

# Università della Calabria

Dipartimento di Matematica e Informatica

---



## Corso di Laurea Triennale in Informatica

Tesi di Laurea

### Implementazione di Robot Autonomi con Moduli di Ragionamento Automatico

**Relatore:**

Ch.mo prof. Giovambattista Ianni

**Secondo Relatore:**

Dott.ssa Denise Angilica

**Candidato:**

Emanuele Galardo

Matricola 230536

---

Anno Accademico 2023/2024

*A zio Pino.*

# Abstract

Il progetto di tesi si propone di sviluppare un robot capace di interagire autonomamente con giochi a turni per dispositivi mobili, attraverso l'uso di un braccio robotico e un sistema di visione artificiale.

Il robot utilizza un pennino per manipolare direttamente lo schermo touch di un dispositivo, emulando le azioni umane di toccare e trascinare. Questo permette al robot di partecipare attivamente al gioco in modo fisico piuttosto che virtuale, imitando l'interazione tipica di un giocatore umano.

Il cuore di questo sistema è un modulo di intelligenza artificiale che analizza le informazioni raccolte dal sistema di visione per identificare le diverse schermate di gioco e interpretare correttamente i dati visivi, come il testo e gli oggetti di gioco. Questa capacità di elaborazione visuale è cruciale, in quanto il successo del robot dipende dalla precisione con cui può interpretare l'ambiente di gioco per fare scelte strategiche.

Un focus particolare del progetto è stato posto su "2048", un gioco puzzle che richiede abilità logiche per combinare tessere numerate in modo da raggiungere la somma di 2048. Questo gioco è stato selezionato per la sua struttura semplice ma sfidante, che richiede due competenze chiave che il robot deve padroneggiare: il riconoscimento di forme geometriche specifiche, come i rettangoli delle tessere, e la capacità di leggere e interpretare i numeri sulle tessere.

# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
<b>2</b>	<b>Tecnologie Utilizzate</b>	<b>4</b>
2.1	Arduino . . . . .	5
2.1.1	Servomotore . . . . .	5
2.1.2	Shield . . . . .	6
2.2	Robot Delta . . . . .	7
2.3	OpenCV - Computer Vision . . . . .	8
2.3.1	Canny Edge Detection . . . . .	8
2.3.2	Contour Detection . . . . .	8
2.4	Optical Character Recognition . . . . .	10
2.5	Videogame "2048" . . . . .	10
2.6	Answer Set Programming . . . . .	11
<b>3</b>	<b>Acting</b>	<b>12</b>
3.1	Introduzione al sistema . . . . .	13
3.1.1	Componenti hardware . . . . .	14
3.1.2	Componenti software . . . . .	15
3.2	Struttura del modulo Acting . . . . .	16
3.3	Firmware Arduino . . . . .	18
3.4	Software di controllo . . . . .	22
3.4.1	Driver BB2 . . . . .	23
3.5	Applicativo Web . . . . .	26
3.5.1	Calibration . . . . .	26
3.5.2	Control . . . . .	27

<b>4</b>	<b>Vision</b>	<b>29</b>
4.1	Struttura del modulo Vision . . . . .	30
4.1.1	Struttura dei sottomoduli dei giochi . . . . .	31
4.1.2	Struttura del sottomodulo 2048 . . . . .	32
4.2	Estensione di Object Finder . . . . .	33
4.2.1	Riconoscimento di testo . . . . .	33
4.2.2	Riconoscimento di rettangoli . . . . .	34
4.3	Elaborazione delle immagini . . . . .	36
4.3.1	Riconoscimento della griglia di gioco . . . . .	37
4.3.2	Riconoscimento dei valori delle tessere . . . . .	38
4.4	Astrazione della griglia di gioco . . . . .	41
4.5	Connessione Vision e Think . . . . .	42
4.5.1	Rappresentazione della conoscenza . . . . .	42
4.5.2	Invocazione del solver . . . . .	43
<b>5</b>	<b>Think</b>	<b>45</b>
5.1	Scelte implementative . . . . .	46
5.2	Modellazione del problema . . . . .	47
5.2.1	Unione delle tessere . . . . .	48
5.2.2	Posizionamento delle tessere . . . . .	50
5.3	Valutazione delle mosse . . . . .	53
5.3.1	Studio delle euristiche . . . . .	53
5.3.2	Implementazione delle euristiche . . . . .	54
<b>6</b>	<b>Conclusioni</b>	<b>56</b>
<b>A</b>	<b>Algoritmo di Canny</b>	<b>59</b>
	<b>Riferimenti bibliografici</b>	<b>62</b>

# Capitolo 1

## Introduzione

### Scopo del lavoro

Il mondo videoludico è un settore in continua espansione ed evoluzione che negli ultimi anni ha visto un incremento esponenziale di nuove tecnologie e nuovi modi di giocare. Nei videogiochi allo stato dell'arte si possono trovare istanze giocattolo di problemi reali, quali possono essere la guida autonoma di veicoli, la gestione di un'azienda o la risoluzione di un problema matematico. Essi ci garantiscono un ambiente controllato e sicuro in cui poter testare nuove idee e nuove tecnologie, senza dover affrontare i rischi e le incertezze del mondo reale. I videogiochi sono quindi diventati un campo di ricerca molto interessante per la comunità scientifica, assieme alla robotica. I robot sono macchine in grado di eseguire compiti autonomamente, senza l'intervento umano; essi sono utilizzati in una vasta gamma di applicazioni che vanno dalla produzione industriale alla medicina. Sono pochi gli ambiti in cui un robot non può trovare applicazione e la ricerca scientifica è attiva nel cercare di sviluppare nuove tecnologie per migliorare le prestazioni e le capacità dei robot.

Il progetto di tesi si pone l'obiettivo di unire questi due mondi, rendere funzionante un robot in grado di giocare autonomamente ai videogiochi.

In particolare, il progetto si concentra sul gioco 2048, un gioco di logica in cui il giocatore deve combinare delle tessere numerate per ottenere il numero 2048. Il gioco è stato scelto perchè per risolverlo è necessario padroneggiare due importanti abilità: il saper riconoscere rettangoli e il saper riconoscere testo. Queste due abilità generalizzate e unite insieme possono essere utilizzate in una vasta gamma di

applicazioni, prima fra tutte il riconoscimento di bottoni con testo in un'interfaccia grafica.

## Obiettivi del lavoro

Nel corso del progetto sono stati definiti i seguenti obiettivi:

- Realizzare un braccio robotico in grado di svolgere alcune operazioni di base su uno schermo touch screen. In particolare il robot deve poter toccare e/o trascinare un pennino touch su schermi di varie dimensioni.
- Integrare il braccio robotico in un sistema preesistente capace di controllare un robot suo simile.
- Sviluppare un sistema di primitive per riconoscere forme rettangolari e testo in un'immagine.
- Implementare un sistema di visione e astrazione del gioco 2048.
- Realizzare un'intelligenza artificiale in grado di risolvere il gioco 2048.
- Unire i tre sistemi in un unico sistema in grado di giocare autonomamente.

## Struttura della tesi

Prendendo in considerazione la mole di lavoro e la diversità di argomenti trattati nel progetto, la tesi è stata suddivisa in quattro capitoli principali, ciascuno dei quali tratta un aspetto specifico del progetto.

Tali capitoli sono i seguenti:

1. **Tecnologie utilizzate:** Nel primo capitolo vengono introdotte le tecnologie utilizzate per la realizzazione del progetto, dai componenti hardware del robot agli strumenti software utilizzati per controllarlo.
2. **Acting:** Nel secondo capitolo viene presentato il modulo Acting, responsabile del controllo del robot. Sono presentate inizialmente il firmware del robot e i driver per collegare il robot al computer. Successivamente sono esposte i sistemi di controllo e calibrazione del robot.

3. **Vision:** Nel terzo capitolo viene presentato il modulo Vision, che si occupa di gestire la visione e l'astrazione del gioco 2048. Vengono analizzate le funzioni necessarie per internalizzare le informazioni del gioco nel sistema.
4. **Think:** Nel quarto e ultimo capitolo viene presentato il modulo Think, sistema di ragionamento automatico che permette al robot di prendere decisioni in base alle informazioni ricevute dal modulo vision. Vengono studiate la modellazione del gioco e le strategie di gioco implementate.



# Capitolo 2

## Tecnologie Utilizzate

### Indice

---

<b>2.1</b>	<b>Arduino . . . . .</b>	<b>5</b>
2.1.1	Servomotore . . . . .	5
2.1.2	Shield . . . . .	6
<b>2.2</b>	<b>Robot Delta . . . . .</b>	<b>7</b>
<b>2.3</b>	<b>OpenCV - Computer Vision . . . . .</b>	<b>8</b>
2.3.1	Canny Edge Detection . . . . .	8
2.3.2	Contour Detection . . . . .	8
<b>2.4</b>	<b>Optical Character Recognition . . . . .</b>	<b>10</b>
<b>2.5</b>	<b>Videogame "2048" . . . . .</b>	<b>10</b>
<b>2.6</b>	<b>Answer Set Programming . . . . .</b>	<b>11</b>

---

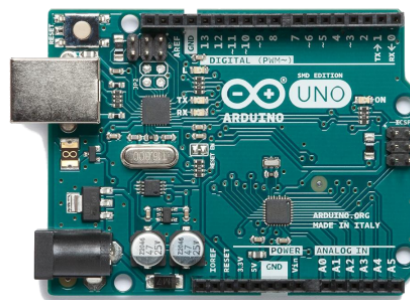
## 2.1 Arduino

*Arduino* è una piattaforma hardware open-source basata su un microcontrollore. Nasce nel 2005 da un progetto di alcuni membri del Interaction Design Institute Ivrea (IDII) di Ivrea, in Italia. [3]

E' composta da una scheda elettronica che può essere programmata per svolgere diverse funzioni. La scheda è dotata di ingressi e uscite digitali e analogiche, che possono essere utilizzate per interfacciarsi con sensori, attuatori e altri dispositivi elettronici.

Arduino è progettato per essere facile da usare anche per chi non ha esperienza di elettronica e la sua programmazione avviene tramite un linguaggio di programmazione derivato dal linguaggio C++.

Allo stato attuale, Arduino è diventato uno standard de facto per la prototipazione di dispositivi elettronici, grazie alla sua semplicità di utilizzo e alla vasta comunità di sviluppatori che lo supporta.



**Figura 2.1:** Scheda Arduino Uno

### 2.1.1 Servomotore

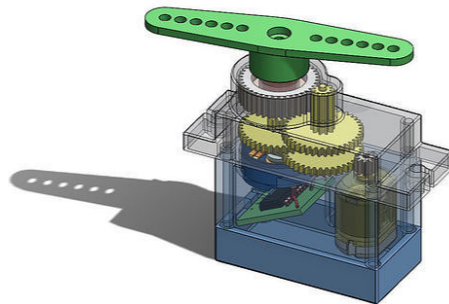
Un *servomotore* è un tipo di attuatore rotativo che permette di controllare la posizione di un albero di uscita. Essi sono molto utilizzati in ambito robotico per controllare le articolazioni dei robot. Tali dispositivi sono dotati di un motore elettrico, di un sistema di ingranaggi e di un circuito di controllo che permette di controllare la posizione dell'albero di uscita.

I servomotori sono dotati di tre cavi: uno per la tensione di alimentazione, uno per la messa a terra e uno per il segnale di controllo. Il segnale di controllo è un segnale

PWM (Pulse Width Modulation) che permette di controllare il comportamento del servomotore.

I servomotori possono essere di due tipi: *servomotori a rotazione continua* e *servomotori a posizione*. I primi permettono di controllare la velocità di rotazione dell'albero di uscita, mentre i secondi permettono di controllare la posizione dell'albero di uscita.

Nel nostro progetto abbiamo utilizzato dei servomotori Dynamixel XL430-250T[4] della Robotis, in grado di operare sia a rotazione continua che a posizione.



**Figura 2.2:** Modello 3D di un servomotore

### 2.1.2 Shield

Gli *shield* sono schede di espansione che si collegano sopra la scheda principale di Arduino attraverso i connettori presenti sulla scheda stessa.

Esistono molti tipi di shield, ognuno con una funzione specifica. Ad esempio, esistono shield per aggiungere connettività Wi-Fi, GSM, GPS, per controllare motori, per interfacciarsi con display LCD, e molti altri.

Alcuni shield sono prodotti direttamente da Arduino, altri sono prodotti da terze parti.

Nel nostro progetto abbiamo utilizzato un DynamixelShield[5] della Robotis per controllare i servo Dynamixel, come mostrato in Figura 2.3. Seppur non sia strettamente necessario, l'utilizzo di uno shield semplifica notevolmente l'interfacciamento con i motori Dynamixel.

Lo shield è dotato di alcuni connettori per collegare i motori Dynamixel, e di un convertitore di livello logico per interfacciarsi con Arduino. Questo shield permette di controllare i motori Dynamixel tramite la libreria *Dynamixel2Arduino*[6] che traduce i comandi inviati da Arduino in comandi comprensibili dai motori Dynamixel.



**Figura 2.3:** Dynamixel Shield per Arduino

## 2.2 Robot Delta

I *robot delta* sono una particolare tipologia di robot paralleli molto diffusi in ambito industriale. Il loro nome deriva dalla forma a triangolo del braccio meccanico, che ricorda la lettera greca  $\Delta$ , come mostrato in Figura 2.4.

Questi robot sono caratterizzati da una struttura meccanica composta da tre bracci, che si muovono in modo indipendente l'uno dall'altro.

I robot delta sono molto veloci e precisi, e sono in grado di eseguire movimenti molto rapidi con una grande libertà di movimento. Hanno particolarmente applicazione in ambiti di picking and placing, cioè per il prelievo e il posizionamento di oggetti. [7]

Queste caratteristiche rendono questi robot particolarmente adatti per lo scopo del nostro lavoro.



