shape)
```

- stigmergy_actions : *ndarray (3D)*

  Array contenente le azioni che i droni devono effettuare all'interno dello spazio stigmergico di ciascuna simulazione parallela.

  L'array fornito deve essere di dimensioni pari a shape = (batch_size, amount_of_drones, 2), dove il 2 è dovuto al fatto che il comando da impartire deve specificare:

  - in prima posizione, il livello su cui effettuare l'azione (quale feromone rilasciare), a partire dal livello zero fino al livello massimo diminuito di 1 (dipende dal numero di feromoni impostato in fase di inizializzazione). Se non si vuole rilasciare un feromone, si imposta il livello pari a -1;
  - in seconda posizione, il raggio che determina l'ampiezza della traccia da rilasciare (deve essere maggiore o uguale a 0).

Restituisce

- observation : *ndarray (4D)*

  Array contenente, per ogni simulazione parallela, lo spazio di osservazione di ciascun drone. Lo spazio è composto da elementi pari a:

  - 0 se la casella della mappa è occupata dal solo terreno;
  - 1 se la casella della mappa è occupata da un drone, da un obiettivo, da un ostacolo o da una traccia stigmergica;
  - -1 se la casella contiene un elemento fuori dalla mappa.

  Nel caso in cui siano stati effettuati un numero di step pari a max_steps, assume il valore None.

- reward : *ndarray (3D)*

  Array che, per ogni simulazione parallela, contiene la ricompensa ottenuta da ciascun drone in seguito allo spostamento effettuato con il comando corrente; la ricompensa è fornita sotto forma di float. Nel caso in cui siano stati effettuati un numero di step pari *a max_steps*, assume il valore None.

- done : *bool*

  Determina se la simulazione è finita e l'ambiente necessita di un reset (True) o meno (False). La simulazione termina quando è stato raggiunto il massimo numero di step previsti.

- info : *dict*

  Raccolta di informazioni sui droni e sull'ambiente di ciascuna simulazione. Nello specifico, sono presenti le seguenti chiavi: *"Drones position - float", "Drones position", "Drones velocity", "Targets achieved", "Stigmergy Space"*.

  Nel caso in cui siano stati effettuati un numero di step pari a max_steps, viene restituita una stringa *"Maximum number of steps reached"*.

__stigmergy_evaporation()

```
1.  def __stigmergy_evaporation(self):
2.      evaporation_levels = np.zeros((self.__stigmergy_evaporation_speed.shape[0],
3.                                     self.__targets.shape[0],
4.                                     self.__targets.shape[1]),
5.                                     int)
6.
7.      for index in range(self.__stigmergy_evaporation_speed.shape[0]):
8.       evaporation_levels[index] = np.full((self.__targets.shape[0],
9.                                            self.__targets.shape[1]),
10.                                           self.__stigmergy_evaporation_speed[index])
11.
12.     for batch_index in range(self.__batch_size):
13.         self.__stigmergy_space[batch_index] -= evaporation_levels
14.
15.     self.__stigmergy_space[self.__stigmergy_space < 0] = 0
```

Gestisce il meccanismo di evaporazione dello spazio stigmergico, effettuando un aggiornamento su tutte le simulazioni parallele.

Un drone, mediante il rilascio di un tipo di feromone, genera una traccia stigmergica: ad ogni step della simulazione, tale valore viene decrementato in base alla velocità di evaporazione definita per il livello in fase di inizializzazione. L'intensità di rilascio è sempre pari a 100 e diminuirà di un valore pari alla velocità di evaporazione ad ogni step; ad esempio

	Velocità di evaporazione: 20	**Velocità di evaporazione: 30**	**Velocità di evaporazione: 50**
Intensità iniziale	100	100	100
Intensità dopo 1 step	80	70	50
Intensità dopo 2 step	60	40	0
Intensità dopo 3 step	40	10	0

La diminuzione dell'intensità non si traduce in una maggiore o minore intensità del colore, ma influisce solamente sul tempo di permanenza della traccia.

Nel caso siano effettuati più rilasci di feromone nello stesso punto, l'intensità aumenta ad incrementi di 100, ma l'evaporazione causa sempre una diminuzione pari alla velocità impostata (il meccanismo di evaporazione rimane inalterato).

	Velocità di evaporazione: 20	**Velocità di evaporazione: 20**	**Velocità di evaporazione: 20**
Intensità iniziale	100	200	300
Intensità dopo 1 step	80	180	280
Intensità dopo 2 step	60	160	260
Intensità dopo 3 step	40	140	240

L'utilizzo di numeri interi è necessario per evitare problemi di precisione dovuti ai float: ad esempio, nell'effettuare una sottrazione, si potrebbe ottenere un numero molto piccolo, ma non pari a 0, causando un permanere non voluto della traccia stigmergica e una inaffidabilità del meccanismo stesso.

__update_stigmergy_space()

