HANAO TOWER

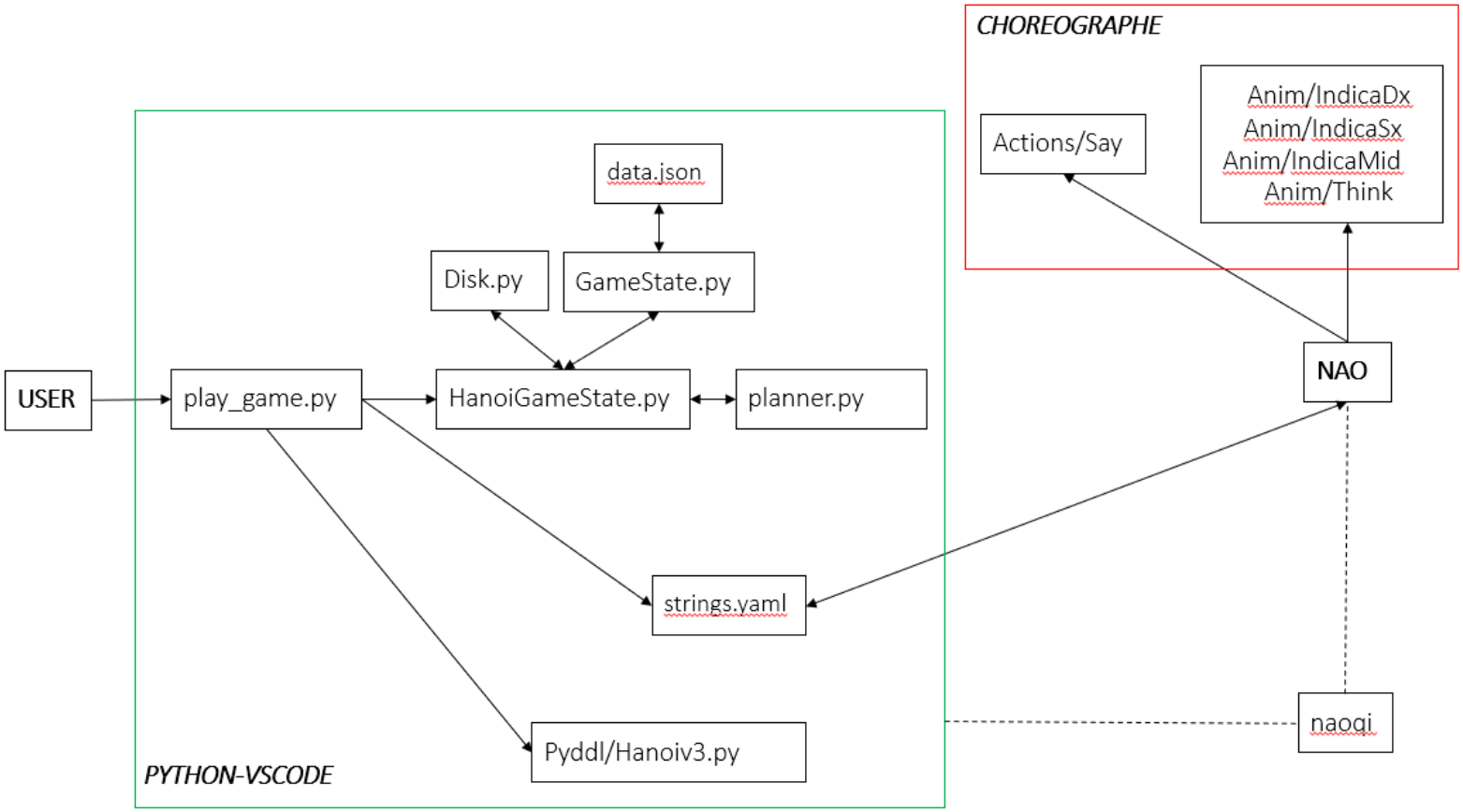
Attività progettuale di Intelligenza Artificiale M

Antonio Pio Volgarino antoniopio.volgarino@studio.unibo.it
Gabriele Ragusa gabriele.ragusa@studio.unibo.it

Abstract

L’attività progettuale svolta ha previsto lo sviluppo di un codice python in grado di risolvere, tramite gli algoritmi di Breadth First e STRIPS, il gioco della torre di Hanoi, con possibilità lato utente di indicare il numero di dischi con cui giocare (è stato scelto di limitare il numero di dischi ad un massimo di cinque).
L’algoritmo procede ad eseguire due planner differenti sullo stesso problema, analizzando i tempi di risoluzione e il numero di azioni necessarie per arrivare al goal per ognuno di essi, mentre il robot NAO interagisce con l’utente.
Quindi NAO indica all’user, attraverso una sequenza di animazioni e parole, le azioni da eseguire per arrivare insieme alla soluzione.
Infine viene illustrata dallo stesso robot una comparazione dei risultati in termini di numero di mosse e complessità temporale.