**Figura 2.4:** Robot Delta

## 2.3 OpenCV - Computer Vision

*OpenCV* (Open Source Computer Vision Library) è una libreria open-source per la visione artificiale. Essa fornisce più di 2500 funzioni per l'elaborazione di immagini.[8] OpenCV opera su immagini rappresentate come array Numpy[9], e fornisce funzioni per manipolare, trasformare e analizzare le immagini.

Sebbene OpenCV sia scritta in C++, esistono binding per Python, che è il linguaggio di programmazione utilizzato in questo progetto. Ne parleremo più approfonditamente in Capitolo 4.

### 2.3.1 Canny Edge Detection

La *Canny Edge Detection* è un algoritmo per la rilevazione dei contorni in un'immagine proposto da John F. Canny nel 1986[10]. OpenCV fornisce una implementazione dell'algoritmo di Canny Edge Detection tramite la funzione `cv2.Canny()`.

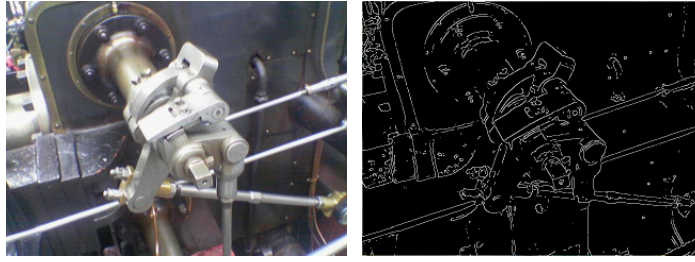
L'algoritmo è composto da cinque passaggi[11]:

1. **Riduzione del Rumore:** l'immagine viene filtrata con un filtro gaussiano per ridurre il rumore.
2. **Calcolo del gradiente:** vengono calcolate le derivate parziali dell'immagine per ottenere la magnitudine e la direzione del gradiente.
3. **Soglia di magnitudine del gradiente :** vengono selezionati i pixel che hanno una magnitudine del gradiente superiore ad una certa soglia.
4. **Soppressione dei non massimi:** vengono eliminati i pixel che non sono massimi locali nella direzione del gradiente.
5. **Isteresi:** vengono selezionati i pixel che sono connessi ai pixel di bordo.

### 2.3.2 Contour Detection

La *Contour Detection* è un algoritmo per la rilevazione dei contorni in un'immagine[12].

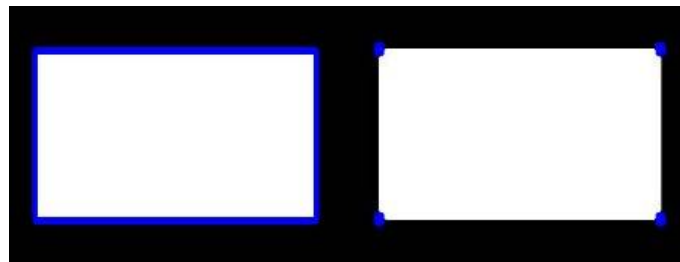
Nella implementazione fornita da OpenCV, l'algoritmo è in grado di rilevare i contorni di oggetti bianchi su sfondo nero. L'implementazione dell'algoritmo di



**Figura 2.5:** Esempio di Canny Edge Detection

Contour Detection può essere invocata tramite la funzione `cv2.findContours()`. Il risultato del metodo è rappresentato dai punti del contorno come una lista di tuple, ciascuna contenente le coordinate del punto (x, y).

Modificando i parametri del metodo, i punti ottenuti possono essere approssimati prendendo solo gli estremi del contorno che porta ad una rappresentazione più compatta dello stesso.



**Figura 2.6:** Esempio di approssimazione di punti

I punti blu rappresentano il contorno, nel primo caso il contorno è rappresentato da tutti i punti, mentre nel secondo caso il contorno è approssimato prendendo solo gli estremi. Più l'oggetto è semplice, più l'approssimazione sarà efficace.



**Figura 2.7:** Esempio di approssimazione di un poligono

Lo standard di OpenCV della rappresentazione dei contorni è usato in molte altre funzioni con gli scopi più disparati. Un esempio come in Figura 2.7 è l'approssimazione di un poligono con un altro poligono con meno vertici.

## 2.4 Optical Character Recognition

L'OCR (Optical Character Recognition) è una tecnologia che permette di riconoscere il testo in un'immagine.[13] L'obiettivo dell'OCR è quello di convertire il testo in un'immagine in testo digitale, in modo che possa essere elaborato da un computer.

L'OCR è utilizzato in molte applicazioni, come la scansione di documenti, la traduzione di testi, la lettura di targhe e molte altre.

*Tesseract* è un motore open-source per il riconoscimento ottico dei caratteri (OCR). Esso è stato sviluppato originariamente da Hewlett-Packard e successivamente rilasciato come open-source. [14] Tesseract è in grado di riconoscere testo in diverse lingue e di operare su immagini in diversi formati.

Tesseract è scritto in C++ ma esistono binding per Python, che è il linguaggio di programmazione utilizzato in questo progetto. Ne parleremo più approfonditamente in Capitolo 4.



Figura 2.8: Esempio di OCR su un'immagine

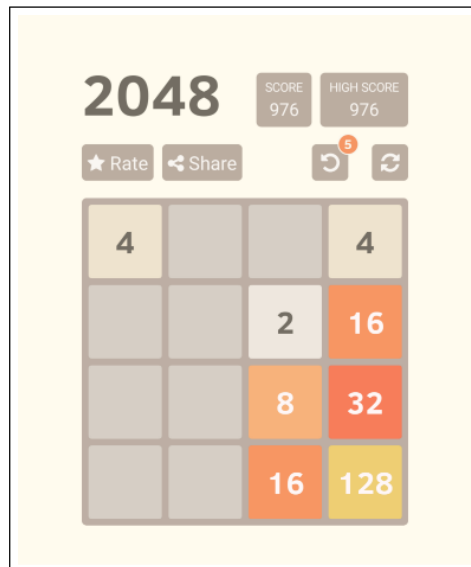
## 2.5 Videogame "2048"

2048 è un videogioco libero per dispositivi mobili e desktop creato da Gabriele Cirulli nel 2014.[15] Ne esistono diverse implementazioni e varianti del gioco originale, ma il principio di base rimane lo stesso, cioè combinare le tessere con lo stesso numero per ottenere una tessera con un numero maggiore di potenza di 2.

Il gioco è composto da una griglia  $n \times n$  in cui vengono generate casualmente delle tessere con numeri potenze di 2, una per mossa. Il giocatore può spostare le tessere in quattro direzioni: su, giù, destra e sinistra. Se due tessere con lo stesso

numero si scontrano, si fondono in una tessera con il numero doppio. Lo scopo del gioco è ottenere una tessera con il numero 2048 nella versione originale 4 x 4, o una tessera diversa nelle varianti.

Il gioco termina quando la griglia è piena e non ci sono più mosse possibili.



**Figura 2.9:** Schermata del gioco 2048

## 2.6 Answer Set Programming

*Answer Set Programming* (ASP) è un paradigma di programmazione logica dichiarativa basato sulla logica dei predicati.

Tale paradigma è utilizzato per la risoluzione di problemi di soddisfacibilità, cioè finalizzato a trovare una soluzione ad un problema che soddisfi un insieme di vincoli. Ai problemi di soddisfacibilità si aggiungono i problemi di ottimizzazione, in cui si cerca la soluzione che massimizza o minimizza una certa funzione obiettivo, tramite l'introduzione nel linguaggio dei vincoli deboli.

Negli anni si sono sviluppate varie estensioni di ASP che ne hanno ampliato le capacità, come ad esempio l'introduzione di funzioni aggregative o di predicati esterni.

ASP è utilizzato in svariati campi, come l'intelligenza artificiale, la pianificazione, la verifica formale e molti altri.[1]

Allo stato dell'arte esistono due principali solver per ASP: *clingo*[16] sviluppato dall'Università di Potsdam e *dlv2*[17] sviluppato dall'Università di Calabria.



# Capitolo 3

## Acting

### Indice

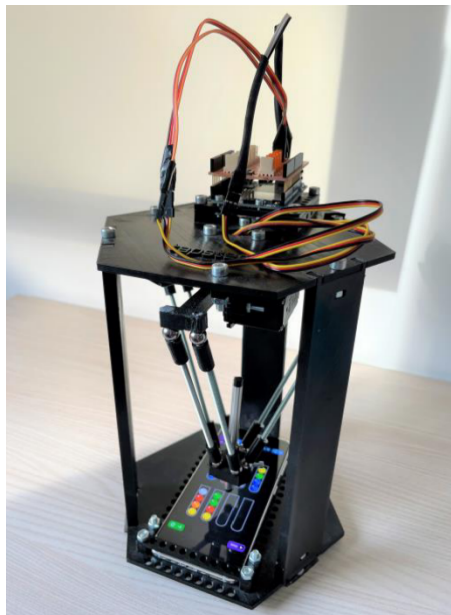
---

<b>3.1</b>	<b>Introduzione al sistema . . . . .</b>	<b>13</b>
3.1.1	Componenti hardware . . . . .	14
3.1.2	Componenti software . . . . .	15
<b>3.2</b>	<b>Struttura del modulo Acting . . . . .</b>	<b>16</b>
<b>3.3</b>	<b>Firmware Arduino . . . . .</b>	<b>18</b>
<b>3.4</b>	<b>Software di controllo . . . . .</b>	<b>22</b>
3.4.1	Driver BB2 . . . . .	23
<b>3.5</b>	<b>Applicativo Web . . . . .</b>	<b>26</b>
3.5.1	Calibration . . . . .	26
3.5.2	Control . . . . .	27

---

### 3.1 Introduzione al sistema

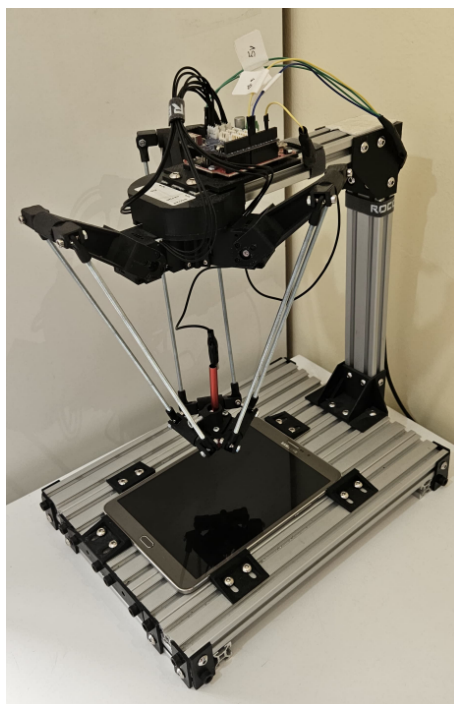
Il lavoro iniziale di questa tesi è stato quello di creare BrainyBot v2 una versione migliorata del robot BrainyBot visibile in Figura 3.1. Esso è un robot delta, tipologia di robot presentata in Sezione 2.2, costruito sul modello del robot Tapster di Jason Huggins[18] da parte di un gruppo di ricerca dell'Università della Calabria con a capo *prof. Giovambattista Ianni*, relatore di questa tesi. Il suddetto automa è in grado di interagire con uno smartphone, simulando l'interazione umana con lo stesso.



**Figura 3.1:** BrainyBot v1

La necessità di creare un versione migliorata del robot BB1, diminutivo di BrainyBot, è dovuta a diverse limitazioni del robot originale, tra cui:

- **Fragilità:** La quasi totalità della struttura è costituita da materiali stampati in 3D, che lo rendono facilmente danneggiabile.
- **Dimensione dispositivo:** BB1 è stato costruito per interagire con smartphone di piccole dimensioni e non è in grado di interagire con dispositivi di dimensioni maggiori come tablet.
- **Limitata Precisione:** Gli attuatori sono collegati tramite magneti, che ne limitano la precisione.



**Figura 3.2:** BrainyBot v2

Il nuovo robot BrainyBot v2 visibile in Figura 3.2, per brevità BB2, è stato progettato con l'obiettivo di superare le limitazioni del suo predecessore.

Il robot è stato costruito con materiali più resistenti e con attuatori più precisi affinché sia in grado di interagire con dispositivi di dimensioni maggiori.

### 3.1.1 Componenti hardware

Il robot è composto da diverse componenti hardware, tra cui:

- **Arduino Uno:** scheda di controllo del robot. Essa contiene i driver necessari per controllare i servo motori e per comunicare con il computer. I suddetti driver verranno trattati in Sezione 3.3.
- **Shield Dynamixel:** scheda di espansione collegata alla scheda Arduino per interfacciare i servo Dynamixel, il collegamento è visibile in Figura 3.3. L'utilizzo di questo ha portato a delle criticità, che verranno discusse in Sezione 3.3.
- **3 Servo Dynamixel:** attuatori del robot collegati alla scheda di controllo tramite lo shield Dynamixel. Essi sono responsabili del movimento del robot.
- **Computer:** il computer è il dispositivo che controlla il robot. Esso invia i comandi al robot e riceve i dati dai sensori.



**Figura 3.3:** Collegamento tra Shield Dynamixel e Arduino

### 3.1.2 Componenti software

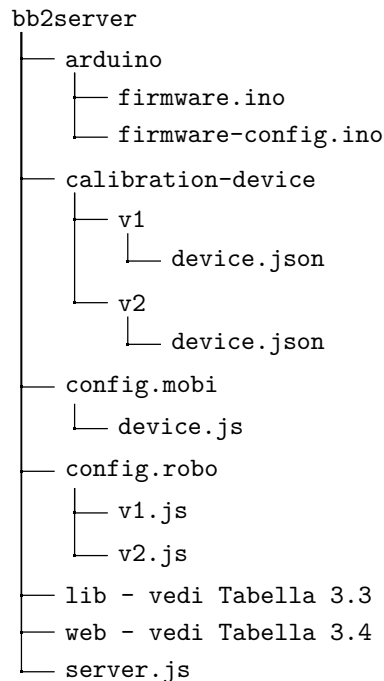
Il software del robot è composto da diverse parti, tra cui:

- **Firmware Arduino:** il firmware Arduino è il codice che viene eseguito sulla scheda di controllo del robot. Esso si occupa di ricevere i comandi dal computer e di inviarli ai motori. Svolge inoltre il compito di leggere i dati dai sensori e di inviarli al computer.
- **Software di controllo:** il software di controllo è il programma che viene eseguito sul computer. Esso si occupa di inviare i comandi al robot e di ricevere i dati dai sensori. Esso è inoltre il responsabile di interpretare i comandi inviati dall'utente e/o da altri sistemi tramite delle API e di tradurli in movimenti del robot.
- **Applicativo Web:** l'applicativo web è un'interfaccia grafica che permette all'utente di interagire con il robot. Avviando calibrazioni, eseguendo movimenti e visualizzando i dati dei sensori.