```
def __update_stigmergy_space(self, stigmergy_actions):
    for batch_index in range(self.__batch_size):
        for drone_index in range(self.__amount_of_drones):
            position_axis_0 = self.__drones_position[batch_index, drone_index, 0]
            position_axis_1 = self.__drones_position[batch_index, drone_index, 1]
            stig_level = int(stigmergy_actions[batch_index, drone_index, 0])
            stig_radius = int(stigmergy_actions[batch_index, drone_index, 1])

            if stig_level == -1:  # No pheromone release
                continue

            interval_axis_0, interval_axis_1 = self.__drawing_boundaries(
                                                                    position_axis_0,
                                                                    position_axis_1,
                                                                    stig_radius)

            # 100 is the intensity of the pheromone released.
            # It is the same for all the drones and for every pheromone type.
            self.__stigmergy_space[batch_index][stig_level][interval_axis_0,
                                                            interval_axis_1] += 100
```

Aggiorna lo spazio stigmergico di tutte le simulazioni implementando il rilascio del feromone; ciascun feromone è rilasciato sempre con un'intensità pari a 100. Anche in questo caso, l'utilizzo di numeri interi è necessario per evitare problemi di precisione dovuti ai float.

Parametri

- stigmergy_actions: *ndarray (3D)*

  Array contenente le azioni che i droni devono effettuare all'interno dello spazio stigmergico di ciascuna simulazione parallela.

__update_drones()

```
1.  def __update_drones(self, drones_actions, rewards_table, observations_table):
2.      for batch_index in range(self.__batch_size):
3.          for drone_index in range(self.__amount_of_drones):
4.              self.__update_velocity(batch_index, drone_index, drones_actions)
5.              self.__update_position(batch_index, drone_index)
6.              self.__draw_drone(batch_index, drone_index)
7.              self.__target_achieved(batch_index, drone_index)
8.              rewards_table[batch_index, drone_index] = self.__reward(batch_index,
9.                                                                      drone_index)
10.             observations_table[batch_index,
11.                                drone_index] = self.__get_observation(batch_index,
12.                                                                       drone_index)
```

Per ogni simulazione, aggiorna lo stato dei droni calcolando le nuove velocità e posizioni; inoltre, esegue il calcolo dei punteggi e degli spazi di osservazione.

Parametri

- drones_actions : *ndarray (3D)*

  Array contenente gli spostamenti che i droni devono effettuare in ciascuna simulazione parallela.

- rewards_table : *ndarray (3D)*

  Array contenente, per ogni simulazione parallela, la ricompensa ottenuta da ciascun drone in seguito allo spostamento effettuato dovuto al comando corrente. La ricompensa è fornita sotto forma di float.

- observations_table : *ndarray (3D)*

  Array contenente, per ogni simulazione parallela, lo spazio di osservazione di ciascun drone.

__update_velocity()

