

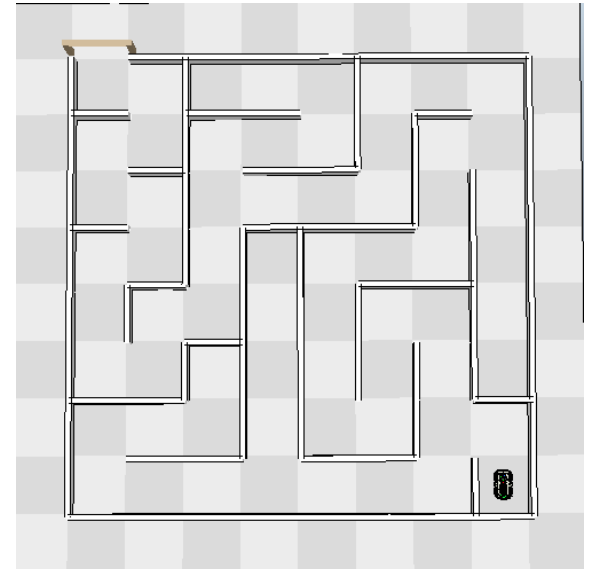
Intelligent Systems and Robotics Laboratory

*Esplorazione e risoluzione del labirinto
con il Robot Freenove4WD*

Obiettivo del progetto

Obiettivo del progetto

- Esplorazione e risoluzione del labirinto con il robot FreeNove4WD
 - CoppeliaSim
 - Robot “reale”
- Struttura non nota al robot
- Algoritmo realizzato **simile** al DFS



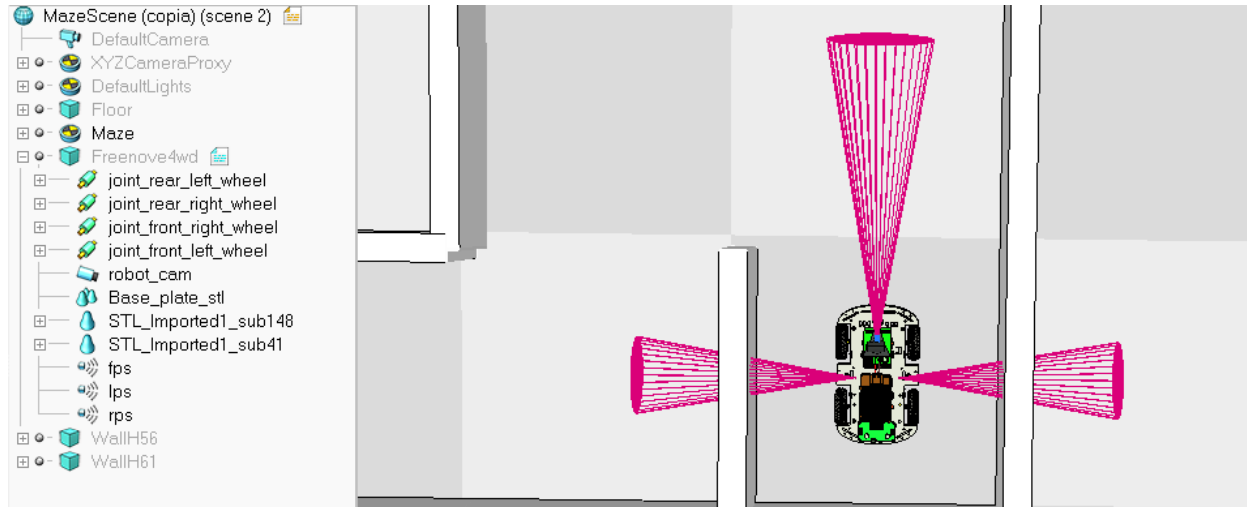
Vincoli del progetto

Vincoli del progetto: Labirinto

- Un solo punto iniziale (**root**) e un solo punto finale (**final**)
- Rappresentato da un albero semplice: **no cicli**
- **Non** si può passare per uno **stesso punto** se si esplora in **profondità**
- Muri che formano angoli di 90° e 180°
- Nella robotica virtuale:
 - Muro con lunghezza minima di 0.45m e massima di 0.50m
 - Spessore del muro deve essere almeno di 0.10m
 - Distanza tra due muri deve essere almeno di 0.45m e massima di 0.50m