Questi componenti saranno trattati approfonditamente nelle sezioni seguenti.

## 3.2 Struttura del modulo Acting

L'intero progetto è stato organizzato in una repository Git[19]. La repository è organizzata in cartelle che contengono il codice sorgente del firmware Arduino, del software di controllo, e dell'applicativo web. La struttura della repository è visibile in Tabella 3.1.



**Tabella 3.1:** Struttura della repository del modulo Acting

La cartella `arduino` contiene il codice del firmware Arduino che verrà trattato in Sezione 3.3.

La cartella `calibration-device` contiene i file di calibrazione dei robot(BB1 e BB2) sui vari dispositivi. Essi contengono le coordinate cartesiane dei punti in cui il pennino del robot deve spostarsi, e gli angoli dei servo corrispondenti.

Sono presenti due cartelle di configurazione, una per i dispositivi e una per i robot. La cartella `config.mobi` contiene i file di configurazione dei dispositivi come nell'esempio Codice 3.1. La cartella `config.robo` contiene i file di configurazione dei robot come nell'esempio in Codice 3.2.

La cartella `lib` contiene le librerie utilizzate dal software di controllo. Esse verranno trattate in Sezione 3.4.

La cartella `web` contiene il codice sorgente dell'applicativo web che verrà trattato in Sezione 3.5.

Il file `server.js` è il file principale del sistema che avvia il server Node.js.

```
1 var config_mobi = {};  
2  
3 config_mobi.deviceName = "P10 Lite";  
4  
5 // P10 Lite values:  
6 config_mobi.calWidth = 26;  
7 config_mobi.calHeight = 38;  
8  
9 // calibration points for each width x and height y  
10 config_mobi.pointOnWidth = 16;  
11 config_mobi.pointOnHeight = 8;  
12  
13 module.exports = config_mobi;
```

**Codice 3.1:** Esempio di file di configurazione di un dispositivo

```
1 var config_robo = {}  
2  
3 // Robot version  
4 config_robo.version = "v2";  
5  
6 config_robo.baudrate = 57600;  
7  
8 config_robo.defaultHeight = -190;  
9 config_robo.s1 = { pin: 1, min: -45, max: 75, rangeMin: 45, rangeMax: -75};  
10 config_robo.s2 = { pin: 2, min: -45, max: 75, rangeMin: 45, rangeMax: -75};  
11 config_robo.s3 = { pin: 3, min: -45, max: 75, rangeMin: 45, rangeMax: -75};  
12  
13 // Set boundries to prevent breaking robot  
14 config_robo.boundary_enabled = true;  
15 config_robo.boundary_x = { min: -70, max: 70 };  
16 config_robo.boundary_y = { min: -100, max: 100 };  
17 config_robo.boundary_z = { min: -240, max: -180 };
```

**Codice 3.2:** Esempio di file di configurazione di un robot

### 3.3 Firmware Arduino

Il firmware Arduino è il codice che viene eseguito sulla scheda di controllo del robot. Nel progetto originale del robot BB1 era stato utilizzata la libreria *Firmata*[20] che permette di controllare la scheda Arduino e l'intero robot da un computer tramite una connessione seriale. Utilizzando lo shield Dynamixel è stato necessario scrivere un firmware personalizzato per il robot, in quanto la comunicazione con i servo avviene tramite la porta seriale e non è compatibile con il protocollo Firmata.

E' stato quindi scritto un firmware che comunica al computer non sulla porta seriale, ma tramite un seriale virtuale. Esso si trova nel file `firmware.ino`.

La prima implementazione del firmware utilizzava la libreria *SoftwareSerial*[21], che permette di creare una porta seriale virtuale scegliendo 2 pin digitali della scheda Arduino: uno per la trasmissione (TX) e uno per la ricezione(RX).

Tuttavia questa libreria si è rivelata inadatta da un punto di vista prestazionale, in quanto rallentava notevolmente la comunicazione tra il robot e il computer. Essa non permette una comunicazione full-duplex e non è in grado di gestire la velocità di trasmissione dei dati richiesta dal robot.

E' stato quindi necessario utilizzare una libreria alternativa, la *AltSoftSerial*[22], che permette di creare una porta seriale virtuale full-duplex che riduce la latenza, ma impone l'uso di determinati pin. Quest'ultimo vincolo è stato soddisfatto, in quanto i pin utilizzati dalla libreria sono compatibili con lo shield Dynamixel e non erano già stati utilizzati da altre componenti del robot.

La versione finale del firmware è visibile in Codice 3.3.

Indubbiamente la scelta di creare un firmware personalizzato su una porta seriale virtuale ha portato ad un alto degrado delle performance del robot.

Valutando la semplicità dell'utilizzo della libreria *Dynamixel2Arduino*[6] e la non necessità di una comunicazione fulminea tra il robot e il computer (data la differenza di vari ordini di grandezza tra i tempi di esecuzione dei movimenti( $\approx 2s$ ) e i tempi di comunicazione ed elaborazione( $\approx 60ms$ )), si è deciso di mantenere l'attuale implementazione.

E' possibile inviare comandi al robot tramite la USB collegata alla porta seriale virtuale. La comunicazione può avvenire controllando direttamente il flusso con un

software di controllo delle porte COM come *Putty*, oppure utilizzando l'applicativo web che verrà descritto in Sezione 3.5.

Il firmware è stato progettato per ricevere comandi con un numero massimo di 32 caratteri terminati da  $\backslash n$ . Tutti i comandi ritornano una risposta che può essere di 3 tipi:

- Valore: il comando ha restituito un valore.
- OK: il comando è stato eseguito correttamente.
- ERR: si è verificato un errore durante l'esecuzione del comando.

L'elenco completo dei comandi è visibile in Tabella 3.2.

Comando	Parametri	Descrizione
GAP	///	Restituisce i valori degli angoli di tutti e tre i servo motori
SAP	$\alpha, \beta, \gamma$	Imposta i valori degli angoli di tutti e tre i servo motori
GSP	id	Restituisce il valore dell'angolo del servo con id specificato
SSP	id, angolo	Imposta il valore dell'angolo del servo con id specificato
GPV	id	Restituisce la velocità massima del servo con id specificato
SPV	id, velocità	Imposta la velocità massima del servo con id specificato
ISM	id	Restituisce se il servo con id specificato si sta muovendo

**Tabella 3.2:** Lista dei comandi del robot

Oltre al firmware base è stato necessario scriverne uno di configurazione che permette di configurare le impostazioni permanenti degli attuatori.

Questo firmware, chiamato `firmware-config.ino`, è necessario che venga eseguito ogni volta che si cambia un attuatore, in quanto i valori di default di quest'ultimo differiscono da quelli richiesti dal robot.

Il firmware di configurazione è visibile in Codice 3.4.



```

1  #include <AltSoftSerial.h>
2  #include <DynamixelShield.h>
3  #define BUFFER_SIZE 32
4
5  using namespace ControlTableItem;
6
7  AltSoftSerial soft_serial;
8  DynamixelShield dxl;
9  uint8_t i = 0, id = 0, bufferIndex = 0;
10 char inputBuffer[BUFFER_SIZE];
11
12 void setup() {
13     soft_serial.begin(57600);
14     dxl.begin(57600);
15     for (i = 1; i < 4; i++) {
16         dxl.torqueOff(i);
17         dxl.writeControlTableItem(PROFILE_ACCELERATION, i, 100);
18         dxl.writeControlTableItem(PROFILE_VELOCITY, i, 70);
19         dxl.torqueOn(i);
20     }
21 }
22
23 void compute() {
24     if(strncmp(inputBuffer, "SAP", 3) == 0){
25         ...
26         soft_serial.println("OK");
27     } else if ... {
28         ...
29     } else soft_serial.println("ERR");
30 }
31
32 void loop() {
33     while (soft_serial.available()) {
34         char c = soft_serial.read();
35         if (c == '\n' || bufferIndex == BUFFER_SIZE - 1) {
36             inputBuffer[bufferIndex++] = '\0';
37             compute();
38             bufferIndex = 0;
39         } else{
40             inputBuffer[bufferIndex++] = toupper(c);
41         }
42     }
43 }

```

**Codice 3.3:** Firmware Arduino

```

1  #include <AltSoftSerial.h>
2  #include <DynamixelShield.h>
3
4  using namespace ControlTableItem;
5
6  AltSoftSerial soft_serial;
7  DynamixelShield dxl;
8
9  void setup() {
10     soft_serial.begin(57600);
11     soft_serial.println("START CONFIGURATION");
12     dxl.begin(57600);
13     dxl.setPortProtocolVersion(2.0);
14
15     for (int id = 1; id < 4; id++) {
16         soft_serial.print("ID : ");
17         soft_serial.print(id);
18         if (dxl.ping(id)) {
19             soft_serial.print(", Model Number: ");
20             soft_serial.println(dxl.getModelNumber(id));
21         } else {
22             soft_serial.println(", NOT FOUND!");
23         }
24     }
25
26     for (int i = 1; i < 4; i++) {
27         dxl.torqueOff(i);
28         dxl.setOperatingMode(i, OP_EXTENDED_POSITION);
29         dxl.torqueOn(i);
30     }
31
32     soft_serial.println("READY");
33 }
34
35 void loop() {}

```

**Codice 3.4:** Firmware Arduino di configurazione

## 3.4 Software di controllo

Il software di controllo è il programma che viene eseguito sul computer connesso fisicamente al robot. Esso svolge vari compiti tra cui:

- Comunicazione in ingresso e in uscita con il robot.
- Calibrazione del robot.
- Elaborazione delle richieste inviate dall'utente (tramite Applicativo Web) e/o da altri sistemi tramite delle API.

Il software di controllo e l'applicativo web sono un'estensione del progetto Tappy di Guntis[23], fork del progetto originale di Jason Huggins[18] che si limita al solo comparto hardware. Tappy è stato esteso per permettere il controllo e la calibrazione del robot BB2.

Particolarmente ostico è stato il riuscire a mantenere la compatibilità con il robot BB1, in quanto i due robot hanno differenze hardware e software significative.

Lo sviluppo del nuovo software per BB2 (chiamato BB2Server) è partito già con l'intenzione di renderlo il più possibile modulare, quindi da permettere l'aggiunta di nuovi robot e dispositivi in futuro; è stato infatti effettuato un refactoring di alcune parti del codice per esternalizzare alcuni parametri specifici del robot BB1 e dei dispositivi mobili in file di configurazione separati.

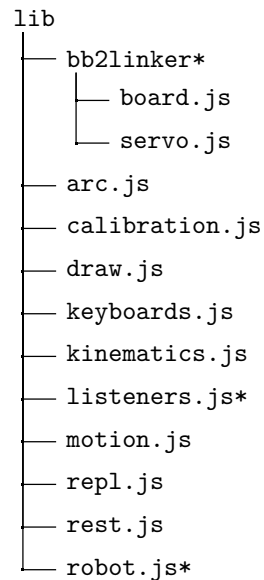
I file di configurazione adottano la stessa struttura e sono composti da un oggetto JavaScript che contiene i parametri specifici del robot o del dispositivo. Questa scelta ha permesso di mantenere il codice più pulito e generale.

Non tutti i file della cartella `lib` sono stati modificati, in quanto alcuni di essi sono stati mantenuti invariati in quanto non necessitano di modifiche oppure perché non utilizzati. I file modificati sono stati contrassegnati con un asterisco(\*) nella rappresentazione della cartella `lib` visibile in Tabella 3.3, su di essi verrà fatta una breve descrizione.

La libreria `bb2linker` è la responsabile della mediazione tra il computer e BB2. Viene trattata in Sezione 3.4.1.

Il file `robot.js` è il file principale del software di controllo. Esso si occupa di inizializzare il robot, di ricevere i comandi dall'utente e di inviarli al robot. Utilizza la libreria `bb2linker` o `johnny-five` a seconda del robot.

Il file `listeners.js` si occupa di gestire gli eventi generati dal robot, come ad esempio il completamento di un movimento. E' stata necessaria una modifica per differenziare la dinamica di movimento tra BB1 e BB2 in quanto meccanicamente diversi.



**Tabella 3.3:** Struttura della cartella lib

Il software di controllo è progettato per essere utilizzato in combinazione con l'applicativo web oppure tramite delle richieste REST API da parte di altri sistemi.

### 3.4.1 Driver BB2

Per permettere l'integrazione con il sistema di controllo esistente è stato necessario scrivere un driver per BB2 con la stessa interfaccia del driver per BB1.

BB1 utilizza il firmware Firmata e tale firmware si interfaccia con il software di controllo tramite la libreria `johnny-five`[24].

Per i motivi precedentemente esposti nel Sezione 3.3 è stato necessario scrivere un firmware personalizzato per BB2 e dunque un driver personalizzato.

Il firmware di BB2, chiamato `bb2linker`, è diviso in due parti: `board.js` e `servo.js`. Il primo si occupa della comunicazione con il robot, il secondo implementa la controparte software dei servo motori.

Per gestire la comunicazione in ingresso e in uscita sono state utilizzate due liste:

- `communicationOutput`: lista che contiene i comandi da inviare a BB2. L'invio dei comandi avviene tramite un thread che periodicamente invia i comandi

presenti nella lista in ordine di arrivo. Se i comandi si aspettano una risposta alla richiesta di inoltro viene associato un ticket univoco che permette di associare la risposta al comando.