```
1.  def __update_velocity(self, batch_index, drone_index, drones_actions):
2.      drone_velocity = self.__drones_velocity[batch_index, drone_index]
3.      drone_command = drones_actions[batch_index, drone_index]
4.
5.      self.__drones_velocity[batch_index, drone_index] =
6.                   (drone_velocity * self.__inertia +
7.                    drone_command * (1 - self.__inertia))
```

Dato un drone in una simulazione, calcola la nuova velocità a partire dal comando impartito. La velocità è calcolata secondo la formula:

$$Velocità = velocità \cdot inerzia + comando \cdot (1 - inerzia)$$

Parametri

- batch_index : *int*

  Indice della simulazione parallela su cui effettuare l'operazione.

- drone_index : *int*

  Indice del drone su cui effettuare il calcolo.

- drones_actions : *ndarray (3D)*

  Array contenente gli spostamenti che i droni devono effettuare in ciascuna simulazione parallela.

__update_position ()

```
def __update_position(self, batch_index, drone_index):
    drone_position = self.__drones_position_float[batch_index, drone_index]
    drone_velocity = self.__drones_velocity[batch_index, drone_index]
    t_constant = 1

    self.__drones_position_float[batch_index, drone_index] = drone_position +
drone_velocity * t_constant
    self.__drones_position = np.copy(self.__drones_position_float).astype(int)
```

Dato un drone in una simulazione, calcola la nuova posizione a partire dalla velocità corrente. La posizione è calcolata secondo la formula:

$$Posizione = posizione + velocità \cdot t$$

dove t è una costante di tempo, per semplicità considerata pari a 1.

Parametri

- batch_index : *int*

  Indice della simulazione parallela su cui effettuare l'operazione.
- drone_index : *int*

  Indice del drone su cui effettuare il calcolo.

__target_achieved()

```
def __target_achieved(self, batch_index, drone_index):
    target_collision = self.__drawn_drones[batch_index, drone_index] + self.__targets
    self.__targets_achieved[batch_index][drone_index][target_collision > 1] = 1
```

Dato un drone in una simulazione, verifica se è stato raggiunto un obiettivo: in caso affermativo, lo aggiunge alla matrice degli obiettivi trovati da quel particolare drone.

Parametri

- batch_index : *int*

  Indice della simulazione parallela su cui effettuare l'operazione.
- drone_index : *int*

  Indice del drone su cui effettuare la verifica.

__reward()

```
1.  def __reward(self, batch_index, drone_index):
2.      average_distance = self.__average_distance_to_targets(batch_index, drone_index)
3.      alpha = 0.5
4.      num_target_achieved = np.count_nonzero(self.__targets_achieved[batch_index,
5.                                                                     drone_index])
6.
7.      reward_score = 1 / average_distance + alpha * num_target_achieved
8.
9.      if self.__detect_collision(batch_index):
10.         return reward_score * (-1)
11.
12.     if self.__out_of_map(batch_index, drone_index, self.__drone_size):
13.         return reward_score * (-2)
14.
15.     return reward_score
```

Dato un drone in una simulazione, calcola la ricompensa ottenuta dal drone stesso. Assumendo che ogni obiettivo occupi una sola casella, definiamo il punteggio base come:

$$Punteggio = \frac{1}{distanza\ media} + \alpha \cdot obiettivi\ trovati$$

dove $distanza\ media$ è la distanza media del drone da tutti gli obiettivi non ancora trovati, $obiettivi\ trovati$ è il numero di obiettivi trovati fino a quel momento e α è un coefficiente di proporzionalità pari a 0.5.

Distribuiamo la ricompensa nel seguente modo:

- se il drone non collide e non esce dalla mappa, riceve una ricompensa pari a $Punteggio$;
- se il drone collide, riceve una ricompensa pari a $-Punteggio$;
- se il drone esce dalla mappa (parzialmente o completamente), riceve una ricompensa pari a $-2 \cdot Punteggio$.

Parametri

- batch_index : *int*

  Indice della simulazione parallela su cui effettuare l'operazione.
- drone_index : *int*

  Indice del drone su cui effettuare il calcolo.

Restituisce

- reward_score : *float*

  Ricompensa ottenuta dal drone.

__average_distance_to_targets()

