Relazione Progetto Programmazione Avanzata

Leonardo Salpini MAT:114525

RESPONSABILITÀ E RELATIVE IMPLEMENTAZIONI

- Rappresentazione di Robot
- Rappresentazione di una forma
- Definizione ambiente di gioco
- Creazione di un robot
- Generazione programma
- Generazione ambiente di gioco
- Rappresentazione di un punto sul piano di gioco
- Direzione del movimento di un robot
- Rappresentazione di un singolo Comando
- Esecuzione del programma

Rappresentazione di Robot:
I robot sono rappresentati come oggetti che implementano l'interfaccia *RobotInterface*. Essi contengono informazioni sulle loro caratteristiche e possono essere dotati di etichette (labels) per indicare la condizione che stanno segnalando in quel momento. La classe che implementa questa interfaccia è la classe *Robot.*

Rappresentazione di una Forma:
Le forme sono oggetti che implementano l'interfaccia ShapesInterface. Ogni forma è caratterizzata da un nome (label) e da un set di coordinate. Queste informazioni vengono utilizzate per rappresentare gli elementi nel piano di gioco. Ogni forma deve anche essere in grado di capire se un robot è presente al suo interno.
Nell'applicazione *ShapeInterface* interface viene implementata dalle classi RectangleShape e *CircleShape* ognuna con le proprie caratteristiche.

Definizione dell'Ambiente di Gioco:
L'ambiente di gioco è gestito attraverso la classe *Environment*, che estende l'interfaccia EnvironmentInterface. Questa classe tiene traccia di tutti i robot e le forme presenti nel piano di gioco. Inoltre, gestisce gli spostamenti dei robot e le interazioni tra di essi e le forme, come il calcolo delle distanze tra due robot o la verifica se un robot è all'interno di una forma.

Creazione dei Robot:
La creazione dei robot è gestita attraverso l'interfaccia *RobotsHandlerInterface che fornisce un metodo di default per la creazione dei robot.* Questa interfaccia riceve un array di coordinate fornite dall'utente tramite un'interfaccia grafica e crea robot

corrispondenti a queste coordinate. Il risultato è un hashmap che associa robot a coordinate.

Generazione del Programma:
La generazione del programma avviene tramite una classe *ProgramGenerator.* Questa classe fa uso di un parser, parser che viene implementato sfruttando l'interfaccia *FollowMeParserHandler,* per leggere comandi da un file di configurazione e li trasforma in una lista di *programCommand.*

Generazione dell'Ambiente di Gioco:
La generazione dell'ambiente di gioco è gestita da una classe *EnvironmentGenerator* che implementa l'interfaccia *EnvironmentGeneratorInterface*. Questa classe chiama il parser per creare un array di forme e genera un nuovo ambiente con le forme e i robot corrispondenti.

Rappresentazione di un Punto sul Piano di Gioco:
L'interfaccia *CoordinatesInterface* è utilizzata per rappresentare un punto nel piano di gioco, memorizzando una coppia di valori. Interfaccia implementata dalla Classe *Coordinates* che contiene anche un metodo per aggiornare le coordinate dovo aver effettuato un movimento.

Rappresentazione di un Singolo Comando:
I comandi del programma sono rappresentati attraverso l'interfaccia ProgramCommandInterface. Ogni comando può avere argomenti associati. L'implementazione avviene attraverso la classe ProgramCommand che ha più costruttori utilizzati per differenziare i comandi:

- *public ProgramCommand(String comm, double[] args)* -> chiamato dal parserHandler quando viene trovato un comando move o move random
- *public ProgramCommand(String comm, String label)* -> chiamato quando vengono trovati comandi signal e unsignal
- *public ProgramCommand(String comm, String label, double[] args)* -> chiamato quando viene trovato un comando follow
- *public ProgramCommand(String comm, ExecuteLoopsCommand<R,S> loop)* -> chiamato quando vengono trovati dei comandi che sono dei loop

Esecuzione dei Comandi del Programma:
L'esecuzione dei comandi del programma è gestita da una classe che itera sulla lista di *ProgramCommand.* Compito di questa classe è capire se il comando che si deve eseguire è un loop o meno e quale sia l'effettivo comando da eseguire.
L'esecuzione effettiva dei comandi viene effettuata dalla classe che implementano rispettivamente *ExecuteBasicCommandInterface* per i comandi semplici e *ExecuteLoopsCommandInterface* per l'esecuzione dei loop.