- **communicationInput**: lista che contiene le risposte ai comandi inviati a BB2. L'inserimento delle risposte nella lista avviene tramite un thread che viene invocato ogni qual volta siano presenti dati sulla seriale. La coda è bloccante quindi non è possibile prelevare una risposta fin quando tale risposta non si trova in cima alla coda.

L'inizializzazione dei thread è visibile in Codice 3.5.

```
1 function startRead(){
2   this.parser.on('data', (data) => {
3     if(this.debug){
4       console.log("<<< " + data);
5     }
6     this.communicationInput.push(data);
7   });
8 }
9
10 function startWrite(){
11   setInterval(() => {
12     if(this.communicationOutput.length > 0){
13       let data = this.communicationOutput.shift();
14       if (this.debug) {
15         console.log(">>> " + data);
16       }
17       this.sp.write(data + "\n");
18     }
19   }, 30);
20 }
```

**Codice 3.5:** Inizializzazione dei thread di lettura e scrittura

Utilizzando una struttura a code si è resa trasparente la comunicazione tra robot e computer, permettendo in future implementazioni di aggiungere nuovi tipi di comunicazione con il robot, diversi dalla comunicazione su porta seriale.

Si è inoltre semplificata l'implementazione di nuovi comandi da inviare al robot, in quanto è sufficiente aggiungere stringhe alla lista `communicationOutput` e leggere le risposte dalla lista `communicationInput` ricordando il nostro ticket associato.

L'implementazione della funzione per leggere una risposta dal robot è visibile in Codice 3.6.

```
1  async function readLine(id, ticket){
2      let line;
3      while(this.actualTicket < ticket){
4          await new Promise((resolve) => {setTimeout(resolve, 10)});
5      }
6      line = this.communicationInput.shift();
7      while (line == undefined){
8          await new Promise((resolve) => {setTimeout(resolve, 10)});
9          line = this.communicationInput.shift();
10     }
11     this.actualTicket++;
12     return line;
13 }
```

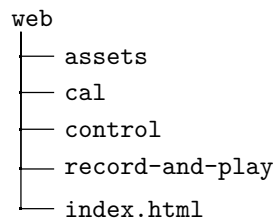
**Codice 3.6:** Funzione per leggere una risposta da BB2

La funzione `readLine()` è implementata come funziona asincrona. Essa attende inizialmente il proprio turno attraverso il meccanismo di ticket e successivamente esegue polling sulla coda di risposte fino a quando non trova una risposta.

## 3.5 Applicativo Web

L'applicativo web è un programma accessibile tramite un browser web, che permette all'utente di interagire con il robot tramite interfaccia grafica. Esso è composto tra 3 pagine:

- **Calibration:** Pagina a cui accede il dispositivo mobile su cui dovrà essere eseguita la calibrazione. vedi Sezione 3.5.1
- **Control:** Pagina a cui accedono gli utenti per controllare il robot. vedi Sezione 3.5.2
- **Record and Play:** Pagina presente nel progetto di Tappy, permette di registrare dei movimenti del robot e di riprodurli in seguito. Questa funzionalità non è stata implementata nella versione attuale del sistema in quanto non ritenuta necessaria. La pagina è stata mantenuta per futuri sviluppi.



**Tabella 3.4:** Struttura della cartella web

### 3.5.1 Calibration

La pagina di calibrazione permette di calibrare il robot, ovvero di associare le coordinate cartesiane del punto in cui si vuole che il pennino del robot si sposti alle posizioni dei 3 servo. La calibrazione è divisa in 4 fasi:

1. **Scelta dei punti di tocco:** Per ovvie ragioni temporali e pratiche non è possibile calibrare il robot su tutti i punti del dispositivo mobile ed è dunque necessario scegliere un numero limitato di punti, che siano rappresentativi di tutto il dispositivo. Il numero di punti da calibrare è scelto dall'utente tramite il file di configurazione del dispositivo insieme alle dimensioni del dispositivo. Il sistema calcolerà automaticamente i punti in modo da coprire l'intera area

del dispositivo e assicurandosi che tra di loro siano linearmente indipendenti. Quest' ultima condizione è necessaria in quanto tutti i punti non inclusi nella calibrazione vengono interpolati linearmente dai punti calibrati.

2. **Tentativi di tocco:** Scelti i punti, il robot si sposterà sopra ognuno di essi ed inizierà ad abbassare il pennino ogni tot millisecondi fino a toccare il dispositivo mobile. Il tempo di attesa tra ogni abbassamento serve per permettere al dispositivo mobile di rilevare e inviare le coordinate del tocco. Una volta ricevuto il tocco il robot ritornerà all'altezza di default e si sposterà al punto successivo; questo processo viene ripetuto per ogni punto scelto e si conclude quando tutti i punti scelti sono stati toccati oppure quando il pennino si abbassa eccessivamente non riuscendo a toccare il dispositivo.
3. **Salvataggio del file di calibrazione:** Una volta completata la calibrazione il sistema salva un file di calibrazione in modo da poterlo utilizzare in futuro e se dovesse già esistere il sistema lo sovrascrive, altrimenti ne crea uno nuovo.
4. **Test di precisione della calibrazione:** Viene avviato un test per verificare la precisione della calibrazione. Il robot inizierà a effettuare dei tocchi circolari intorno al centro del dispositivo e stamperà in console l'errore tra la posizione reale del tocco e la posizione calcolata dalla calibrazione. Questo test è utile per verificare la precisione della calibrazione e per capire se è necessario ricalibrare il robot.

Nella pagina sono presenti le informazioni del dispositivo mobile e 3 pulsanti: *fullscreen* per visualizzare la pagina a schermo intero, *mcalibrate* per avviare la calibrazione, e *STOP* per interrompere la calibrazione.

Rispetto alla versione originale la pagina ha ricevuto un restyling grafico ed è stato risolto un bug che rendeva selezionabili le informazioni del dispositivo mobili su cui viene svolta la calibrazione.

### 3.5.2 Control

La pagina di controllo permette agli utenti di controllare il robot tramite l'interfaccia grafica.



Sono presenti 2 modi per controllare il robot: tramite la scelta degli angoli dei servo, oppure tramite la scelta delle coordinate cartesiane del punto in cui si vuole che il pennino del robot si sposti.

Quest'ultima opzione è operativa solo se il robot è stato calibrato.

Rispetto alla versione originale la pagina ha ricevuto un restyling grafico e sono stati risolti dei bug che rendevano errati i valori visualizzati degli angoli dei servo e delle coordinate cartesiane quando avvenivano dei cambi repentini di posizione.

Questi bug erano dovuti alla mancata sincronizzazione tra il software e l'arduino, infatti quest'ultimo non era in grado di rispondere a tutte le richieste in tempo reale e quindi il software non riceveva i dati corretti. Il problema è stato risolto introducendo un meccanismo di controllo degli accessi alla seriale; in questo modo il software attende la risposta dell'Arduino prima di inviare un nuovo comando. E' stato inoltre introdotto una coda di richiesta per servire le richieste in ordine di arrivo.

Inoltre l'applicativo web quando viene utilizzato accoppiato a BB2 permette anche di controllare la velocità di movimento dei servo. Nell'utilizzo con BB1 questa funzionalità non è presente in quanto i servo non sono in grado di muoversi a velocità diverse.



**Figura 3.4:** Pagina di controllo

# Capitolo 4

## Vision

### Indice

---

<b>4.1</b>	<b>Struttura del modulo Vision</b>	<b>30</b>
4.1.1	Struttura dei sottomoduli dei giochi	31
4.1.2	Struttura del sottomodulo 2048	32
<b>4.2</b>	<b>Estensione di Object Finder</b>	<b>33</b>
4.2.1	Riconoscimento di testo	33
4.2.2	Riconoscimento di rettangoli	34
<b>4.3</b>	<b>Elaborazione delle immagini</b>	<b>36</b>
4.3.1	Riconoscimento della griglia di gioco	37
4.3.2	Riconoscimento dei valori delle tessere	38
<b>4.4</b>	<b>Astrazione della griglia di gioco</b>	<b>41</b>
<b>4.5</b>	<b>Connessione Vision e Think</b>	<b>42</b>
4.5.1	Rappresentazione della conoscenza	42
4.5.2	Invocazione del solver	43

---

## 4.1 Struttura del modulo Vision

Come è stato fatto per il modulo Acting in Sezione 3.2, anche il modulo Vision è stato organizzato in una repository git.

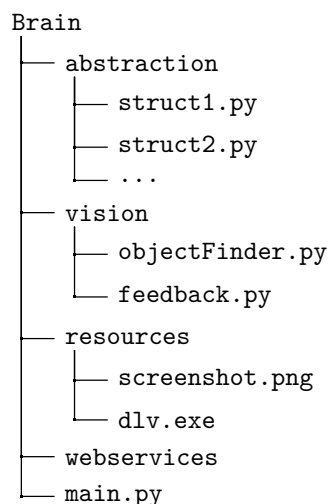
Il componente di nostro interesse è la cartella Brain, parte della più ampia repository BrainyBot, che contiene tutti i moduli del progetto.

Tale cartella contiene tutto il codice significativo per il modulo Vision, sviluppato interamente in Python con l'ausilio di librerie come OpenCV (visto in Sezione 2.3) e Tesseract (visto in Sezione 2.4).

La cartella Brain contiene 4 package principali:

- **abstraction**: contiene le strutture dati utilizzate per rappresentare le informazioni estratte dalle immagini;
- **vision**: contiene il file `objectFinder.py` che contiene le primitive per l'estrazione delle informazioni dalle immagini e il file `feedback.py` che contiene delle primitive per fornire feedback sullo stato della vision;
- **resources**: contiene le risorse utilizzate dal software, come gli screenshot delle schermate da analizzare e gli eseguibili di DLV, quest'ultimi verranno discussi nel prossimo capitolo.
- **webservices**: contiene il codice per ottenere gli screenshot

La struttura della repository del modulo Vision è mostrata in Tabella 4.1.



**Tabella 4.1:** Struttura del modulo Vision

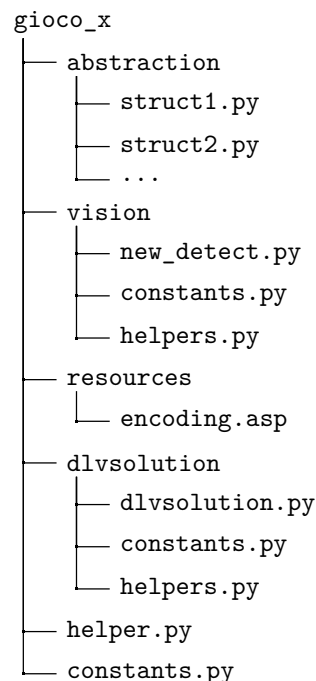
Ai package principali vanno aggiunti i package specifici per ogni gioco.

#### 4.1.1 Struttura dei sottomoduli dei giochi

Il package di ogni gioco contiene ulteriori sotto-package con lo stesso nome dei package principali. Questi sotto-package contengono il codice specifico per ogni gioco, come le strutture dati specifiche e le primitive per l'estrazione delle informazioni.

Inoltre, ogni package contiene un file `helper.py` contenente la funzione principale per l'esecuzione del sottomodulo e una package `dlvsolution` contenente il codice necessario per invocare l'Intelligenza Artificiale responsabile della risoluzione del gioco.

Per rendere più chiaro il codice nella struttura del package di ogni gioco, si è scelto di aggiungere file `constants.py` e `helpers.py` che contengono rispettivamente le costanti e le funzioni di supporto utilizzate nei vari file del sottomodulo.



**Tabella 4.2:** Struttura di un package di un gioco generico

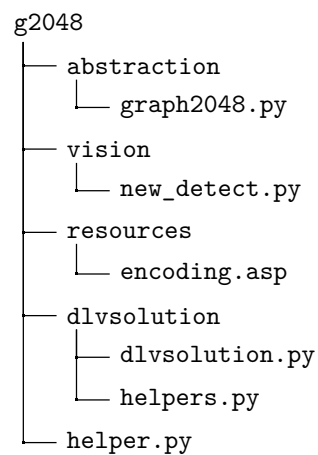
Si è scelto di organizzare la vision di ogni gioco con una struttura modulare per rendere più semplice l'aggiunta di nuovi giochi.

Inoltre tutti i componenti non ritenuti strettamente dipendenti dal gioco devono essere spostati nei package principali al fine di evitare ridondanze e di realizzare una libreria di vision generica.

### 4.1.2 Struttura del sottomodulo 2048

Nel lavoro di tesi è stato sviluppato il sottomodulo per il gioco 2048. Esso è stato sviluppato seguendo la struttura descritta in Sezione 4.1.1 e la sua struttura è mostrata in Tabella 4.3.

Come precedentemente detto, le funzioni primitive generiche ossia quelle responsabili del riconoscimento dei caratteri sono state spostate nel package principale `vision`, nello specifico nel file `objectFinder.py`. Esso insieme al sottomodulo 2048 verranno discussi nel proseguo del capitolo.



**Tabella 4.3:** Struttura del package del gioco 2048

## 4.2 Estensione di objectFinder

Il file principale del modulo Vision è `objectFinder.py`. Esso contiene le primitive per l'estrazione delle informazioni dalle immagini.

Nel lavoro svolto è stata estesa la classe `objectFinder` con nuove funzionalità al fine di rendere più generica la libreria e di permettere l'estrazione di informazioni da immagini di giochi differenti.

Oltre a funzioni già presenti nell'attuale libreria, sono state aggiunte funzioni per il riconoscimento di testo e di rettangoli, funzionalità che, combinate insieme, permettono svariate applicazioni.

### 4.2.1 Riconoscimento di testo

Per il riconoscimento di testo è stata utilizzata il motore OCR Tesseract, presentato in Sezione 2.4.

Per utilizzare Tesseract in Python è stata utilizzata la libreria `pytesseract` che permette di interfacciarsi con il programma Tesseract.