```
def __average_distance_to_targets(self, batch_index, drone_index):
    targets_not_achieved = self.__targets + self.__targets_achieved[batch_index,
                                                                     drone_index]
    targets_not_achieved[targets_not_achieved > 1] = 0
    average_distance = 1

    if np.any(targets_not_achieved):
       targets_positions = np.array(list(zip(*np.nonzero(targets_not_achieved))))
        drone_position = np.repeat(self.__drones_position[batch_index,
                                                          drone_index].reshape(1, 2),
                                   targets_positions.shape[0], 0)

        average_distance = np.average(np.sqrt(np.sum(
                np.power(np.subtract(drone_position, targets_positions), 2), 1)))

    return average_distance
```

Dato un drone in una simulazione, calcola la distanza media tra il drone e gli obiettivi non ancora raggiunti.

Parametri

- batch_index : *int*

  Indice della simulazione parallela su cui effettuare l'operazione.

- drone_index : *int*

  Indice del drone su cui effettuare il calcolo.

Restituisce

- average_distance : *float*

  Distanza media del drone dagli obiettivi non raggiunti. Nel caso in cui il drone abbia raggiunto tutti gli obiettivi, è pari a 1.

__get_observation()

```
def __get_observation(self, batch_index, drone_index):
    position_axis_0 = self.__drones_position[batch_index, drone_index, 0]
    position_axis_1 = self.__drones_position[batch_index, drone_index, 1]
    observation_radius = self.__drone_size + self.__observation_range

    drones = np.sum(self.__drawn_drones[batch_index], axis=0)
    stigmergy_space = np.sum(self.__stigmergy_space[batch_index], axis=0)
    environment = self.__targets + self.__collision + drones + stigmergy_space
    environment[environment > 1] = 1

    if self.__out_of_map(batch_index, drone_index, observation_radius):
      # Map view enlargement: the drone will see -1 if a space is outside the map
      # The position of the drone is reevaluated according to the new map dimensions

        environment, position_axis_0, position_axis_1 =
                                            self.__enlarge_map_view(position_axis_0,
                                                                    position_axis_1,
                                                                    observation_radius,
                                                                    environment)

    observation = environment[position_axis_0 - observation_radius:
                              position_axis_0 + observation_radius + 1,
                              position_axis_1 - observation_radius:
                              position_axis_1 + observation_radius + 1]

    return observation
```

Dato un drone in una simulazione, calcola il suo spazio di osservazione. Nel caso in cui lo spazio di osservazione sia composto da spazi fuori dalla mappa, esegue l'allargamento della mappa stessa.

Parametri

- batch_index : *int*

  Indice della simulazione parallela su cui effettuare l'operazione.

- drone_index : *int*

  Indice del drone su cui effettuare il calcolo.

Restituisce

- observation : *ndarray (2D)*

  Array bidimensionale contenente lo spazio di osservazione di un drone.

  Lo spazio è composto da elementi pari a:

  - 0 se la casella della mappa è occupata dal solo terreno;
  - 1 se la casella della mappa è occupata da un drone, da un obiettivo, da un ostacolo o da una traccia stigmergica;
  - -1 se la casella contiene un elemento fuori dalla mappa.

__enlarge_map_view()

