# Università degli studi di Torino

## Dipartimento di Informatica


Intelligenza Artificiale e Laboratorio

Relazione di progetto

Soar

Studente: Lorenzo Botto a.a. 2021/2022

# Indice

# 1. Descrizione implementazione

Per l'implementazione del sistema intelligente in Soar parte da una situazione in cui si trova in una stanza ed ha a disposizione degli oggetti. Deve trovare il modo di scappare, scardinando una finestra, e:

- può muoversi da un oggetto verso l'altro;
- può prendere degli oggetti in mano quando è vicino a loro;
- può posare un oggetto che ha in mano;
- può combinare gli oggetti;
- può scombinare gli oggetti combinati precedentemente;
- può prendere i due tronchi e posizionarli vicini all'uscita;
- dopo che ha combinato la fionda può caricare la fionda e colpire la finestra;
- una volta che la finestra è rotta (5 colpi) e ha posizionato i due tronchi può arrampicarsi e scappare dalla stanza.

Il sistema utilizza il meccanismo di reward di Soar in grado di far apprendere il sistema tramite reinforcement learning. Verrà spiegato in seguito.

Il sistema lavora come segue:

- proposta di azione, quindi vengono assegnati all'agente tutti gli operatori applicabili in un determinato stato;
- applicazione di un azione a seconda del reward;
- elaborazione del reward a seconda dell'azione che è stata scelta.

Queste tre operazioni vengono ripetute fino a quando l'agente non riesce a scappare dalla stanza, il quale è il nostro goal.

# 2. Azioni

## 2.1 Movimento

L'agente può muoversi da e verso gli oggetti. Una volta che si troverà vicino ad un oggetto potrà prendere in mano l'oggetto.
Inoltre, l'agente può muoversi verso l'uscita, dove si trova la finestra.

La proposta di movimento ha due precondizioni:

- l'oggetto deve esistere, ovvero che non ho quell'oggetto in mano altrimenti non posso muovermi verso quell'oggetto;
- se mi trovo già vicino ad un oggetto, sono già vicino a quell'oggetto e non ha senso proporre il movimento.

Questo non vale per l'uscita, dove potrà muoversi sempre, a meno che non si trovi già vicino all'uscita.

L'applicazione del movimento modifica la posizione del robot e l'ultima azione dello stato.

## 2.2 Presa oggetto

Quando l'agente è vicino ad un certo oggetto può prenderlo in mano.

La proposta di presa oggetti ha delle precondizioni:

- i bracci occupati devono essere < 2, quindi che ce ne sia almeno uno libero;
- l'oggetto a cui si trova vicino il robot esista, ovvero che non ce l'abbia già in mano.

L'applicazione dell'operatore di presa dell'oggetto modifica:

- il numero di bracci occupati;
- il braccio avrà un certo oggetto in mano;
- l'oggetto non esisterà più, così non si può più prendere in mano;
- lo stato cambia l'ultima azione effettuata.

Per quanto riguarda i tronchi c'è un operatore a parte, perché siccome i tronchi sono più grandi servono due braccia libere per poterlo prendere in mano e si controlla anche che non sia già stato posizionato all'uscita sennò non si può prendere in mano e spostarlo. Inoltre, quando viene applicato l'operatore tutte le due braccia diventano occupate.

## 2.3 Posa oggetto

Quando l'agente ha un oggetto in mano può decidere di posarlo.

La proposta di posa dell'oggetto ha la precondizione che almeno una delle due braccia abbia un oggetto in mano.
Quando si applica l'operatore, il braccio che aveva l'oggetto diventa libero, l'oggetto tornerà ad esistere per i movimenti e per prenderlo in mano. Inoltre, viene modificato lo stato con l'ultima azione effettuata.

Per quanto riguarda il tronco, anche qui, ci sono gli operatori a parte perché come precondizioni per la proposta di movimento tutte le due braccia devono avere quell'oggetto in mano e tutte le braccia devono essere occupate.
Quando si applica l'operatore per il tronco tutte le due braccia si liberano e l'oggetto torna ad esistere per i movimenti e per prenderlo in mano.

## 2.4 Combina oggetto

Quando l'agente ha due oggetti in mano può fare tre tipologie di combinazioni:

- rametto e molla;
- rametto e pietra;
- pietra e molla.

La proposta di combinazione ha come precondizione che l'agente abbia in mano due oggetti di quest'ultimi.

Quando si applica l'operatore di combinazione, un braccio si libera mentre l'altro braccio avrà in mano l'oggetto combinato. Quindi un braccio diventa libero. Inoltre, viene modificata l'ultima azione dello stato.

## 2.5 Scomponi oggetto

Quando l'agente ha un oggetto in mano che è una combinazione di due oggetti, può decidere di scomporlo (posando anche gli oggetti per terra).