Fasi di implementazione



Fase 1: Connessione al robot virtuale e inizializzazione dei moduli *(play_game.py)*

E' richiesta l'iniziale esecuzione dell'ambiente Choregraphe, in cui avviare il robot virtuale e connetterlo localmente.
Il collegamento tra il modulo Python, in cui avviene la connessione al robot (il cliente si interfaccia al robot tramite indirizzo IP), e l'ambiente Choregraphe/robot virtuale è stato implementato attraverso la creazione di un python broker a runtime tramite la libreria naoqi.

```
myBroker = ALBroker("myBroker", "0.0.0.0", 0, robot_ip, 9559)
```

All'utente è garantita la possibilità di eseguire in modalità DEBUG o meno, aumentando la verbosità dell'algoritmo e diminuendo la quantità di frasi dette dal robot.

Fase 2: Definizione del gioco e dei componenti
(play_game.py - HanoiGameState.py - GameState.py - Disk.py)

Successivamente all'inizializzazione dei moduli necessari per interfacciarsi con l'utente (es. `ALProxy("ALAnimatedSpeech")`) il modulo play_game si interfaccia con la classe nel modulo "HanoiGameState.py" per definire lo stato iniziale del problema. Quest'ultima classe contiene funzioni ad-hoc per il problema della torre di Hanoi e per l'aggiornamento dello stato del problema, definito come classe in "GameState.py".

In GameState, infatti, sono collocati i metodi `def move(self, disk_name, pole_name)`, `def is_goal(self)` e `def is_start(self)` richiamati nel modulo HanoiGameState.

Per ridurre al minimo la possibilità di errori nel passaggio da un modulo all'altro e semplificare la fase di testing e debug, è stato creato un file di appoggio denominato "data.json", contenente il GameState visitato in tempo reale.

Fase 3: Implementazione del Breadth First Search Algorithm
(play_game.py - planner.py - HanoiGameState.py)

La soluzione implementata in un primo momento per il raggiungimento del goal prevede l'utilizzo di un algoritmo di Breadth First, effettuando una ricerca in ampiezza nel grafo degli stati. Utilizzando il metodo `get_gamestate(self)` della classe HanoiGameState è possibile recuperare lo stato del gioco, e quindi la posizione dei dischi sulle varie pile.
Il metodo `find_optimal_path(game_state)` permette di ottenere il percorso ottimale visitando tutti i possibili stati e transitando in ognuno di essi orizzontalmente fino a raggiungere il goal.
Giunti nello stato finale, il metodo termina restituendo il path ottimale contenente la lista ordinata di azioni da eseguire.

Fase 4: Implementazione dei movimenti e scelta delle frasi
(play_game.py - string.yaml)

Successivamente alla prima risoluzione del problema, si è fatto interagire NAO con l'utente attraverso delle frasi che spiegano il funzionamento del gioco, alternate a indicazioni del robot su come risolvere il gioco (si è supposto in questo caso che il robot avesse di fronte la torre di Hanoi con i dischi inizialmente disposti alla propria sinistra).
Tali indicazioni vengono fornite tramite una serie di azioni svolte da NAO.
Le animazioni, richiamate utilizzando il modulo `ALProxy("ALAnimatedSpeech")` di naoqi, sono state create da terzi per riprodurre i movimenti:

- "indica a destra"
- "indica a sinistra"
- "indica il centro"
- "pensa"

e vengono sincronizzate con le frasi presenti nel file "strings.yaml".
La corretta esecuzione delle animazioni è garantita importando le animazioni in Choregraphe ed installando queste ultime sul NAO Robot.

Fase 5: Implementazione di STRIPS Algorithm
(Pyddl/hanoiv3.py - planner.py - play_game.py)

In questa fase è stato utilizzato PyDDL come punto di incontro tra la sintassi STRIPS e la sintassi di python; di conseguenza si è avuta la necessità di definire il modulo hanoiv3.py in cui il dominio iniziale, la lista di mosse possibili e il problema fossero definiti in maniera esplicita.
Il planner Strips viene invocato attraverso il modulo planner.py contenuto nella directory Pyddl.
Gli argomenti passati come parametri interni alla funzione `planner` sono `problem`, definito in hanoiv3, e, per facilitare il debug, `verbose`, necessario a rendere più verboso il contenuto del planner.