```
1.  def __enlarge_map_view(self, position_axis_0, position_axis_1,
2.                             observation_radius, environment):
3.
4.      enlargement_axis_before_0 = 0
5.      enlargement_axis_after_0 = 0
6.      enlargement_axis_before_1 = 0
7.      enlargement_axis_after_1 = 0
8.
9.      if position_axis_0 - observation_radius < 0:
10.         enlargement_axis_before_0 = abs(position_axis_0 - observation_radius)
11.
12.     if position_axis_0 + observation_radius >= self.__targets.shape[0]:
13.         enlargement_axis_after_0 = position_axis_0 + observation_radius -
14.                                                 (self.__targets.shape[0] - 1)
15.
16.     if position_axis_1 - observation_radius < 0:
17.         enlargement_axis_before_1 = abs(position_axis_1 - observation_radius)
18.
19.     if position_axis_1 + observation_radius >= self.__targets.shape[1]:
20.         enlargement_axis_after_1 = position_axis_1 + observation_radius -
21.                                                   (self.__targets.shape[1] - 1)
22.
23.     enlarged_map = np.pad(environment, [(enlargement_axis_before_0,
24.                                          enlargement_axis_after_0),
25.                                         (enlargement_axis_before_1,
26.                                          enlargement_axis_after_1)],
27.                           constant_values=(-1))
28.
29.     return enlarged_map, position_axis_0 + enlargement_axis_before_0,
30.                           position_axis_1 + enlargement_axis_before_1
```

Ingrandisce le dimensioni della mappa fornita: sono aggiunti spazi esterni inizializzati a -1, ma solamente dove necessario (lo sforamento della mappa è considerato in base alle coordinate del punto e al raggio di osservazione forniti).

Parametri

- position_axis_0 : *int*

  Posizione del drone sull'asse 0 (righe).

- position_axis_1 : *int*

  Posizione del drone sull'asse 1 (colonne).

- observation_radius : *int*

  Raggio necessario per stabilire lo sforamento della mappa. Si osservi che il raggio è applicato a partire dalle coordinate fornite e deve comprendere la dimensione del drone se si vuole che sia considerata.

- environment : *ndarray (2D)*

  Array bidimensionale che rappresenta la mappa da ingrandire.

Restituisce

- enlarged_map : *ndarray (2D)*

  Array bidimensionale che rappresenta la mappa espansa.

- new_position_axis_0 : *int*

  Posizione del drone sull'asse 0 (righe) nel sistema di riferimento della nuova mappa.

- new_position_axis_1 : *int*

  Posizione del drone sull'asse 1 (colonne) nel sistema di riferimento della nuova mappa.

__environment_info()

```
1.  def __environment_info(self):
2.      info = {
3.          "Drones position - float": self.__drones_position_float,
4.          "Drones position": self.__drones_position,
5.          "Drones velocity": self.__drones_velocity,
6.          "Targets achieved": self.__targets_achieved,
7.          "Stigmergy Space":  self.__stigmergy_space
8.      }
9.
10.     return info
```

Costruisce un dizionario contenente informazioni aggiuntive sull' ambiente e sui droni di ciascuna simulazione.

Restituisce

- info : *dict*

  Raccolta di informazioni sui droni e sull'ambiente di ciascuna simulazione.

render()

```
1.  def render(self):
2.      if self.__rendering_allowed:
3.          environment = np.copy(self.__environment_bitmap)
4.          drones = np.sum(self.__drawn_drones[0], axis=0)
5.          stigmergy_space = self.__stigmergy_space[0]
6.
7.          environment[drones == 1, :] = environment[drones == 1, :] +
8.                                        self.__drone_colour
9.          environment[environment > 255] = environment[environment > 255]//2
10.
11.         for index in range(stigmergy_space.shape[0]):
12.             environment[stigmergy_space[index] > 0, :] +=
13.                                                  self.__stigmergy_colours[index]
14.             environment[environment > 255] = environment[environment > 255]//2
15.
16.         self.__image_semaphore.acquire()
17.         np.copyto(self.__image, environment)
18.         self.__image_semaphore.release()
```

Se il rendering è abilitato, aggiorna il contenuto dell'immagine utilizzata per il disegno a video, aggiungendo le informazioni dinamiche (droni e tracce stigmergiche) all'ambiente fisso. La chiamata a questo metodo non modifica direttamente il disegno a video: ciò è effettuato da un timer interno allo strumento grafico.

reset()

