Relazione Progetto Programmazione Avanzata

Leonardo Salpini MAT:114525

RESPONSABILITÀ E RELATIVE IMPLEMENTAZIONI

Rappresentazione di un Punto sul Piano di Gioco:
L'interfaccia *Coordinate* è utilizzata per rappresentare un punto nel piano di gioco, memorizzando una coppia di valori. Interfaccia implementata dalla Classe *CartesianCoordinate* che semplicemente restituisce i valori quando vengono richiesti.

Rappresentazione dei Robot:
I robot sono rappresentati come oggetti che implementano l'interfaccia *Robot*. Ogni robot è caratterizzato da: una copia del programma e il relativo indice di esecuzione, una lista di labels che indica le condizione che il robot sta segnalando e da una coppia di valori che stanno ad indicare l'ultimo spostamento effettuato.

Rappresentazione delle Forme:
Le forme sono oggetti che implementano l'interfaccia Shapes. Le implementazioni di questa interfaccia sono *RectangleShape* e *CircleShape*, che sono due record entrambi caratterizzati da un parametro in comune *"conditionLabel"* e da altri parametri riguardanti i rispettivi dettagli (base, altezza, raggio).

Rappresentazione dei Comandi:
L'interfaccia per la rappresentazione di un comando è *Command* che fornisce un solo metodo per ritornare la tipologia del comando.
Tutti i Comandi Base implementano l'interfaccia Command, sia direttamente che indirettamente attraverso un'altra interfaccia RunnableCommand.

- **RunnableCommand**: interfaccia con un solo metodo Run(). È implementata da tutti i comandi che necessitano di un metodo run per eseguire le operazioni sullo stato del robot (coordinate o condition labels).
- **Move**: (Implementa RunnableCommand) il cui obiettivo è quello di calcolare il movimento del robot, e sfruttare i metodi dell'environment per muovere quest'ultimo. Viene anche usato per il movimento randomico, semplicemente viene passata una coppia randomica come coordinate target al costruttore (in questo caso non per forza compresi tra -1 e 1 come il move normale). Da notare che il movimento viene calcolato utilizzando le coordinate relative del robot, quindi sempre (0;0) come partenza.
- **Done**: (Implementa Command) un semplice record che contiene il riferimento al comando di inizio loop.
- **Follow**: (Implementa RunnableCommand) in caso di robot che stanno segnalando una determinata condizione, fa la media delle coordinate dei robot

per poi chiamare il metodo run del comando Move per effettuare il movimento. Nel caso che nessun robot stia segnalando la condizione genera dei numeri randomici. Prima di effettuare il movimento è necessario calcolare il movimento utilizzando le coordinate assolute del robot nel piano. (Es. se il robot si muove dalle coordinate 30;30 alle coordinate 20;20 dovrebbe tornare verso la posizione 0;0. Ma siccome il Move calcola partendo dalla posizione 0;0 allora il robot andrà sempre più lontano dal centro. Per questo viene prima calcolata la differenza tra le coordinate cioè -10;-10 e poi normalizzata per ottenere valori tra 1 e -1 che verranno utilizzati per effettuare il movimento con il comando Move).

- **Continue**: (Implementa RunnableCommand) prende il valore contenuto in "*lastMovementDirection*" del robot e lo somma alle attuali coordinate per X secondi.
- **Stop**: (Implementa RunnableCommand) azzera "*lastMovementDirection*" e "*conditionLabels*" del robot.
- **Signal**: (Estende RunnableCommand) aggiunge una determinata label al robot se quest'ultimo si trova all'interno di una shape con il rispettivo label.
- **Unsignal**: (Estende RunnableCommand) rimuove una lablel dal robot

I comandi Loop che estendono la classe astratta *LoopCommand* che a sua volta implementa l'interfaccia *Command*:

- **LoopCommand**: classe astratta il cui scopo è quello di memorizzare l'indice di inizio e di fine del loop con relativi getters e setters e un metodo astratto "*conditionStatus*" per controllare se il loop deve continuare o terminare.
- **Repeat**: che gestisce sia il repeat N volte che il repeat forever e implementa il metodo astratto "*conditionStatus*" della superclasse.
- **Until**: implementa semplicemente il metodo "*conditionStatus*" della superclasse. Continua a ripetere i comandi finché il robot che sta segnalando una label non esce dalla shape corrispondente.