La proposta di scomposizione ha come precondizioni che ci sia un braccio libero e che l'altro braccio abbia in mano un oggetto composto.

Quando si applica l'operatore di scomposizione, il braccio che teneva l'oggetto diventa libero e i due oggetti torneranno ad esistere, per i movimenti e per prenderli in mano. Viene modificata l'ultima azione dello stato.

## 2.6 Carica fionda

Quando l'agente ha in mano una fionda e nell'altra mano una pietra può decidere di caricare la fionda.

La proposta di caricamento della fionda ha come precondizioni che l'agente abbia in una mano la fionda e nell'altra mano una pietra. Inoltre, la fionda non deve essere già carica e l'uscita (la finestra) deve avere più di 0 colpi rimanenti per scardinarla.

Quando si applica l'operatore di caricamento della fionda, la fionda diventa carica e il braccio che aveva le pietre non ha più nessuno oggetto (ma non diventa libero).

## 2.6 Colpisci uscita

Quando l'agente ha la fionda carica può decidere di colpire l'uscita (la finestra) e avrà due possibilità:

- colpire l'estremità;
- colpire in un posto ovunque (non estremità);

La proposta per colpire ha come precondizioni che l'agente abbia una fionda carica in una mano e che l'uscita abbiamo più di 0 colpi rimanenti per scardinarla.

Quando si applica l'operatore per colpire l'uscita, possono capitare due casi:

- viene applicato l'operatore che colpisce l'estremità: servirà un colpo in meno per scardinare la finestra, la fionda non sarà più carica e le pietre tornano ad esistere come oggetto così si riattiva il movimento e la possibilità di prenderle in mano. Infine, solo più un braccio sarà occupato e viene modificata l'ultima azione dello stato;
- viene applicato l'operatore che NON colpisce l'estremità: ci saranno le stesse conseguenze di come se avesse colpito nell'estremità, tranne che i colpi che servono per scardinare la finestra rimangono invariati.

## 2.7 Posiziona tronco

Quando l'agente si trova vicino all'uscita e ha nelle due braccia lo stesso oggetto (ovvero un tronco) e quest'ultimo non è posizionato, può posizionarlo vicino all'uscita.

La proposta di posizionamento del tronco ha come precondizioni che l'agente sia vicino all'uscita, abbia un tronco nelle due mani e che quest'ultimo non sia posizionato.

Quando si applica l'operatore ci saranno due casi:

- se non c'era già un altro tronco posizionato: tutti le due braccia si liberano, il tronco viene posizionato, le braccia occupate ora saranno zero, e si aumenta l'altezza arrivabile per avvicinarsi all'uscita. Inoltre, viene modificata l'ultima azione dello stato;
- se c'era già un altro tronco posizionato: stesse conseguenze del caso precedente, ma semplicemente viene posizionato sopra l'altro.

## 2.8 Arrampicati ed esci

Quando entrambi i tronchi sono posizionati, l'agente si trova vicino all'uscita, ha entrambe le braccia libere e la finestra è stata scardinata, può decidere di arrampicarsi ed uscire dalla stanza, raggiungendo il goal.

La proposta per arrampicarsi ed uscire ha come precondizioni:

- l'agente ha entrambe le due braccia libere e si trova vicino all'uscita;
- entrambi i tronchi sono posizionati;
- la finestra ha zero colpi, ovvero è stata scardinata.

Quando l'operatore viene applicato si modifica lo stato segnando che l'agente è scappato e l'ultima azione dello stato viene modificata.

Successivamente viene controllato che se lo stato corrente è uguale a quello desiderato, allora viene aggiunto un attributo di successo e viene terminata l'esecuzione.

# 3. Rewards

L'aggiornamento avviene tramite la politica del Q-Learning, in quanto facendo delle prove ho ottenuto risultati migliori rispetto a Sarsa.
Vengono inserite delle regole RL per ogni operatore che potrà essere eseguito con un valore pari a zero. In questo modo l'agente non ha conoscenza e la deve acquisire durante le varie iterazioni delle esecuzioni.
Le regole RL (tutte inizializzate a zero) verranno aggiunte per i seguenti operatori:

- movimento;
- presa di un oggetto;
- presa del tronco;
- posa di un oggetto;
- posa del tronco;
- combinazione della fionda;
- combinazione del rametto e della pietra;
- combinazione della molla e della pietra;
- scomposizione della fionda;
- scomposizione del rametto e della pietra;
- scomposizione della molla e della pietra;
- carica della fionda;
- colpire la finestra nell'estremità;
- colpire la finestra NON nell'estremità;
- posizionamento del tronco;
- arrampicarsi ed uscire.

Queste regole permetteranno al meccanismo di RL di capire che ci sono delle regole di aggiornamento per i valori di reward.