```
def reset(self):
    observation_dimension = 2 * (self.__drone_size + self.__observation_range) + 1
    observations_table = np.zeros((self.__batch_size, self.__amount_of_drones,
                                   observation_dimension, observation_dimension))

    self.__init_drones_parameters()
    self.__init_drones()
    self.__init_stigmergy_space()

    # Initial observation
    for batch_index in range(self.__batch_size):
        for drone_index in range(self.__amount_of_drones):
            observations_table[batch_index, drone_index] =
                                   self.__get_observation(batch_index, drone_index)

    return observations_table
```

Reimposta l'ambiente tramite la reinizializzazione dei droni e dello spazio stigmergico, in modo tale che possa essere avviata una nuova simulazione.

Restituisce

- observation : *ndarray (4D)*

  Array contenente, per ogni simulazione parallela, lo spazio di osservazione iniziale di ciascun drone. Lo spazio è composto da elementi pari a:

  - 0 se la casella della mappa è occupata dal solo terreno;
  - 1 se la casella della mappa è occupata da un drone, da un obiettivo, da un ostacolo o da una traccia stigmergica;
  - -1 se la casella contiene un elemento fuori dalla mappa.

6.3 Esecuzione di una simulazione

Dopo aver importato il package contenente la classe DroneSimulator, è necessario inizializzare un oggetto fornendo i parametri della simulazione; ad esempio:

```
env = DroneSimulator(
    bitmap = './maps/test-with-2-levels.bmp',
    batch_size = 1,
    observation_range = 5,
    drone_size = 0,
    amount_of_drones = 3,
    stigmergy_evaporation_speed = np.array([10, 20, 30]),
    stigmergy_colours = np.array([[255, 64, 0],[255, 128, 0],[255, 255, 0]]),
    inertia = 0,
    collision_detection = np.array([True, False]),
    max_steps = 1000,
    rendering_allowed = True
)
```

Successivamente, è possibile utilizzare l'ambiente con un semplice ciclo for, in modo analogo a quanto visto con OpenaAI Gym; supponendo per semplicità di eseguire sempre le stesse azioni ad ogni passo, si ha

```
stig_actions = np.array([[[0, 1], [1, 2], [2, 3]]])
actions = np.array([[[1, 0], [0, 1], [-1, 0]]])