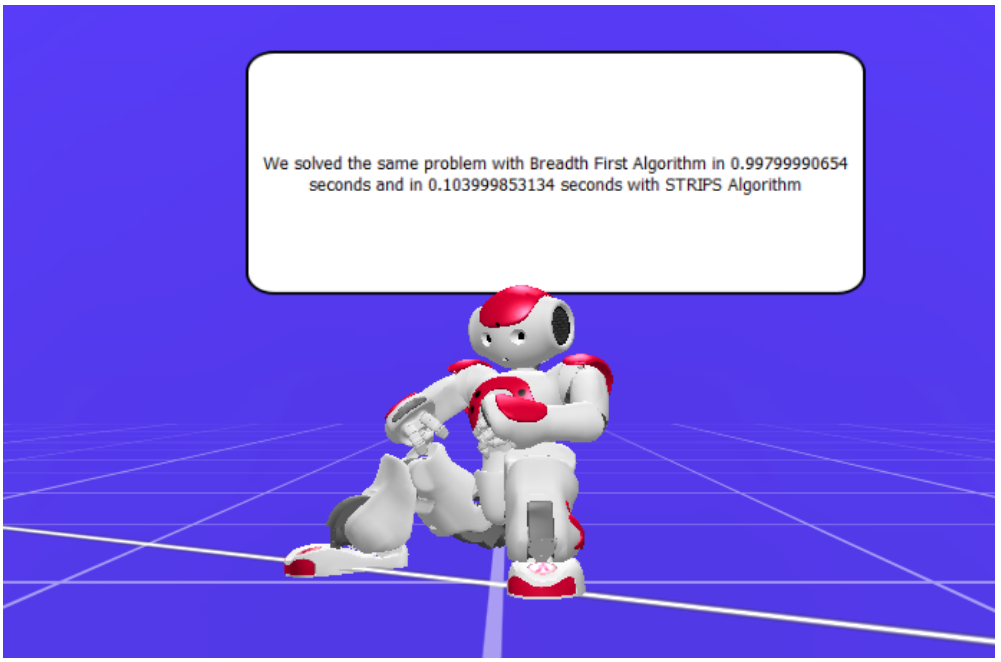
```
plan = planner(problem, verbose=verbose)
```

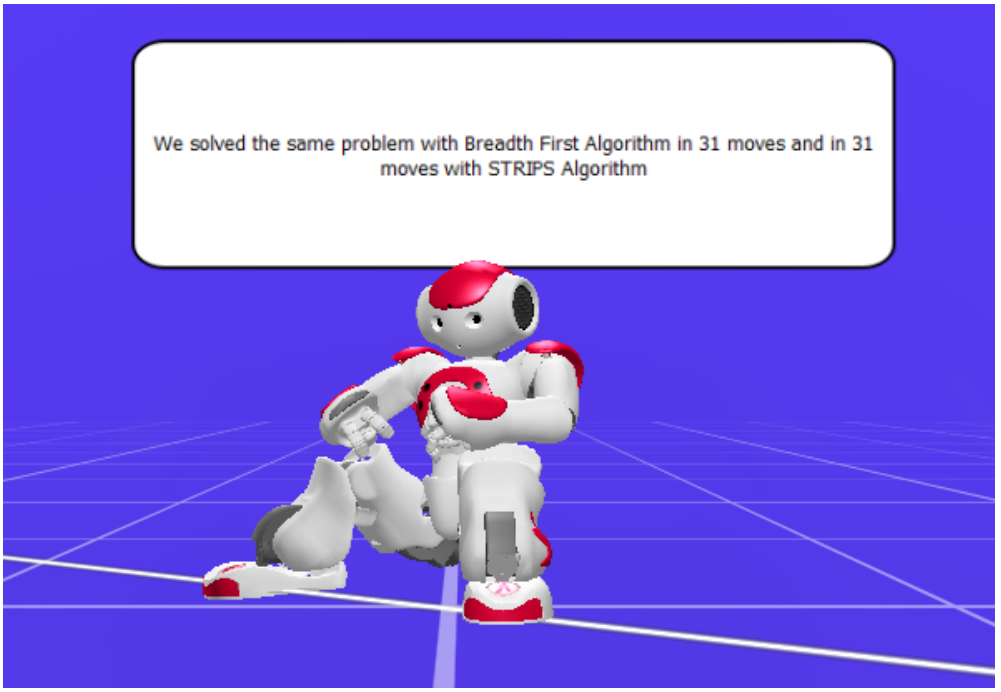
Il metodo runquiet, utilizzato nel modulo play_game.py, richiama havoiv3 e permette di ottenere una lista ordinata di mosse svolte dal planner Strips per risolvere il gioco col numero di dischi passato come parametro.

```
mosse=hanoiv3.runquiet(str(dischi),verbose=DEBUG)
```

Fase 6: Risultati

L'ultima fase prevede l'esplicazione del numero di mosse necessarie a risolvere il gioco con entrambi gli algoritmi implementati e la corrispondente complessità temporale.





Strumenti

Per lo sviluppo del progetto è stato utilizzato un editor per codice “Visual Studio Code” e il linguaggio di programmazione “Python v2.7.18”.
La libreria naoqi è stata importata utilizzando il corrispondente SDK per NAO Robot aggiornato ad agosto 2021, disponibile gratuitamente sul sito ufficiale.
Per testare il codice e mettere in esecuzione le diverse animazioni su NAO robot, è stato utilizzato l’ambiente offerto dall’applicativo Choregraphe, il quale consente di simulare la connessione in locale ad un robot virtuale.