La prima funzione realizzata è `find_text()` che dato la posizione di una sezione di immagine restituisce il testo contenuto in essa.

```
1 def find_text(self, x, y, w, h) -> str:
2     img = self.__img_matrix[y:y+h, x:x+w]
3     # Upscale the image to improve the OCR if the image is too small
4     if img.shape[0] < 150 or img.shape[1] < 150:
5         img = cv2.resize(img, (round(img.shape[1]*1.5),
6                               round(img.shape[0]*1.5)))
7     img = self.__img_matrix[y:y+h, x:x+w].copy()
8     elem = ts.image_to_string(img, config='--psm 8 --oem 3')
9     return elem if elem != '' else None
```

**Codice 4.1:** Funzione per il riconoscimento di testo in un'immagine

La funzione `find_text()` prende in input la posizione di un rettangolo all'interno dell'immagine e restituisce il testo contenuto in esso. Per migliorare la qualità del riconoscimento si è scelto di ingrandire l'immagine se essa è troppo piccola.

L'invocazione del programma Tesseract avviene tramite la chiamata alla funzione `image_to_string()` che restituisce il testo contenuto nell'immagine.

Si è notato che il riconoscimento di testo è un'operazione molto onerosa in termini di tempo di calcolo, per tale motivo si è scelto di parallelizzare tale operazione. Per ricercare testo in più porzioni dell'immagine si è implementata la funzione `find_texts_multithread()`.

```
1 def find_texts_multithread(self, boxes) -> list:
2     num_processes = min(multiprocessing.cpu_count(), len(boxes))
3     boxes_per_process = len(boxes)
4     ...
5     with multiprocessing.Manager() as manager:
6         for i in range(num_processes):
7             res_list.append(manager.list())
8             if i == num_processes - 1:
9                 processes.append(multiprocessing.Process(target
10                     ↪ = self.__worker_find_text,
11                     ↪ args=(boxes[(num_processes - 1) *
12                     ↪ boxes_per_process:], res_list[i])))
13             else:
14                 processes.append(multiprocessing.Process(target
15                     ↪ = self.__worker_find_text, args=(boxes[i *
16                     ↪ boxes_per_process: (i + 1) *
17                     ↪ boxes_per_process], res_list[i])))
18             processes[i].start()
19         for i in range(num_processes):
20             processes[i].join()
21             res.extend(list(res_list[i]))
22     return res
```

**Codice 4.2:** Funzione per il riconoscimento di testo in un'immagine in multithread

## 4.2.2 Riconoscimento di rettangoli

Per rendere più strutturata la ricerca di testo è stata implementata la funzione `find_boxes()` che restituisce una lista di tutti i rettangoli presenti in un'immagine.

Alla funzione è necessario passare due parametri, `min_thresh` e `max_thresh`, che rappresentano rispettivamente i valori minimi e massimi per la soglia di Canny, algoritmo di edge detection utilizzato per trovare i contorni. Questi parametri sono necessari per adattare l'algoritmo alle diverse tipologie di immagini.

Tale funzione ha molteplici utilizzi, uno dei quali è il riconoscimento di bottoni nelle schermate dei giochi. Combinando la funzione `find_text()` con `find_boxes()` è possibile ricostruire la posizione di un bottone e il suo testo, permettendo di eseguire azioni in base al testo riconosciuto.

```

1 def find_boxes(self, min_thresh = 25, max_thresh = 80) -> list:
2     contour = cv2.Canny(self.__img_matrix, min_thresh, max_thresh)
3     contours, _ = cv2.findContours(contour, ...)
4     boxes = []
5     for i in range(len(contours)):
6         per = cv2.arcLength(contours[i], True)
7         epsilon = 0.05 * per
8         approx = cv2.approxPolyDP(contours[i], epsilon, True)
9         if len(approx) != 4 or cv2.isContourConvex(approx) == False or
10            cv2.contourArea(approx) < 3000:
11             continue
12         x, y, w, h = cv2.boundingRect(approx)
13         boxes.append((x, y, w, h))
14     return boxes

```

**Codice 4.3:** Funzione per la ricerca di rettangoli in un'immagine

Si noti come la funzione `find_boxes()` svolga internamente una serie di operazioni per filtrare i contorni trovati dall'algoritmo di edge detection. In particolare ci si accerta che il contorno sia un rettangolo chiuso e che abbia un'area sufficientemente grande da poter contenere del testo, in quanto è necessario per evitare falsi positivi dovuti a rumore o a contorni non desiderati. Si è ritenuta necessaria un'approssimazione dei contorni per riconoscere rettangoli con angoli non perfettamente retti, tipici delle moderne interfacce grafiche.

Per riconoscere strutture di rettangoli più complesse, come potrebbe essere un menù contestuale, si è implementata la funzione `find_boxes_and_hierarchy()` mostrata in Codice 4.4. Essa restituisce una lista di rettangoli e la gerarchia tra essi nello stesso formato restituito dalla funzione `findContours()` di OpenCV.

```

1 def find_boxes_and_hierarchy(self):
2     mat_contour = np.zeros((self.__img_matrix.shape[0],
3         self.__img_matrix.shape[1]), dtype=np.uint8)
4     boxes = self.find_boxes()
5     for box in boxes:
6         x, y, w, h = box
7         cv2.rectangle(mat_contour, (x, y), (x + w, y + h), 255, ...)
8     boxes, hierarchy = cv2.findContours(mat_contour, ...)
9     return boxes, hierarchy[0]

```

**Codice 4.4:** Funzione per la ricerca di rettangoli con gerarchia



## 4.3 Elaborazione delle immagini

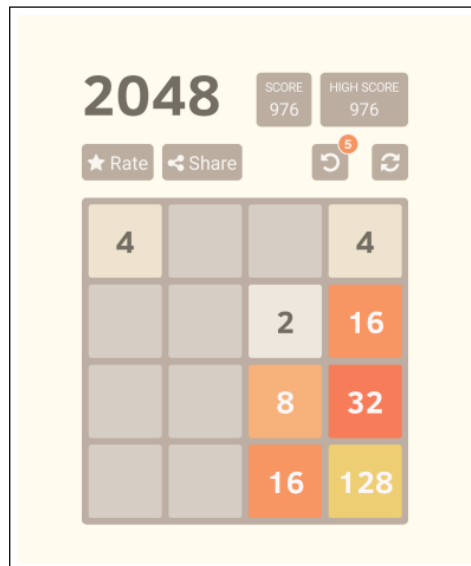
Le funzioni presenti nell'oggetto `objectFinder` vengono utilizzate all'interno dell'oggetto `Matching2048` che si trova in `new_detect.py`. Questo oggetto utilizza tutte le primitive del modulo `vision` e le combina per estrarre le informazioni necessarie per il gioco 2048.

Rispetto alla struttura classica del package `Vision`, il sottomodulo `vision` di 2048 ha un comportamento differente. Infatti, la struttura classica prevede che ad ogni nuova immagine venga generato un nuovo oggetto di `new_detect` e che vengano estratte tutte le informazioni necessarie per il gioco di interesse.

Nel caso del gioco 2048, invece, si è scelto di mantenere un oggetto di `new_detect`, chiamato appunto `Matching2048`, attivo per tutta la durata del gioco.

La scelta di conservare l'oggetto della `vision` è stata presa per un motivo di efficienza, in quanto il gioco 2048 mantiene la griglia di gioco invariata per tutta la durata del gioco.

Un esempio di schermata di gioco è mostrato in Figura 4.1 ed è stato utilizzato come riferimento nel resto del capitolo.



**Figura 4.1:** Schermata di gioco di 2048

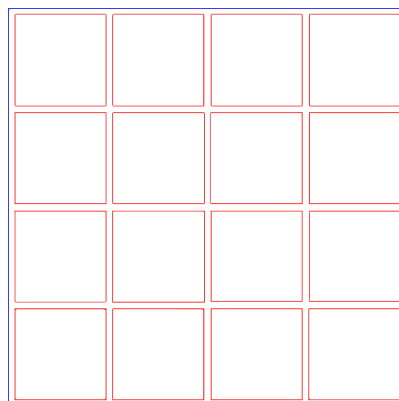
### 4.3.1 Riconoscimento della griglia di gioco

Il riconoscimento della griglia di gioco è un'operazione fondamentale. Com'è stato detto in precedenza, la griglia rimane invariata durante tutto il gioco e quindi è necessario riconoscerla una sola volta.

Al fine di identificare la griglia di gioco si è scelto di utilizzare un'euristica principale: la griglia è composta da  $n^2$  rettangoli, con  $n$  il lato della griglia.

Questa informazione ci permette di riconoscere la griglia in modo semplice, contando il numero di rettangoli contenuti in ogni rettangolo trovato con la funzione `find_boxes()`.

Se il numero di rettangoli contenuti in un rettangolo è uguale a  $n^2$  allora è stata trovata la griglia di gioco, altrimenti, nel caso in cui più rettangoli rispettino tale condizione, si sceglie quello con  $n$  maggiore che con alta probabilità corrisponde alla griglia di gioco.



**Figura 4.2:** Riconoscimento della griglia di gioco

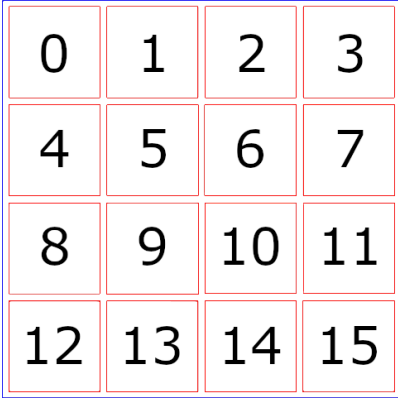
In Figura 4.2 è mostrato il riconoscimento della griglia di gioco. Il rettangolo blu rappresenta la griglia di gioco, mentre i rettangoli rossi rappresentano le tessere.

Svolta quest'operazione si prosegue con l'ordinamento delle tessere all'interno della griglia. Quest'azione è necessaria in quanto i rettangoli trovati con la funzione `find_boxes()` non sono ordinati e non si sa a priori quale rettangolo corrisponda ad una determinata tessera.

```
1 ordered_boxes = sorted(numbers_boxes, key=lambda x: x[0] * 10 + x[1] * 100)
```

Per risolvere questo problema si è scelto di ordinare i rettangoli in base alla loro posizione, in modo da avere una lista ordinata da sinistra a destra e dall'alto in basso.

Nonostante l'algoritmo utilizzato sia greedy, si è dimostrato efficace nella totalità dei casi testati.



0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

**Figura 4.3:** Riconoscimento delle tessere all'interno della griglia

In Figura 4.3 è mostrato il riconoscimento della griglia di gioco con le tessere numerate per posizione.

Si noti come le tessere siano state numerate come una matrice linearizzata dunque tutte le tessere sono indicizzate su un vettore.

#### 4.3.2 Riconoscimento dei valori delle tessere

Il riconoscimento dei valori delle tessere è l'unica operazione che viene eseguita ad ogni frame. Questo è dovuto al fatto che i valori delle tessere cambiano ad ogni mossa del giocatore.

La computazione ripetuta di questa operazione rende critico il tempo di calcolo, per tale motivo sono state implementate e testate diverse versioni del riconoscimento dei valori delle tessere.

La prima versione implementata è quella più semplice, che consiste nel riconoscere il testo all'interno di ogni singola tessera utilizzando una variante della funzione `find_text()`, visibile in Codice 4.1, specifica per il riconoscimento dei numeri. Questa versione si è dimostrata molto lenta e poco efficiente, in quanto il riconoscimento di ogni singola tessera richiede molto tempo ed aumenta in termini di tempo di calcolo all'aumentare delle tessere.

La seconda versione implementata sfrutta `find_texts_multithread()` per parallelizzare il riconoscimento delle tessere. Questa versione si è dimostrata molto più efficiente della precedente, in quanto il riconoscimento delle tessere viene diviso in

più thread, ognuno responsabile di una parte delle tessere. Tuttavia questa versione non è stata sufficiente per ridurre il tempo di calcolo a valori accettabili.

La terza e ultima versione implementata sfrutta alcune euristiche per ridurre il numero di tessere da riconoscere. In particolare 2048 è semi deterministico, in quanto una parte dei valori delle tessere è prevedibile. Per sfruttare questa caratteristica si è scelto di riconoscere solo le tessere che non sono prevedibili e che quindi potrebbero assumere nuovi valori.

Già con questa euristica si è ottenuto un notevole miglioramento delle prestazioni, in quanto il numero di tessere da riconoscere si riduce all'aumentare delle tessere con valori prevedibili.

La seconda euristica sfrutta la conoscenza del colore delle tessere senza valore. Infatti, le tessere senza valore sono sempre dello stesso colore e quindi è possibile riconoscerle senza utilizzare il riconoscimento di testo. Facendo ciò il riconoscimento di testo, l'operazione più onerosa, viene eseguita solo una volta per ogni frame.

Il Codice 4.5 mostra la funzione `find_numbers_with_cache()` che implementa le euristiche descritte sopra. In particolare utilizza una vettore chiamato `cache` che contiene i valori delle tessere già conosciute, le tessere che non si conoscono sono inizializzate a 0.

```
1 def find_numbers_with_cache(self, cache, only_first=True):
2     numbers = cache
3     for i in range(len(numbers)):
4         if numbers[i] == 0:
5             x, y, w, h = self.__numbers_boxes[i]
6             if np.array_equal(self.__finder.get_image()[y+h//2,
7                 ↪ x+w//2], self.__blank_box_color):
8                 continue
9             number = self.__finder.find_number(x, y, w, h)
10            if number != None:
11                numbers[i] = int(number)
12                if only_first:
13                    return numbers
14
15            return numbers
```

**Codice 4.5:** Funzione per il riconoscimento dei valori delle tessere con meccanismo di cache

Le tessere associate a valori nel vettore diversi da 0 vengono proiettate sul vettore risultato chiamato `numbers`. Le tessere restanti vengono inizialmente controllate con il controllo del colore e successivamente con il riconoscimento del testo. Appena

viene riconosciuto il valore di una tessera, esso viene aggiornato nel vettore `numbers` e la funzione ritorna. Il nuovo vettore dunque differisce dal vettore `cache` solo per il valore di una tessera.