for i in range(5):
    obs, rew, done, info = env.step(actions, stig_actions)
    env.render()
```

dove actions contiene i movimenti da eseguire nello spazio bidimensionale e stig_actions contiene le azioni da eseguire nello spazio stigmergico.

Il risultato a video è simile al seguente:

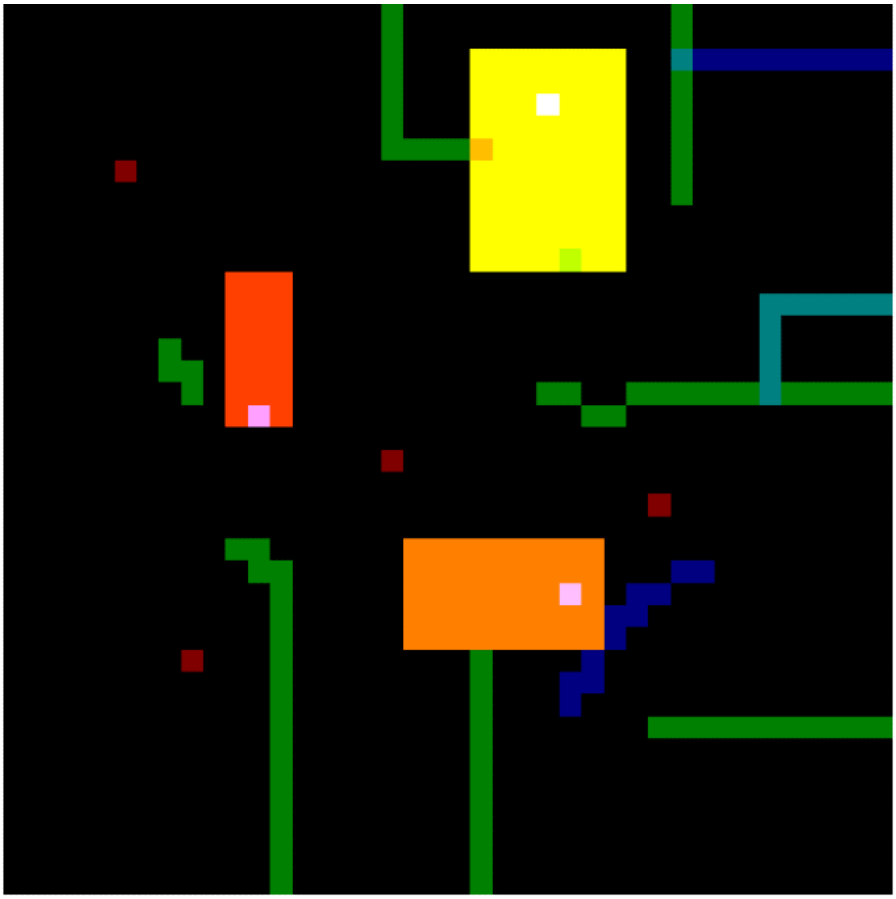

Figura 6.2.3.2 Esempio di schermata di simulazione.

dove la posizione iniziale dei droni, rappresentati in bianco, è determinata casualmente all'inizio della simulazione.

I colori giallo, arancione chiaro e arancione scuro rappresentano le tracce stigmergiche dei diversi feromoni, mentre i colori rosso, verde e blu (con eventuali sovrapposizioni) rappresentano rispettivamente gli obiettivi, il primo ed il secondo livello degli ostacoli.

7 CONCLUSIONI

Nello svolgimento della tesi è stato illustrato il percorso di progettazione, documentazione e sviluppo dell'ambiente di simulazione per droni: adatto all'addestramento di reti neurali artificiali secondo il paradigma di apprendimento per rinforzo, l'ambiente permette di simulare la ricerca di bersagli in una mappa con l'impiego di una flotta di droni, i quali interagiscono con meccanismi di sciame basati sul rilascio di feromoni digitali.

Essendo uno strumento di base, la sua effettiva bontà dovrà essere verificata attraverso un vero processo di addestramento: sarà cruciale stabilire se il programma fornisce dati in numero sufficiente e se la granularità del rinforzo è tale da garantire un apprendimento efficace e rapido.

Un ulteriore sviluppo futuro potrebbe vedere il codice prodotto come base per la progettazione di un ambiente di simulazione per agenti generici, con supporto per un insieme più vario di funzioni di rinforzo.

Infine, essendo progettato per l'esecuzione di più simulazioni indipendenti contemporaneamente, il simulatore potrebbe necessitare di miglioramenti in termini di performance. Il metodo più oneroso è rappresentato da __update_drones(), la cui velocità di esecuzione diminuisce all'aumentare del numero di simulazioni e del numero di droni che compongono la flotta: prima di ripensarne l'intera struttura, è possibile tentare un approccio meno invasivo avvalendosi di un compilatore JIT di supporto (ad es. Numba) o traducendo il codice in Cython (linguaggio di programmazione sovrainsieme di Python, progettato per fornire prestazioni simili al C).

BIBLIOGRAFIA E SITOGRAFIA

Alfeo L., et al. *Studio del comportamento di uno sciame di droni attraverso un ambiente di simulazione visuale.* Università di Pisa, 2016.

Araujo dos Santos, L. *Artificial Intelligence.* s.d. https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/linear_algebra.html (consultato il giorno settembre 9, 2019).

Asperti, A. *An introduction to Neural Networks and Deep Learning.* Department of Mathematics, University of Bologna, 2018.

Bonabeau, E., M. Dorigo, e G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems.* Oxford University Press, 1999.

Burnett, Colin M.L. *An example artificial neural network with a hidden layer.* 27 dicembre 2006. https://upload.wikimedia.org/wikipedia/commons/e/e4/Artificial_neural_network.svg (consultato il giorno settembre 8, 2019).