I reward vengono assegnati in questo modo:

- se l'ultima azione eseguita è il movimento, viene assegnato un reward di -0.2 in quanto voglio incoraggiare la presa di un oggetto e le combinazioni rispetto al muoversi in giro di continuo;
- se l'ultima azione eseguita è la posa del tronco, viene assegnato un reward di -0.5 in quanto voglio preferire i movimenti per andare verso l'uscita e posizionare il tronco, rispetto a posarlo;
- se l'ultima azione eseguita è posa di un oggetto, viene

assegnato un reward di -0,2 in quanto voglio incoraggiare a combinare gli oggetti piuttosto che posarli;
- se l'ultima azione eseguita è la combinazione della fionda, viene assegnato un reward di 1 perché è la combinazione giusta di cui abbiamo bisogno;
- se l'ultima azione eseguita è la combinazione di rametto e pietre o la combinazione di molla e pietre, viene assegnato un reward di -1 perché sono combinazioni inutili, che non ci servono;
- se l'ultima azione eseguita è la scomposizione della fionda, viene assegnato un reward di -0.5 in quanto vogliamo preferire che vada a caricare la fionda piuttosto che scomporla che è un'azione sbagliata;
- se l'ultima azione eseguita è il posizionamento del tronco, viene assegnato un reward di 1 in quanto è un'azione giusta per il raggiungimento del goal;
- se l'ultima azione eseguita è caricare la fionda, viene assegnato un reward di 1 in quanto è un'azione corretta;
- se l'ultima azione eseguita è colpire NON all'estremità: viene assegnato un reward di -0.5 in quanto non è propriamente un'azione sbagliata perché colpisce la finestra, ma voglio preferire che colpisca nell'estremità;
- se l'ultima azione eseguita è colpire all'estremità: viene assegnato un reward di 1 in quanto è l'azione perfetta per scardinare la finestra;
- se l'ultima azione eseguita è quella di arrampicarsi ed uscire: viene assegnato un reward di 1 in quanto è l'azione giusta per scappare.

Ovviamente più volte avviene una certa azione e più volte, di conseguenza, viene elaborato il reward per quella azione andando dopo ogni iterazione a modificare sempre di più tutti i rewards.
Se una certa azione ha un determinato reward, questo reward viene sommato a quello restituito dall'elaborazione.

Utilizzando il meccanismo dell'ultima azione, i reward non sono collegati a quelli dell'azione precedente ma vengono inseriti in modo indipendente. Questo succede perché le produzioni degli operatori di elaborazione, quando viene modificata l'ultima azione e non soddisfanno più le precondizioni, vengono eliminati e la ricompensa viene tolta.

# 4. Risultati

Con questo sistema di reward si può notare che l'agente incomincia a muoversi e preferirà prendere un oggetto in mano. Dopo diverse iterazioni dove prova a prendere un oggetto e posarlo sarà in grado di preferire la combinazione degli oggetti piuttosto che posarli e capirà quali combinazioni sono giuste da fare rispetto alle altre. Una volta che capisce che è più giusto caricare la fionda e colpire, imparerà a colpire nell'estremità per scardinare la finestra. Inoltre, impara che avendo un tronco in mano sarà corretto posizionarlo vicino all'uscita. Una volta che tutto ciò è compiuto riuscirà ad uscire imparando che è giusto arrampicarsi per uscire.

L'efficienza di questo sistema si può notare effettuando una singola run, ma ovviamente saranno necessari molte più run, anche sulle migliaia, per fare in modo che l'agente diventi sempre più intelligente e capire quali azioni sono veramente corrette da effettuare evitando di bloccarsi nelle stesse azioni.

I risultati di cui ho precedentemente parlato sono stati ottenuti analizzando 10 run.

Sequenza delle 10 run con i numeri di iterazione necessari per raggiungere il goal: 529 → 120 → 243 → 235 → 178 → 211 → 150 → 217 → 105 → 237.

# 5. Osservazioni finali

Dopo aver sviluppato l'agente posso affermare che:

- l'agente, partendo da tabula rasa, è in grado di imparare quali azioni sono migliori rispetto alle altre dato il sistema di rewards anche su una singola run, riuscendo a raggiungere il goal;
- con l'esecuzione di un numero elevato di run, anche sulle migliaia, sarebbe in grado di diventare sempre più intelligente;
- si potrebbe migliorare introducendo anche un reward sui movimenti giusti e sbagliati.

# 6. Come eseguire

Una volta ottenuto il codice dal repository di GitHub https://github.com/lorenzobotto/progettoSoar, si può caricare il file su Soar ed eseguire tramite il pulsante Run dell'interfaccia.

Se si vogliono eseguire più Run è necessario inizializzare l'agente al termine di una Run premendo sul pulsante Init-soar e successivamente si può rieseguire.