Rappresentazione Ambiente di Simulazione:
L'ambiente di gioco è rappresentato tramite il record *SimulationEnvironment* che implementa l'interfaccia *Environment*. Il record contiene due mappe per la memorizzazione dei robot e delle shapes con relative coordinate, oltre che le classiche operazioni per aggiungere e togliere robot e shapes fornisce anche diversi metodi utilizzati poi dai comandi per eseguire le loro funzioni: Ad esempio metodo per controllare se un determinato robot è all'interno di una shape, calcolare la distanza tra due coordinate …

Parsing del Programma:
Il parsing del programma è gestito dal *FollowMeParser* che legge i comandi dal file e chiama i metodi della classe *ProgramParserHandler* che implementa *FollowMeParserHandler.* Ogni metodo di questa classe viene chiamato dal FollowMeParse ogni volta che incontra il rispettivo comando. Ogni metodo crea il suo

rispettivo comando e lo aggiunge ad una lista, lista che viene poi aggiunta al *RobotSimulator*.

Generazione delle Shapes:
La generazione delle shapes è gestita dalla classe *DefaultShapeGenerato* che implementa l'interfaccia *ShapeGenerator*. Questa classe sfrutta il metodo parseEnvironment di *FollowMeParser* che restituisce una lista di ShapeData. Presa la lista vengono presi tutti gli elementi uno a uno e viene creata la rispettiva shape e inseriti in una collection.

Clonazione di un comando:
Per clonare i comandi viene utilizzata l'interfaccia *CommandClonerFactory* che fornisce un solo metodo che prende un singolo parametro di tipo *Command* e restituisce lo stesso comando clonato. Per ogni comando è stata creata la rispettiva clone factory che implementa le relative logiche di clonazione.

Clonazione del Programma:
La classe *ProgramCloner* si occupa di clonare un intero programma utilizzando le varie clone factory relative ad ogni tipo di comando

Tempo di Simulazione:
Classe *SimulationTimer* con attributi e metodi statici per tenere traccia del tempo della simulazione.

Esecuzione della Simulazione:
La gestione della simulazione è affidata alla classe *RobotSimulator*, la quale implementa l'interfaccia Simulator ed estende la classe *SimulationTimer*. L'esecuzione della simulazione avviene mediante l'invocazione del metodo *simulate()*, che richiede tre parametri:

- **delta_t**: il tempo che impiega un comando per essere eseguito;
- **execution_time**: il tempo massimo di esecuzione
- **numberOfCommandsForExecution**: il numero di comandi che deve essere eseguito per ogni passo della simulazione.

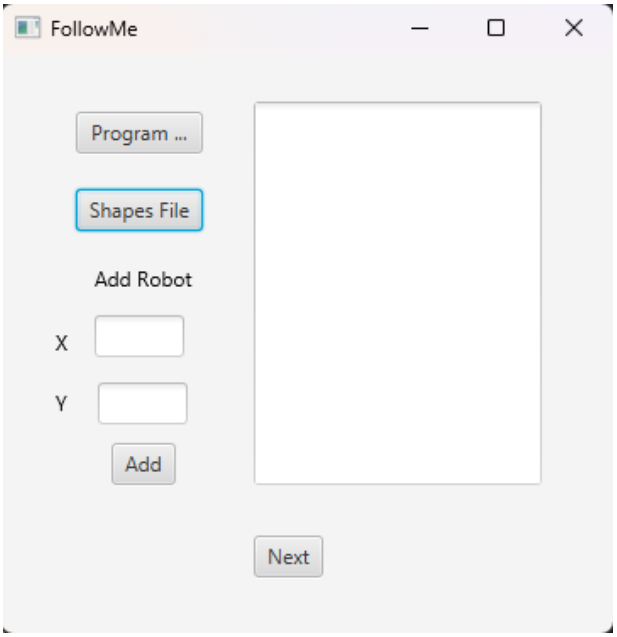
Il numero massimo di comandi eseguibili calcolato facendo execution_time/delta_t e approssimato all'intero più vicino. Se il programma deve ancora terminare la sua esecuzione, ma è stato raggiunto il tempo massimo di esecuzione verrà immediatamente interrotto.

Ogni volta che viene chiamato il metodo *simulate()* vengono eseguiti uno o più comandi per ogni robot.