Casu, D. *DroneSimulator.* 2019. https://github.com/diegocasu/DroneSimulator (consultato il giorno settembre 9, 2019).

DoITPoMS, Dissemination of IT for the Promotion of Materials Science. *Tensors in Materials Science.* University of Cambridge. s.d. https://www.doitpoms.ac.uk/tlplib/tensors/index.php (consultato il giorno settembre 9, 2019).

Dréo, J. *Shortest path find by an ant colony.* 27 maggio 2006. https://upload.wikimedia.org/wikipedia/commons/a/af/Aco_branches.svg (consultato il giorno settembre 7, 2019).

Kaplan, A., e M. Haenlein. «Siri, Siri, in my hand: Who's the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence.» *Business Horizons. 62,* gennaio 2019: 15–25.

Kennedy, J., e R. C. Eberhart. *Swarm Intelligence.* Morgan Kaufmann Publishers, 2001.

Lanzi, P. L. *Apprendimento per rinforzo.* Metodologie per Sistemi Intelligenti, Laurea in Ingegneria Informatica, Polo regionale di Como: Politecnico di Milano, s.d.

Leung, K., e L. Chew. *Learning Paradigms in Neural Networks.* 28 maggio 2019. https://medium.com/swlh/learning-paradigms-in-neural-networks-30854975aa8d (consultato il giorno settembre 8, 2019).

Minutella, F. 2018/2019a. https://github.com/minu7/drone-simulator (consultato il giorno luglio 9, 2019).

Minutella, F. *Modellazione in algebra tensoriale di uno sciame di droni alla ricerca di target e realizzazione su framework Python di un ambiente per il reinforcement learning della strategia di esplorazione.* Tesi di laurea, Università di Pisa, 2018/2019b.

Nicholson, C. *A Beginner's Guide to Neural Networks and Deep Learning.* s.d. https://skymind.ai/wiki/neural-network (consultato il giorno settembre 8, 2019).

NumPy developers. *About NumPy.* s.d. https://numpy.org/ (consultato il giorno settembre 9, 2019).

OpenAI. *Gym.* s.d. http://gym.openai.com/; http://gym.openai.com/docs (consultato il giorno settembre 9, 2019).

Redazione Osservatori Digital Innovation. *Alla scoperta del Deep Learning: significato, esempi e applicazioni.* Politecnico di Milano. 19 aprile 2019. https://blog.osservatori.net/it_it/deep-learning-significato-esempi-applicazioni (consultato il giorno settembre 8, 2019).

Rossant, C. *Understanding the internals of NumPy to avoid unnecessary array copying.* s.d. https://ipython-books.github.io/45-understanding-the-internals-of-numpy-to-avoid-unnecessary-array-copying/ (consultato il giorno settembre 10, 2019).

Scotti, F., e D. Sana. *Corso di reti neurali: approfondimento.* Università di Milano, s.d.

Silver, D. *Deep Reinforcement Learning.* 17 giugno 2016. https://deepmind.com/blog/article/deep-reinforcement-learning (consultato il giorno settembre 9, 2019).

Watkins, C.J.C.H. *Learning from Delayed Rewards.* PhD thesis, Cambridge University, 1989.

Zhou, V. *Machine Learning for Beginners: An Introduction to Neural Networks.* 5 marzo 2019. https://towardsdatascience.com/machine-learning-for-beginners-an-introduction-to-neural-networks-d49f22d238f9 (consultato il giorno settembre 9, 2019).