La potenza di questa funzione è che essa è debolmente legata alla dimensione della griglia, in quanto il numero di nuove tessere con valore sconosciuto.

In Figura 4.4 è mostrato il prodotto di tutto le operazioni di elaborazione della schermata di gioco, ovvero l'astrazione della griglia di gioco in un vettore di numeri.



4	0	0	4
0	0	2	16
0	0	8	32
0	0	16	128

**Figura 4.4:** Riconoscimento dei valori delle tessere

Equivalente al vettore `[4,0,0,4,0,0,2,16,0,0,8,32,0,0,16,128]`.

## 4.4 Astrazione della griglia di gioco

L'astrazione della griglia di gioco è stata realizzata con la classe `Graph2048` presente in `graph2048.py`. Essa rappresenta la griglia di gioco come un grafo, dove i nodi rappresentano le tessere e gli archi le connessioni tra le tessere.

Gli archi sono di due tipi (`Superior` e `Left`) e rappresentano rispettivamente la connessione tra una tessera e quella sopra superiore e la connessione tra una tessera e quella alla sua sinistra. Si è ritenuto superfluo aggiungere anche gli archi di tipo `Inferior` e `Right` in quanto essi sono ridondanti e possono essere ottenuti dagli archi già presenti. Non sono implementati metodi per la modifica della griglia in quanto essa non varia durante tutto il gioco.

L'oggetto, istanza della classe `Graph2048` mostrata in Codice 4.6, viene istanziato dopo il riconoscimento della griglia di gioco in modo da conoscere  $n$  ossia il lato della griglia. A questo punto viene generata tutta la struttura interna del grafo, ovvero i nodi e gli archi.

Il metodo `get_value()` restituisce i valori delle tessere presenti nella griglia, tale metodo prende in input il vettore dei numeri riconosciuti durante l'elaborazione del frame di gioco.

```
1 class Graph2048:
2     def __init__(self, n):
3         ...
4         for i in range(self.n**2):
5             self.nodes.append(Node(i))
6             if i % self.n != self.n-1:
7                 self.left.append(Left(i, i+1))
8             if i + self.n < self.n**2:
9                 self.superior.append(Superior(i, i+self.n))
10
11     def get_value(self, l):
12         value = []
13         for i in range(len(l)):
14             if l[i] != 0:
15                 value.append(Value(i, l[i]))
16         return value
```

**Codice 4.6:** Classe `Graph2048`

## 4.5 Connessione Vision e Think

Il modulo Vision è stato progettato per essere indipendente dal modulo Think. Tuttavia, per permettere una comunicazione tra i due moduli, è stato necessario implementare un sistema di comunicazione tra di essi.

Tale sistema è stato realizzato con l'utilizzo della libreria wrapper embASP per DLV. Essa permette di invocare DLV da Python e di passare i risultati ottenuti da DLV a Python.

Nel sottomodulo Vision di 2048 il sistema di comunicazione è stato implementato nel package `dlvsolution`. Esso contiene il file `dlvsolution.py` e il file `helpers.py`. Il primo contiene la classe `DLVSolution` che si occupa di invocare DLV mentre il secondo contiene le classi per rappresentare la conoscenza.

### 4.5.1 Rappresentazione della conoscenza

La conoscenza è rappresentata in DLV tramite un insieme di fatti e regole. I fatti rappresentano le informazioni estratte dall'immagine, come ad esempio i valori delle tessere. Le regole rappresentano le azioni che il modulo Think deve eseguire per risolvere il gioco, quest'ultimo aspetto verrà discusso nel prossimo capitolo.

I fatti sono rappresentati con embASP tramite la classe `Predicate`.

Per rappresentare i fatti necessari per risolvere il gioco 2048 sono state implementate le classi `Value`, `Node`, `Superior` e `Left`, tutte presenti in `helpers.py`. Tali classi estendono la classe `Predicate` e rappresentano rispettivamente il valore di una tessera, un nodo, un arco superiore e un arco sinistro.

Ai precedenti predicati si aggiungono `Output` e `Direction` che rappresentano il risultato dell'Intelligenza Artificiale.

Un esempio di classe `Node` è mostrato in Codice 4.7.

La struttura della classe è standard, essa contiene un attributo `predicate_name` che equivale al nome del predicato in ASP e un costruttore che inizializza gli attributi della classe. Tutto gli attributi devono avere un getter e un setter che rispetta il name convention `get_nome_attributo()` e `set_nome_attributo()`.

```

1 class Node(Predicate):
2     predicate_name = "node"
3
4     def __init__(self, id=None):
5         self.__id = id
6         Predicate.__init__(self, [("id", int)])
7
8     def get_id(self):
9         return self.__id
10
11     def set_id(self, id):
12         self.__id = id

```

**Codice 4.7:** Esempio di predicato con classe Node

## 4.5.2 Invocazione del solver

L'avvio del modulo Think è gestito dalla classe `DLVSolution` presente nel file `dlvsolution.py`. Essa si occupa di tutte le operazioni necessarie per invocare DLV e di passare i risultati ottenuti a Python.

Per poter invocare il solver è necessario istanziare l'oggetto `DLVSolution` con il path del solver che si intende utilizzare (nel nostro caso DLV) e successivamente occorre chiamare il metodo `start_asp()`. Questo metodo ha il compito di mappare i predicati in ASP con gli oggetti Python. Per fare ciò è necessario registrare le classi degli oggetti Python con il metodo `register_class()` di `ASPMapper`. Dopo aver fatto ciò, il metodo aggiunge l'encoding ASP con le regole necessarie per risolvere il gioco e i fatti statici, ossia quelli che non cambiano durante l'esecuzione del gioco. Tali fatti sono quelli generati da `Graph2048`, trattato in Sezione 4.4, e sono rappresentati dai nodi e dagli archi presenti nella griglia di gioco.

Per poter avviare il solver è necessario chiamare il metodo `recall_asp()` che si occupa di ricevere i risultati dal solver e di passarli a Python. Inizialmente il metodo rimuove i fatti dinamici della precedente iterazione e aggiunge i fatti dinamici della iterazione corrente. Successivamente il metodo avvia il solver e recupera gli answer sets, dopodichè restituisce l'azione da eseguire e il nuovo stato della griglia di gioco, quest'ultima utilizzata come cache (vedi Sezione 4.3.2). In caso non ci siano answer sets, il metodo setta il flag `gameOver` a `True` e ritorna `None, None`.



```

1 def start_asp(self, encoding :str, nodes : list, superior : list, left : list):
2     ASPMapper.get_instance().register_class(Node)
3     ASPMapper.get_instance().register_class(Superior)
4     ASPMapper.get_instance().register_class(Left)
5     ASPMapper.get_instance().register_class(Value)
6     ASPMapper.get_instance().register_class(Direction)
7     ASPMapper.get_instance().register_class(Output)
8
9     self.__init_fixed(encoding)
10    self.__init_static(nodes, superior, left)
11
12    self.__handler.add_program(self.__fixed_input_program)
13    self.__handler.add_program(self.__static_facts)

```

**Codice 4.8:** Metodo per l'impostazione del solver

```

1 def recall_asp(self, value : list):
2     if self.__index != None:
3         self.__handler.remove_program_from_id(self.__index)
4     self.__dynamic_facts = ASPInputProgram()
5     self.__init_dynamic(value)
6     self.__index = self.__handler.add_program(self.__dynamic_facts)
7     answer_sets : AnswerSets = self.__handler.start_sync()
8     oas = answer_sets.get_optimal_answer_sets()
9     if len(oas) == 0:
10        self.__gameOver = True
11        return None, None
12    dir = None
13    output = []
14    for answer_set in oas:
15        for obj in answer_set.get_atoms():
16            if isinstance(obj, Direction):
17                dir = obj.get_dir()
18            elif isinstance(obj, Output):
19                output.append(obj)
20    return dir, output

```

**Codice 4.9:** Metodo per l'invocazione del solver

# Capitolo 5

## Think

### Indice

---

<b>5.1</b>	<b>Scelte implementative . . . . .</b>	<b>46</b>
<b>5.2</b>	<b>Modellazione del problema . . . . .</b>	<b>47</b>
5.2.1	Unione delle tessere . . . . .	48
5.2.2	Posizionamento delle tessere . . . . .	50
<b>5.3</b>	<b>Valutazione delle mosse . . . . .</b>	<b>53</b>
5.3.1	Studio delle euristiche . . . . .	53
5.3.2	Implementazione delle euristiche . . . . .	54

---

## 5.1 Scelte implementative

E' stato necessario procedere fin da subito con una scelta implementativa non banale:

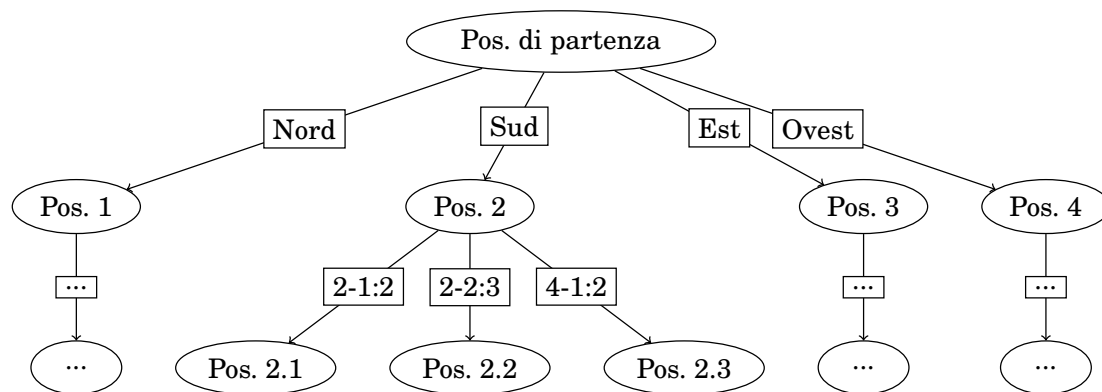
- **NO Look Ahead:** L'IA non va oltre la mossa corrente, si limita a valutare la situazione attuale e a prendere la mossa migliore.
- **Look Ahead:** L'IA valuta le mosse possibili e le conseguenze delle stesse fino ad un certo numero di mosse in avanti.

La scelta è ricaduta sulla prima opzione in quanto la seconda avrebbe richiesto un tempo di calcolo troppo elevato.

Per quanto le possibili mosse siano solo 4 (su, giù, destra, sinistra), lo stato successivo alla mossa è stocastico, quindi non è possibile prevedere con certezza il risultato di una mossa. Ciò è dovuto alle meccaniche del gioco, che dopo ogni mossa effettuata, inseriscono un nuovo blocco in una posizione casuale.

L'esposizione combinatoria generata da questa meccanica rende complesso il calcolo di una sequenza di mosse ottimali.

Un esempio semplificato di albero dei possibili stati su una griglia  $2 \times 2$  dopo una mossa è mostrato in Figura 5.1.



**NB:** La sintassi N-X:Y indica che viene inserito un blocco di valore N in posizione X,Y.

**Figura 5.1:** Albero delle possibili mosse

## 5.2 Modellazione del problema

Il problema è stato modellato su una struttura dati a grafo, dove i nodi rappresentano le celle dove possono essere posizionate le tessere e gli archi rappresentano le celle confinanti. Le celle che contengono una tessera sono quelle con un valore associato.

La parte più corposa dell'Intelligenza Artificiale è stata la generazione dello stato successivo ad una mossa. Tale generazione è fondamentale per poter scegliere la mossa migliore.

Per generare le possibili mosse è stato utilizzato il paradigma *Guess&Check*, che consiste nel generare tutte le possibili combinazioni su una scelta e poi scartare quelle che non rispettano i vincoli.

Inizialmente si effettua il *Guess* di tutte le possibili mosse, ossia le 4 direzioni cardinali.

```
1 direction(1) | direction(2) | direction(3) | direction(4).
```

**Codice 5.1:** Guess delle direzioni

In tale formalismo numerico il numero 1 rappresenta il movimento verso nord, 2 verso sud, 3 verso est e 4 verso ovest.

Si è scelto di rappresentare le direzioni con numeri per semplificare le regole di generazione degli stati successivi. Ad esempio, prendendo *direction(N)* con  $N < 3$  si prendono in considerazione le mosse che spostano le tessere tra le righe, mentre con  $N > 2$  si prendono in considerazione le mosse che spostano le celle sulle colonne.

Il calcolo dello stato successivo ad ogni mossa è diviso in due parti:

- **Unione delle tessere:** Vengono unite le tessere adiacenti con lo stesso valore, rispettando le regole del gioco.
- **Posizionamento delle tessere:** Vengono spostate le tessere in base alla direzione scelta.

La scelta di dividere il calcolo dello stato successivo in due parti è stata presa per rendere la modellazione più semplice e leggibile.

Nel proseguo della trattazione si fa riferimento varie volte al concetto di nodi e tessere. I nodi equivalgono alle celle della griglia di gioco, mentre le tessere sono le celle con un valore assegnato, ovvero i nodi con un valore associato.

### 5.2.1 Unione delle tessere

Per effettuare il merge delle tessere adiacenti sono state implementati dei predicati ausiliari, in Codice 5.2:

- **reachable/2**: contiene le coppie di nodi che sono raggiungibili tra loro in una direzione specifica, se sono adiacenti in quella direzione o se non sono presenti tessere tra loro.
- **mergeable/2**: contiene le coppie di nodi che possono essere uniti tra loro, ovvero che sono raggiungibili e hanno lo stesso valore.

```

1 reachable(Node1, Node2):- direction(D), superior(Node1, Node2), D < 3.
2 reachable(Node1, Node2):- direction(D), left(Node1, Node2), D > 2.
3 reachable(Node1, Node2) :- reachable(Node1, Node3), reachable(Node3, Node2), not
   ↪   valued(Node3).
4
5 mergeable(Node1, Node2) :- reachable(Node1, Node2), value(Node1, Value),
   ↪   value(Node2, Value).

```

**Codice 5.2:** Predicati ausiliari per l'unione delle tessere

Riapplicando il paradigma *Guess&Check*, com'è stato fatto sulla scelta delle mosse, si è scelto di generare tutte le possibili combinazioni di unione tra le tessere adiacenti e di ignorare le combinazioni che non rispettano le regole del gioco.