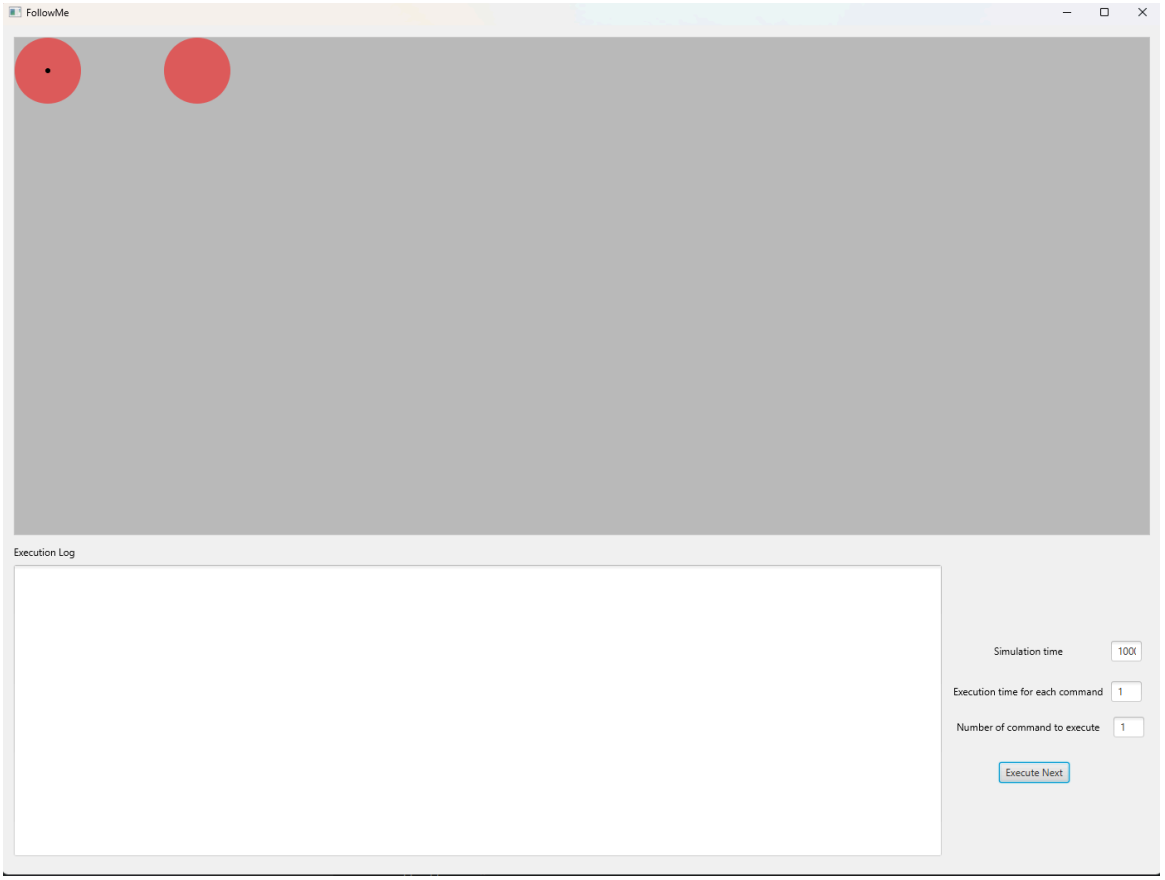
Interazione con modulo App:
Le interazioni con il modulo app vengono gestite dalla classe *ModelController*, che riceve i dati inseriti dagli utenti nel modulo App, fa le dovute operazioni ed esegue i metodi.

INTERFACCIA GRAFICA ED ESECUZIONE DEGLI ESEMPI

L'interfaccia è composta da due Stage; il primo 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 e successivamente devono essere inseriti i robot.



Cliccando su Next si aprirà il secondo stage che inizializza l'environment chiamando i metodi dell' ApiController e mostrerà a schermo le varie shape e i vari robot. In alto abbiamo l'area della simulazione. in basso, da una parte, abbiamo le varie configurazioni e il tasto per l'esecuzione e dall'altra invece il log dei comandi.
Nel log vengono mostrati i soli comandi eseguiti durante quello step di esecuzione, se si vuole vedere il log completo si trova nella cartella "configuration_files".



Avviare l'applicazione con i comandi **gradle build** e **gradle run** tramite cmd aperto all'interno della cartella. Per avviare la simulazione di test caricare i file nell'applicazione i file di configurazione presenti all'interno della cartella "configuration_files".

- Per l'esecuzione del file per i comandi base inserire due robot alla posizione (0;0) e (0;90).
- Per l'esecuzione dei repeat e do forever selezionare i rispettivi file e aggiungere un singolo robot alla posizione (0;0).
- Per l'esecuzione del programma contenente tutti i loop, aggiungere due robot alla posizione (0;0) e (0;-100).