ExecuteBasicCommandInterface è implementata dalla classe ExecuteBasicCommand che contiene l'implementazione dei comandi che possono essere eseguiti da un robot (move, move random, signal, unsignal, continue, follow e stop).

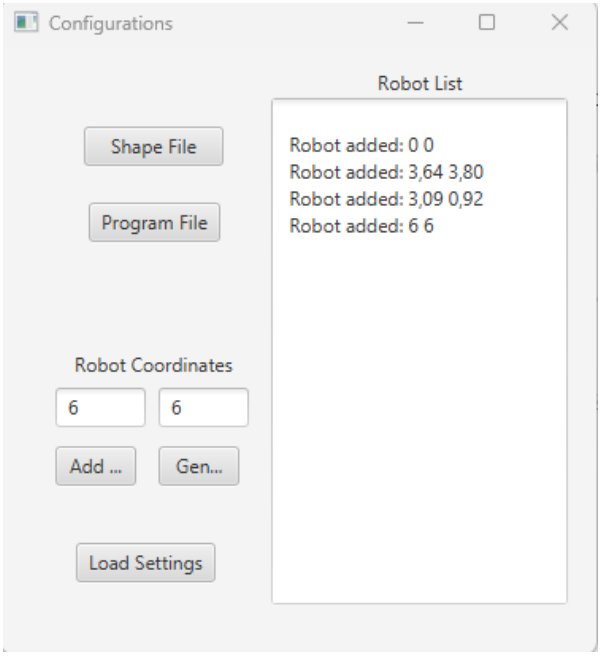
ExecuteLoopCommandInterface è implementata da tre classi:

- *ExecuteDoneCommand*: gestisce l'esecuzione del comando done, riportando l'indice di esecuzione all'inizio del ciclo corrispondente a quel done
- *ExecuteRepeatCommand*: Esegue i comandi di repeat, e doforever (se il numero di ripetizione è settato a -1 allora viene effettuato il doforever) ed anche il comando continue.
- *ExecuteUntilCommand*: gestisce il comando Until, che continua a ripetere i comandi fino a quando non trova una determinata label

Calcolo direzione:
Per calcolare la direzione che deve prendere il robot durante il movimento e la sua velocità. La direzione è rappresentata da un coppia di valori che vengono normalizzati e poi moltiplicati per la velocità.
L'interfaccia che si occupa di questa funzionalità è *RobotDirectionInterface* implementata dalla classe *RobotDirection*

INTERFACCIA GRAFICA

L'interfaccia è composta da due Stage; il primo configurationView si occupa di prendere i dati in input dall'utente. È necessario inserire il file di configurazione per le shape e il programma che deve essere eseguito dai robot. Dopo vengono inseriti i robot o inserendo manualmente le coordinate o generandole casualmente.



Una volta cliccato su Load Settings si aprirà il secondo stage simulationAreaView che inizializza l'environment chiamando i metodi del ModelController e mostrerà a schermo le varie shape e i vari robot.

A sinistra abbiamo l'area di simulazione, a destra un log dei comandi eseguiti dai robot e in basso il tasto per mandare in avanti l'esecuzione del programma con al fianco il numero di istruzioni che devono essere eseguite per ogni esecuzione.

Avviare l'applicazione con i comandi **gradle build** e **gradle run**. Per avviare la simulazione di test caricare i file nell'applicazione i file di configurazione **shapes.txt** e **program.txt** contenuti nella cartella configurations all'interno della cartella dell'applicazione. Aggiungere un **robot alla posizione (-100;0)** ed avviare la simulazione.