In prima fase è stata effettuata l'operazione di *Guess*, in Codice 5.3.

In **inMerge/2** sono presenti tutti le unioni che vanno effettuate, mentre in **outMerge/2** sono presenti tutti le unioni che non si effettuano.

```

1 inMerge(Node1, Node2) | outMerge(Node1, Node2) :- mergeable(Node1, Node2).

```

**Codice 5.3:** Guess delle unioni tra tessere

In seconda fase è stata effettuata l'operazione di *Check* del paradigma, in Codice 5.4. In questa fase vengono scartate le combinazioni che non rispettano le regole del gioco.

```

1 merged(Node) :- inMerge(Node, _).
2 merged(Node) :- inMerge(_, Node).
3
4 :- outMerge(Node1, Node2), not merged(Node1), not merged(Node2).
5 :- inMerge(_, Node), inMerge(Node, _).
6
7 :- direction(1), outMerge(Node1, Node2), inMerge(Node2, Node3), upper(Node1,
   ↪ Node3).
8 :- direction(2), outMerge(Node1, Node2), inMerge(Node3, Node1), upper(Node3,
   ↪ Node1).
9 :- direction(3), outMerge(Node1, Node2), inMerge(Node2, Node3), lower(Node1,
   ↪ Node3).
10 :- direction(4), outMerge(Node1, Node2), inMerge(Node3, Node1), lower(Node3,
   ↪ Node1).

```

**Codice 5.4:** Check delle unioni tra tessere

Nel predicato ausiliario `merged/1` sono inseriti i nodi che partecipano a unioni tra tessere.

Sono presenti 3 regole da rispettare, implementate in Codice 5.4:

- Se due tessere possono unirsi tra loro e se tale unione è stata scartata almeno una delle due tessere deve essere coinvolta in un'altra unione. Tale regola è verificata tramite il vincolo a riga 6.
- Se un nodo è coinvolto in un'unione non può essere coinvolto in un'altra unione. Tale regola è verificata tramite il vincolo a riga 7.
- Le tessere devono essere unite rispettando l'ordine impartito dalla direzione scelta. Tale regola è verificata tramite i vincoli alle righe 9-12. Ognuno di questi vincoli è specifico per una direzione.

Una volta applicati i vincoli si ottiene deterministicamente un'unica combinazione di unioni tra tessere, possibile per la mossa scelta.

Fatto ciò si procede quindi con l'applicazione delle unioni, in Codice 5.5.

Viene definito il predicato `partialOutput/2` che restituisce il valore dei nodi dopo l'unione delle tessere.

```

1 partialOutput(Node, Value) :- notMerged(Node), value(Node, Value).
2 partialOutput(Node, Value) :- inMerge(Node, _), value(Node, Value1), Value =
  ↪ Value1 * 2.

```

**Codice 5.5:** Generazione dello stato parziale dopo le unioni

Le tessere non coinvolte in un'unione rimangono invariate, invece le tessere unite vengono sostituite con un'unica tessera con valore doppiato, posizionata nel nodo della prima tessera coinvolta nell'unione.

Si noti come il predicato `notMerged/1` sia l'opposto del predicato `merged/1`.

## 5.2.2 Posizionamento delle tessere

Una volta unite le tessere si procede con il posizionamento delle stesse nella posizione corretta.

Per svolgere tale operazione sono stati implementati dei predicati ausiliari, in Codice 5.6:

- `tempValuated/1`: contiene i nodi con valore dopo l'operazione di unione, va a sostituire il predicato `valued/1` presente nella prima fase.
- `indirectReachable/2`: contiene le coppie di nodi che sono raggiungibili in una direzione, estende il predicato `reachable/2` con la possibilità di raggiungere tutti i nodi presenti in una direzione, anche con tessere intermedie.
- `sorted/2`: contiene la posizione su una riga o colonna di un nodo a seconda se la direzione è verso sopra/sotto o sinistra/destra.

Tramite tali predicati ausiliari si è in grado di procedere alla generazione degli stati successivi.

Inizialmente si va a popolare il predicato `blank/2`, che associa ad ogni nodo il numero di nodi senza valore presenti in una direzione; tale numero ci servirà per calcolare l'offset necessario per traslare le tessere nella posizione corretta.

Con il numero di offset calcolato si è in grado di generare il predicato `mapping/2`, che associa ad ogni tessera la sua posizione dopo il riposizionamento.

```

1 tempValuated(Node) :- partialOutput(Node, _).
2
3 indirectReachable(Node1, Node2) :- reachable(Node1, Node2).
4 indirectReachable(Node1, Node2) :- reachable(Node1, Node3),
   ↳ indirectReachable(Node3, Node2).
5
6 sorted(Node, Pos, r) :- direction(D), D < 3, firstUpper(Node), Pos = 1.
7 sorted(Node, Pos, r) :- direction(D), D < 3, superior(Node2, Node), sorted(Node2,
   ↳ Pos1, r), Pos = Pos1 + 1.
8 sorted(Node, Pos, c) :- direction(D), D > 2, firstLeftier(Node), Pos = 1.
9 sorted(Node, Pos, c) :- direction(D), D > 2, left(Node2, Node), sorted(Node2,
   ↳ Pos1, c), Pos = Pos1 + 1.

```

**Codice 5.6:** Predicati ausiliari per il posizionamento delle tessere

Entrambi i predicati sono fortemente dipendenti dalla direzione scelta, infatti si noti come il predicato `blank/2` e `mapping/2` siano definiti 4 volte, una per ogni direzione.

Questi predicati sono implementati in Codice 5.7.

```

1 blank(Node, N) :- direction(1), partialOutput(Node, _), N = #count{Node2:
   ↳ indirectReachable(Node2, Node), not tempValuated(Node2)}.
2 blank(Node, N) :- direction(2), partialOutput(Node, _), N = #count{Node2:
   ↳ indirectReachable(Node, Node2), not tempValuated(Node2)}.
3 blank(Node, N) :- direction(3), partialOutput(Node, _), N = #count{Node2:
   ↳ indirectReachable(Node2, Node), not tempValuated(Node2)}.
4 blank(Node, N) :- direction(4), partialOutput(Node, _), N = #count{Node2:
   ↳ indirectReachable(Node, Node2), not tempValuated(Node2)}.
5
6 mapping(Node1, Node2) :- direction(1), sorted(Node1, Pos1, r), sorted(Node2,
   ↳ Pos2, r), blank(Node1, Offset), partialOutput(Node1, _), Pos2 = Pos1 -
   ↳ Offset, inCol(Node1, Node2).
7 mapping(Node1, Node2) :- direction(2), sorted(Node1, Pos1, r), sorted(Node2,
   ↳ Pos2, r), blank(Node1, Offset), partialOutput(Node1, _), Pos2 = Pos1 +
   ↳ Offset, inCol(Node1, Node2).
8 mapping(Node1, Node2) :- direction(3), sorted(Node1, Pos1, c), sorted(Node2,
   ↳ Pos2, c), blank(Node1, Offset), partialOutput(Node1, _), Pos2 = Pos1 -
   ↳ Offset, inRow(Node1, Node2).
9 mapping(Node1, Node2) :- direction(4), sorted(Node1, Pos1, c), sorted(Node2,
   ↳ Pos2, c), blank(Node1, Offset), partialOutput(Node1, _), Pos2 = Pos1 +
   ↳ Offset, inRow(Node1, Node2).

```

**Codice 5.7:** Generazione del mapping dei nodi

Una volta calcolato il mapping dei nodi siamo in grado di fare due operazioni fondamentali.



La prima è la verifica che la direzione scelta sia valida, in Codice 5.8, ovvero che almeno una tessera sia stata spostata oppure è avvenuta almeno un'unione tra tessere. Non è possibile fare mosse che non alterino lo stato del gioco.

Il vincolo coincide con il *Check* della direzione che completa il *Guess* presentato in Codice 5.1.

```
1 mapChanged :- mapping(Node1, Node2), Node1 != Node2.  
2 mapChanged :- inMerge(_, _).  
3 :- not mapChanged.
```

**Codice 5.8:** Check della direzione

La seconda operazione, implementata in Codice 5.9, è la generazione dello stato successivo alla mossa su cui valutare la qualità della mossa stessa.

Il predicato `output/2` restituisce il valore dei nodi dopo il posizionamento delle tessere, esso contiene la nuova griglia dopo la mossa, esclusa fatta per la tessera che viene generata casualmente dal gioco dopo la mossa.

Pertanto le informazioni contenute nel predicato vengono passate al modulo *Vision* per utilizzare il meccanismo di caching, presentato in Sezione 4.3.2.

```
1 output(Node2, Value) :- mapping(Node1, Node2), partialOutput(Node1, Value).
```

**Codice 5.9:** Generazione dello stato successivo alla mossa

## 5.3 Valutazione delle mosse

Una volta generati tutti gli stati successivi alle mosse possibili si procede con la valutazione delle mosse.

Non disponendo della potenza di calcolo necessaria per valutare tutte le mosse possibili, si è scelto di utilizzare delle euristiche per valutare le mosse.

Le euristiche sono delle funzioni che permettono di valutare la qualità di una mossa in base a delle informazioni parziali che in questo caso sono le tessere presenti nella griglia dopo la mossa. In particolare si è scelto di ignorare le tessere generate casualmente dal gioco dopo la mossa.

### 5.3.1 Studio delle euristiche

Per sviluppare le euristiche della nostra intelligenza artificiale si è scelto di basarsi sulle informazioni contenute nella pubblicazione "Composition of Basics Heuristics for the Game 2048"[2] prodotta da un gruppo di ricercatori del California Polytechnic State University. Tale paper analizza varie euristiche per valutare le mosse in 2048 e le confronta tra loro.

Nonostante il paper in questione implementa un algoritmo di ricerca minimax, le euristiche possono essere utilizzate per valutare le mosse in un contesto di ricerca locale come quello implementato in questo progetto.

Esistono varie euristiche per valutare le mosse in 2048, quelle valutate nel paper sono le seguenti:

- **Greedy:** Valuta la mossa in base al nuovo valore del punteggio ufficiale del gioco. Maggiore è il punteggio, maggiore è il valore dell'euristica.
- **Empty:** Valuta la mossa in base al numero di celle vuote presenti nella griglia. Maggiore è il numero di celle vuote, maggiore è il valore dell'euristica.
- **Uniformity:** Valuta la mossa in base alla distribuzione dei valori delle tessere nella griglia. Maggiore è la varianza dei valori delle tessere, minore è il valore dell'euristica.
- **Monotonicity:** Valuta la mossa in base alla monotonicità della griglia. Una griglia è monotona se le tessere sono ordinate in modo crescente o decrescen-

te in ogni riga e in ogni colonna. Maggiore è la monotonicità della griglia, maggiore è il valore dell'euristica.

A queste euristiche si è aggiunta una euristica casuale, che valuta la mossa in modo casuale al fine di evitare situazioni di stallo.

Dalla pubblicazione appena citata è stata presa in considerazione la combinazione di euristiche che ha ottenuto i risultati migliori, ossia la combinazione *empty, monotonicity, random*. Questa combinazione impone che la mossa migliore sia quella che massimizza il numero di celle vuote e che in caso di pareggio si preferiscono le mosse che contribuiscono maggiormente alla monotonicità della griglia e, in caso di ulteriore pareggio, si sceglie la mossa casualmente tra le mosse possibili.

Nel nostro caso si è scelto di partire da tale combinazione con l'aggiunta di ulteriori euristiche per valutare le mosse.

La prima euristica aggiunta è la preferenza per le mosse che creano celle adiacenti con lo stesso valore. Questa euristica è stata aggiunta per favorire le unioni alle mosse successive.

La seconda euristica aggiunta è la preferenza nel non fare mosse verso l'alto, tranne nel caso in cui non ci siano altre mosse possibili. Questa euristica permette di bloccare le tessere con valore maggiore in basso.

### 5.3.2 Implementazione delle euristiche

Nel nostro caso le euristiche sono state implementate tramite "weak constraints" in ASP. I weak constraints sono delle regole che possono essere violate, ma che se rispettate permettono di ottenere una soluzione migliore rispetto a quelle che le violano.

Le regole che implementano le euristiche sono presentate in Codice 5.10:

1. La prima regola (riga 1) punisce le mosse verso l'alto. Dunque non ci sarà mai una mossa verso l'alto se ci sono altre mosse possibili.
2. La seconda regola (riga 3) punisce le mosse ogni qual volta un nodo ha un valore associato, dunque favorisce le mosse che creano griglie con meno tessere, favorendo quindi le mosse con più unioni.
3. La terza regola (riga 5) punisce le mosse ogni qual volta che una tessera più in alto a sinistra rispetto ad un'altra ha un valore maggiore rispetto alla seconda.

Questa regola favorisce le mosse che creano griglie con tessere ordinate in modo crescente da sinistra a destra e dall'alto in basso.

4. La quarta regola (riga 7-8) punisce le mosse ogni qual volta due tessere adiacenti non hanno lo stesso valore. Questa regola favorisce le mosse che creano griglie con tessere adiacenti con lo stesso valore, in modo da favorire le unioni alla mossa successiva.

L'euristica casuale è indirettamente implementata nel linguaggio *ASP* dato che, se i criteri di valutazione delle mosse danno lo stesso valore ad una mossa, quella scelta sarà la prima trovata dal risolutore.

```
1 :~ direction(1). [4@1]
2
3 :~ output(Node, Value). [1@3, Node]
4
5 :~ output(Node1, Value1), output(Node2, Value2), sorted(Node1, PosR1, r),
   ↳ sorted(Node2, PosR2, r), sorted(Node1, PosC1, c), sorted(Node2, PosC2, c),
   ↳ Pos1 = PosR1 + PosC1, Pos2 = PosR2 + PosC2, Pos1 < Pos2, Value1 > Value2.
   ↳ [1@2, Node1, Node2]
6
7 :~ output(Node1, Value1), output(Node2, Value2), superior(Node1, Node2), Value1
   ↳ != Value2. [1@1, Node1, Node2]
8 :~ output(Node1, Value1), output(Node2, Value2), left(Node1, Node2), Value1 !=
   ↳ Value2. [1@1, Node1, Node2]
```

**Codice 5.10:** Valutazione delle mosse

# Capitolo 6

## Conclusioni

### Difficoltà incontrate

Il lavoro svolto si è rivelato nascondere difficoltà in tutti i moduli del progetto.