Vincoli del progetto: Robot

- Labirinto sconosciuto e uscita sconosciuta
- Deve essere in grado di osservare ciò che è attorno a lui
- Sensori di prossimità: **Front, Left, Right**



Algoritmo

Algoritmo

- Inizio esplorazione da un **punto iniziale** (**root/stato iniziale**)
- Lettura sensori di prossimità ed elaborazione dei dati (tra cui lo stato del robot)
- Decide il verso e direzione in cui si deve muovere
- Se incontra un **vicolo cieco**: ripercorre il percorso intrapreso fino a quando non trova un **nuovo percorso da esplorare**
- Esplorazione si conclude quando il robot trova il **punto finale** ovvero l'uscita (**final**)

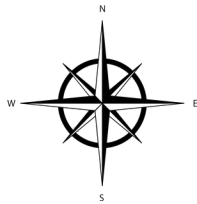
Algoritmo: cosa svolge il Controller

- Contiene l'algoritmo principale per l'esplorazione
- Architettura **Sense - Think - Act**
- Prende decisioni e invia comandi al **PhysicalBody**
- Decisioni prese in base a:
 - Valori dei sensori: **Front, Left, Right**
 - **Lista di priorità**
 - Stato del Robot: **State**
 - Posizione: **Position**
 - Modalità di esplorazione: **Mode**
 - Albero del labirinto: **Tree**

```
def algorithm(self):  
    """ Algorithm used to explore and solve the maze """  
  
    if self.goal_reached():  
        return True  
  
    # SENSE  
    self.read_sensors()  
    self.left_values.append(self.left_value)  
    self.front_values.append(self.front_value)  
    self.right_values.append(self.right_value)  
  
    # THINK  
    actions, com_actions = self.control_policy()  
    com_action = self.decision_making_policy(com_actions)  
    command, action = com_action  
  
    # Update of the tree  
    self.update_tree(actions, action)  
  
    # ACT  
    performed = self.do_action(com_action)  
  
    return False
```

Lista di priorità

- Lista costituita dai 4 punti cardinali:
NORTH, SOUTH, EAST, WEST



- La funzione **ControlPolicy** genera coppie di comandi e azioni:
[command, action]
- Una di queste coppie deve essere eseguita dal robot

- La posizione nella lista determina la priorità:
 - punto cardinale in **prima posizione** ha priorità più **alta**
 - **ultima posizione** priorità più **bassa**
- La coppia che deve essere eseguita viene scelta dalla funzione **DecisionMakingPolicy** in base alla lista di priorità

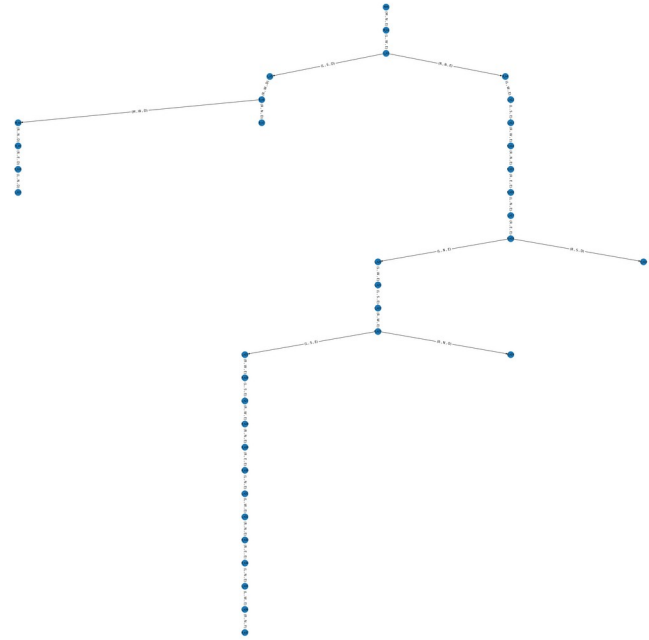
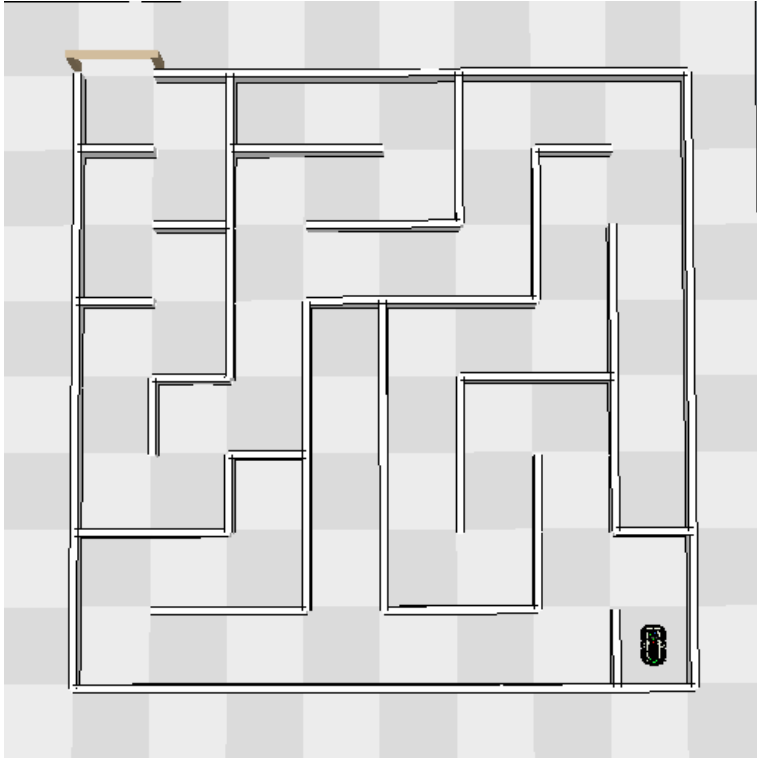
Lista di priorità: Esempio

- `priority_list = <SOUTH, WEST, EAST, NORTH>`
- Dalla lista di priorità si può dedurre che la DMP sceglierà di andare a **SOUTH** dato che ha priorità più alta e, se non fosse possibile, sceglierà **WEST** e poi preferirà **EAST** prima di **NORTH**.

Imparare dalle esperienze passate: Albero

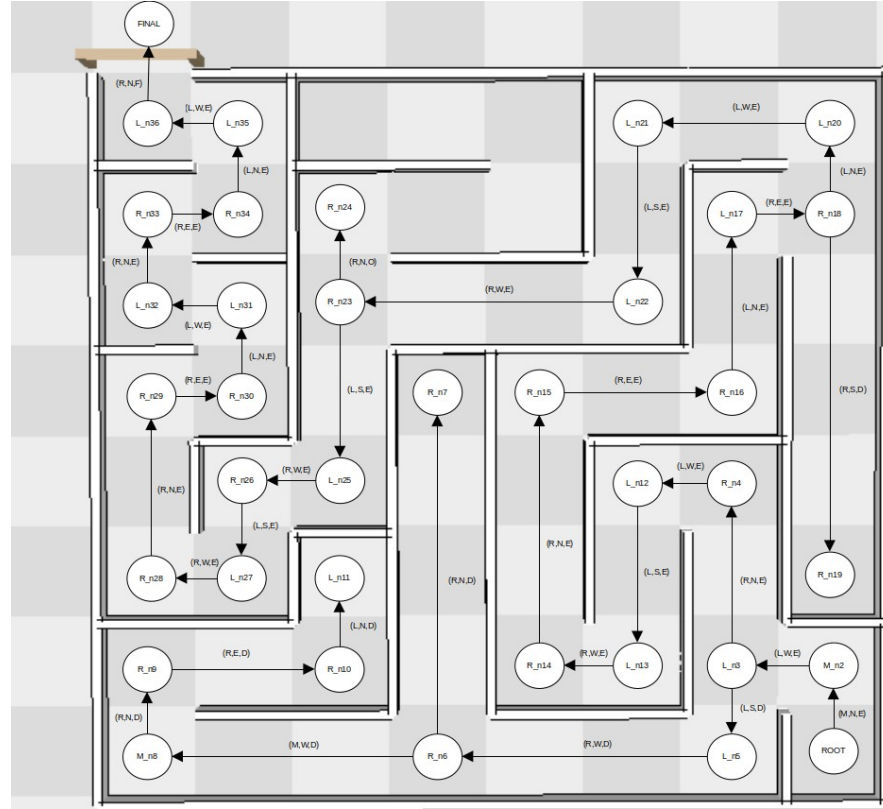
- Albero dell'intero labirinto è **unico**
- La struttura dell'albero/sotto-albero del labirinto generato dal robot **dipende** da come esso viene esplorato
- Robot potrebbe **non esplorare** l'intero labirinto ma solo una parte di esso
 - Dipende dalla lista di priorità scelta e dall'intelligenza
- Nell'esplorazione genera nodi in parti specifiche del labirinto
- Ogni nodo ha diversi attributi:
 - Name, Action, Type, Parent, Left, Mid, Right