Nel modulo Acting è stato necessario studiare e comprendere un software sviluppato da terzi, non debitamente documentato e sviluppato con vari framework e linguaggi al tesista sconosciuti. Ci si è scontrati con problemi di carattere fisico:

- A causa di un iniziale problema di messa a terra, i dispositivi mobili non riuscivano a registrare i tocchi del robot.
- Il robot non riusciva a muoversi in modo fluido a causa di problemi sul canale seriale che collegava il robot al computer.
- Il robot si muoveva faraginosamente a causa di un sistema di controllo non ottimizzato per operare con un robot di tali dimensioni.

Tali problemi non conosciuti inizialmente hanno richiesto uno sforzo non indifferente per essere risolti e hanno rallentato il lavoro.

Nel modulo Vision si sono dovute sperimentare varie tecniche di visione artificiale per riuscire a riconoscere con accuratezza i componenti del gioco. Una volta trovata la tecnica migliore è stata necessaria una fase di ottimizzazione per ridurre i tempi di esecuzione a valori approssimati al secondo.

Nel modulo Think si è dovuto affrontare il non determinismo del gioco e la necessità di prendere decisioni in un tempo accettabile. La ricerca di un euristica

valida ha richiesto molti tentativi, uno studio approfondito del gioco e di varie ricerche in letteratura e su internet.

## **Risultati ottenuti**

Il lavoro finale è stato soddisfacente. Si è riusciti a sviluppare un sistema fluido e funzionante che permette di giocare a giochi per dispositivi mobili con un bassa percentuale di insuccesso.

Il robot si è dimostrato valido anche in movimenti difficili e rapidi, riuscendo a risolvere oltre a 2048 anche altri giochi sviluppati nell'ambito del progetto BrainyBot.

Il software di controllo è stato rinnovato e ottimizzato per operare con robot e dispositivi diversi, in modo totalmente modulare, attraverso l'uso di file di configurazione specifici per ogni robot e dispositivo, intercambiabili tra loro. L'applicativo web associato è stato rinnovato nella veste grafica e nelle funzionalità.

La libreria di visione artificiale su cui si basa il progetto BrainyBot è stata espansa con nuove primitive per il riconoscimento di testo e di forme geometriche, sia singlethread che multithread per scalare su macchine più potenti.

Il modulo Think è stato sviluppato con successo riuscendo a giocare a 2048 come un essere umano, con un'euristica che si è dimostrata valida anche con griglie di dimensioni diverse.

Non da sottovalutare sono state le competenze acquisite durante il progetto, sia in ambito di robotica che di visione e di intelligenza artificiale. Con molta probabilità tali competenze diverranno utili in futuro.

## **Possibili sviluppi futuri**

Il progetto BrainyBot è in continua evoluzione e sono molte le funzionalità che si potrebbero implementare per migliorare un sistema già valido.

Si potrebbe espandere BB2 con un modulo Wi-Fi o Bluetooth per permettere la comunicazione wireless senza l'uso di cavi.

Si potrebbe espandere la flotta di robot a disposizione di BrainyBot con ulteriori robot in modo da poter fare sfide tra robot per trovare le configurazioni migliori.

Nella libreria di Vision si potrebbero aggiungere nuove primitive per il riconoscimento di forme geometriche più complesse e per l'individuazione di oggetti in movimento. Si potrebbero inoltre impiegare nuove tecniche di visione artificiale per migliorare la precisione e la velocità di riconoscimento dei componenti dei giochi.

La connessione tra il modulo Vision e il modulo Think potrebbe essere riscritta con una nuova libreria meno prolissa e più efficiente.

Il modulo Think di 2048 potrebbe implementare una ricerca della soluzione migliore con Look Ahead, disponendo di una potenza di calcolo maggiore in grado di sopportare una tale esplosione combinatoria di possibili posizioni.

Il progetto BrainyBot ha ancora molto da offrire e con certezza si può affermare che in futuro verrà sviluppato ulteriormente al fine di espandere le sue funzionalità e migliorare le sue prestazioni.

# Appendice A

## Algoritmo di Canny

L'algoritmo di Canny è un algoritmo di rilevamento dei bordi sviluppato da John F. Canny nel 1986.

L'algoritmo di Canny è composto da cinque passi:

1. **Riduzione del rumore**
2. **Calcolo del gradiente**
3. **Soglia di magnitudine del gradiente**
4. **Soppressione dei non massimi**
5. **Isteresi**

**Noise Reduction** Il primo passo dell'algoritmo di Canny è la riduzione del rumore. Per fare ciò l'immagine viene filtrata con un filtro gaussiano. Il filtro gaussiano è un filtro lineare che calcola la convoluzione tra l'immagine e una funzione gaussiana 5 x 5.

$$B = \frac{1}{159} \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * A. \quad (\text{A.1})$$

A è l'immagine originale, B è l'immagine filtrata.



**Calcolo del gradiente** Il secondo passo dell'algoritmo di Canny è il calcolo del gradiente. Vengono calcolate le derivate parziali dell'immagine per ottenere la magnitudine e la direzione del gradiente. Per fare ciò vengono utilizzati due filtri di Sobel, vedi Equazione (A.2), uno per calcolare la derivata rispetto all'asse x e uno per calcolare la derivata rispetto all'asse y.

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * A \quad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A \quad (\text{A.2})$$

La magnitudine del gradiente e la direzione del gradiente vengono calcolate rispettivamente con le Equazioni (A.3) e (A.4).

$$Edge\_Gradient(G) = \sqrt{G_x^2 + G_y^2} \quad (\text{A.3})$$

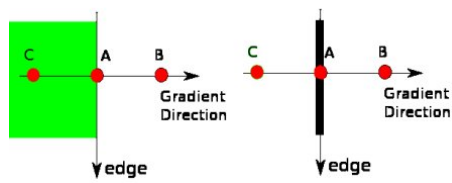
$$Edge\_Angle(\theta) = \arctan\left(\frac{G_y}{G_x}\right) \quad (\text{A.4})$$

**Soglia di magnitudine del gradiente** Il terzo passo dell'algoritmo di Canny è la selezione dei pixel che hanno una magnitudine del gradiente superiore ad una certa soglia. Questo passo è importante per ridurre il numero di pixel che vengono considerati come bordi.

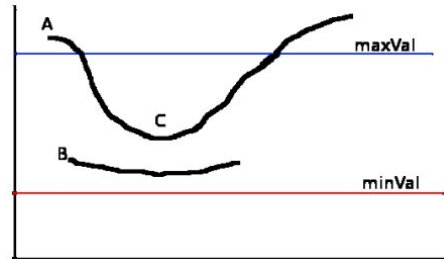
**Soppressione dei non massimi** Il quarto passo dell'algoritmo di Canny è la soppressione dei non massimi. In questo passo vengono eliminati i pixel che non sono massimi locali nella direzione del gradiente. Questo passo è importante per ottenere bordi sottili.

In Figura A.1 C e A hanno la stessa magnitudine del gradiente, ma A è un massimo locale, mentre C non lo è. Quindi C viene eliminato. Dopo la soppressione dei non massimi, viene eseguito un doppio taglio di soglia alto e basso per ridurre ulteriormente il numero di pixel considerati come bordi.

**Isteresi** Il quinto passo dell'algoritmo di Canny è l'isteresi. In questo passo vengono selezionati i pixel che sono connessi ai pixel di bordo. Questo passo è importante per ottenere bordi continui. In Figura A.2 A supera la soglia alta, quindi viene selezionato. B e C superano la soglia bassa, ma non la soglia alta, quindi si passa



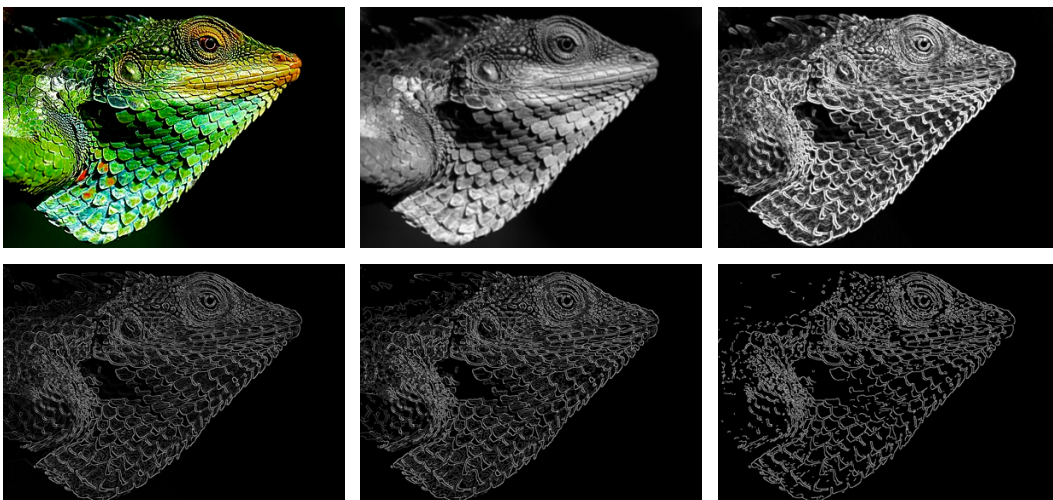
**Figura A.1:** Riduzione dei massimi



**Figura A.2:** Isteresi

al controllo dei pixel adiacenti. C è connesso ad A; che è un bordo, quindi viene selezionato. B non è collegato a nessun bordo, quindi viene eliminato.

**Esempio** Nella Figura A.3 sono mostrate tutte le fasi dell'algoritmo di Canny.



**Figura A.3:** Fasi dell'algoritmo di Canny

# Riferimenti bibliografici

## Bibliografia

- [1] Thomas Eiter, Giovambattista Ianni e Thomas Krennwallner. «Answer Set Programming: A Primer». In: vol. 5689. 2009. ISBN: 978-3-642-03753-5. DOI: 10.1007/978-3-642-03754-2\_2.
- [2] Iris Kohler, Theresa Migler e Foaad Khosmood. «Composition of basic heuristics for the game 2048». In: *Proceedings of the 14th International Conference on the Foundations of Digital Games*. 2019. DOI: 10.1145/3337722.3341838. URL: <https://doi.org/10.1145/3337722.3341838>.

## Sitografia

- [3] IEEE Spectrum. *The Making of Arduino*. URL: <https://spectrum.ieee.org/the-making-of-arduino>.
- [4] Robotis. *Dynamixel XL430-W250-T Servo*. URL: <https://emanual.robotis.com/docs/en/dxl/x/xl430-w250/>.
- [5] Robotis. *Dynamixel Shield*. URL: [https://emanual.robotis.com/docs/en/parts/interface/dynamixel\\_shield/](https://emanual.robotis.com/docs/en/parts/interface/dynamixel_shield/).
- [6] Robotis. *Dynamixel2Arduino*. URL: <https://github.com/ROBOTIS-GIT/Dynamixel2Arduino>.
- [7] AI4Business. *Reymond Clavel, il genio dei robot paralleli*. URL: <https://www.ai4business.it/intelligenza-artificiale/gli-inventori-dellai/reymond-clavel-il-genio-dei-robot-paralleli/>.
- [8] OpenCV. *OpenCV*. URL: <https://opencv.org/about>.

- [9] NumPy. *NumPy*. URL: <https://numpy.org/doc/stable/>.
- [10] OpenCV. *Canny Edge Detection*. URL: [https://docs.opencv.org/master/da/d22/tutorial\\_py\\_canny.html](https://docs.opencv.org/master/da/d22/tutorial_py_canny.html).
- [11] Wikipedia. *Canny Edge Detection Algorithm*. URL: [https://en.wikipedia.org/wiki/Canny\\_edge\\_detector](https://en.wikipedia.org/wiki/Canny_edge_detector).
- [12] OpenCV. *Contours*. URL: [https://docs.opencv.org/master/d4/d73/tutorial\\_py\\_contours\\_begin.html](https://docs.opencv.org/master/d4/d73/tutorial_py_contours_begin.html).
- [13] Wikipedia. *Optical Character Recognition (OCR)*. URL: [https://it.wikipedia.org/wiki/Riconoscimento\\_ottico\\_dei\\_caratteri](https://it.wikipedia.org/wiki/Riconoscimento_ottico_dei_caratteri).
- [14] Tesseract-OCR. *Tesseract*. URL: <https://github.com/tesseract-ocr/tesseract>.
- [15] Wikipedia. *2048*. URL: [https://it.wikipedia.org/wiki/2048\\_\(videogioco\)](https://it.wikipedia.org/wiki/2048_(videogioco)).
- [16] Potassco. *Clingo*. URL: <https://potassco.org/clingo/>.
- [17] DLV System. *DLV2*. URL: <http://www.dlvsystem.com/dlv/>.
- [18] Jason Huggins. *Tapster Bot*. URL: <https://github.com/hugs/tapsterbot>.
- [19] Git. *Git*. URL: <https://git-scm.com/>.
- [20] Arduino. *Firmata*. URL: <https://www.arduino.cc/reference/en/libraries/firmata/>.
- [21] Arduino. *Software Serial*. URL: <https://docs.arduino.cc/learn/built-in-libraries/software-serial/>.
- [22] PJRC. *AltSoftSerial*. URL: [https://www.pjrc.com/teensy/td\\_libs\\_AltSoftSerial.html](https://www.pjrc.com/teensy/td_libs_AltSoftSerial.html).
- [23] Guntis. *Tappy*. URL: <https://github.com/guntiss/tappy>.
- [24] Johnny-Five. *Johnny-Five*. URL: <http://johnny-five.io/>.