Esempio: `priority_list = <SOUTH, WEST, EAST, NORTH>`



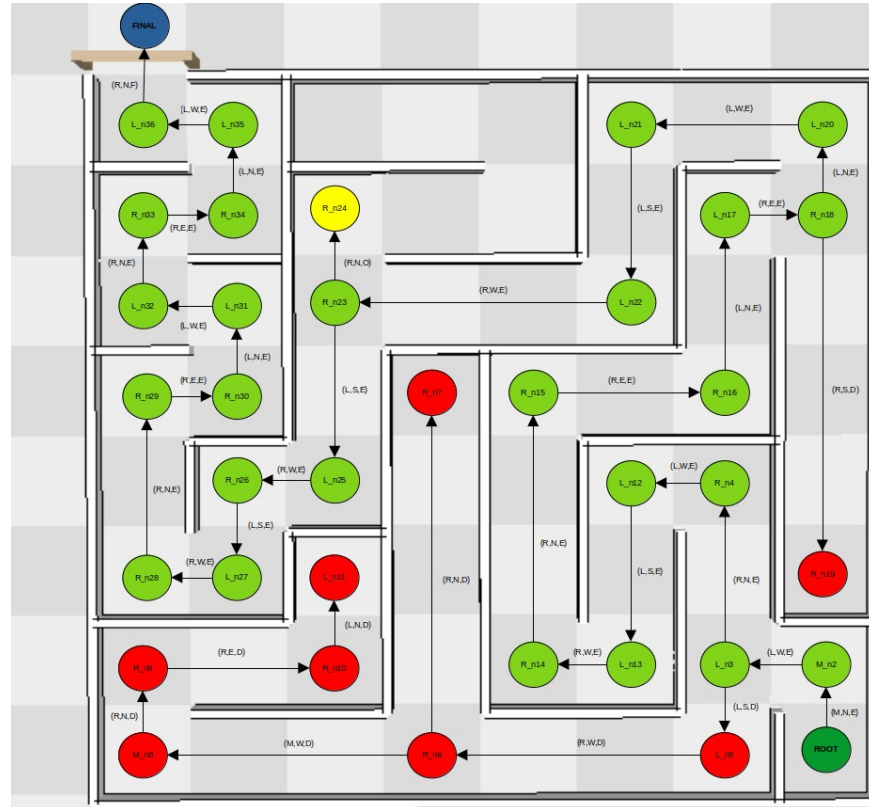
Labirinto + Albero

<S, W, E, N>



Labirinto + Albero: differenziare i nodi generati in base al tipo (Type)

<S, W, E, N>



Verde scuro: **ROOT**
Verde: **EXPLORED**
Giallo: **OBSERVED**
Rosso: **DEAD_END**
Blu: **FINAL**