

Robotica

**Guardie e Ladri 2.0**

**Prof. Antonio Chella**

**Ing. Francesco Lanza**

**A cura di:**

**Davide Lavalle**

**Cedric Ferdico**

**Luigi Gatto**

**Indice**

[1. Introduzione 4](#_Toc68616114)

[1.1. Generalità del progetto 4](#_Toc68616115)

[1.2. Scopo del gioco 4](#_Toc68616116)

[2. Caratteristiche Hardware 5](#_Toc68616117)

[3. Ambiente di sviluppo 8](#_Toc68616118)

[3.1. Java 8](#_Toc68616119)

[3.2. Eclipse 8](#_Toc68616120)

[3.3. Webots 8](#_Toc68616121)

[4. Struttura del progetto 9](#_Toc68616122)

[4.1. Suddivisione in package 10](#_Toc68616123)

[5. Il protocollo di rete 11](#_Toc68616124)

[5.1. La classe “AsynchronousSocketChannel”[1] 11](#_Toc68616125)

[5.2. I pacchetti 12](#_Toc68616126)

[5.2.1. CTS\_PEER\_INFO 12](#_Toc68616127)

[5.2.2. STC\_SEND\_MAP 12](#_Toc68616128)

[5.2.3. CTS\_WORLD\_READY 12](#_Toc68616129)

[5.2.4. CTS\_OBSTACLE\_IN\_MAP 12](#_Toc68616130)

[5.2.5. CTS\_GOING\_TO 13](#_Toc68616131)

[5.2.6. CTS\_NEW\_GUARDIA\_POS 13](#_Toc68616132)

[5.2.7. CTS\_GOAL\_CHANGED 13](#_Toc68616133)

[5.2.8. CTS\_LADRO\_FOUND 13](#_Toc68616134)

[5.2.9. CTS\_LADRO\_HIDDEN 14](#_Toc68616135)

[5.2.10. STC\_START\_GUARDIE 14](#_Toc68616136)

[6. Proto 15](#_Toc68616137)

[6.1. Guardia 15](#_Toc68616138)

[6.2. Ladro 15](#_Toc68616139)

[6.3. WoodenBox 15](#_Toc68616140)

[7. Algoritmi ed implementazioni 16](#_Toc68616141)

[7.1. Creazione della Mappa 16](#_Toc68616142)

[7.2. Ricerca del punto 16](#_Toc68616143)

[7.3. Ladri 16](#_Toc68616144)

[7.4. Guardie 16](#_Toc68616145)

[7.5. Ricerca di un path 17](#_Toc68616146)

[7.6. Navigazione 17](#_Toc68616147)

[7.7. Correzione del Moto 17](#_Toc68616148)

[7.8. Esplorazione (Guardie) 18](#_Toc68616149)

[7.9. Collision Avoidance (Guardie) 19](#_Toc68616150)

[8. Ambiente di simulazione 20](#_Toc68616151)

[9. Possibili implementazioni future 21](#_Toc68616152)

[10. Guida all'installazione 22](#_Toc68616153)

[11. Conclusioni 26](#_Toc68616154)

[12. Bibliografia 26](#_Toc68616155)

# Introduzione

## Generalità del progetto

Il progetto, condotto dal team GLF, è stato quello realizzare una simulazione del gioco “Guardia e Ladri“ utilizzando il simulatore “Webots”.

Abbiamo analizzato e studiato approfonditamente il suddetto ambiente comprendendo le meccaniche e le strutture per riuscire a sfruttare la maggior parte delle funzionalità offerte dal software.

Tra queste, vi è la possibilità di utilizzare dei controllori esterni per la gestione di uno o più robot, l’uso di telecamere con incluso il riconoscimento di oggetti e il controllo dell’intera simulazione (come per esempio lo spawn di oggetti e/o robot), da parte di un robot dotato di particolari “permessi” definito come “**supervisor**”.

Si è deciso di sviluppare, inoltre, un background di intelligenza artificiale. In particolare, si è utilizzato l’algoritmo A\* che, insieme alle proprietà della Quattro e della Otto Adiacenza, si è implementata una soddisfacente tecnica di esplorazione ed un buon algoritmo di creazione pseudocasuale della mappa di gioco.

Inoltre, tramite un protocollo applicativo (su TCP) da noi definito, abbiamo fatto in modo che l’esplorazione della mappa, da parte delle guardie per la ricerca dei ladri, avvenga in un ambiente multi agente collaborativo. Possiamo quindi dire di aver effettuato un primo approccio alla “swarm robotics “ per la logica di collaborazione dei robot.

Infine, abbiamo scelto di introdurre musiche, suoni e personalizzazioni dei robot per introdurre quel senso di “videogioco“ alla quale ci siamo ispirati.

## Scopo del gioco

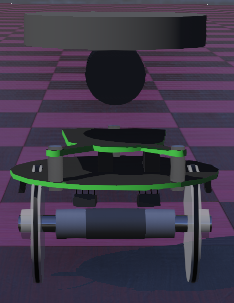
Una volta generata la mappa dal server, in maniera pseudocasuale e basandosi sulle caratteristiche specificate nel file “Config.txt”, quest’ultimo la invia al robot “**supervisor**” il quale effettua lo spawn degli ostacoli nei punti specificati nella mappa di gioco.

Anche i ladri ricevono la mappa completa dal server i quali cercheranno una posizione “sicura” calcolata tramite l’algoritmo di ricerca spiegato nei paragrafi successivi.

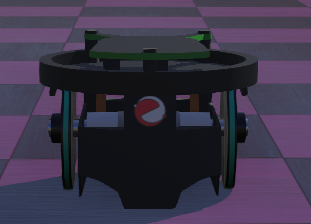
Una volta che il team dei ladri si nasconde, le guardie, le quali conoscono solamente la dimensione della mappa ed il bordo, inizieranno ad esplorare la mappa alla ricerca dei ladri. Durante l’esplorazione, le varie guardie comunicheranno continuamente tra di loro segnalando la propria posizione, il luogo in cui stanno andando e gli ostacoli trovati in modo da ottimizzare e velocizzare la ricerca. Inoltre, non appena una guardia riuscirà a riconoscere, tramite l’ausilio della Camera integrata, un ladro, l’evento verrà segnalato al resto delle guardie.  
Il “gioco” termina quando tutti i ladri sono stati trovati, le guardie si fermeranno e daranno in output il risultato della loro esplorazione (con le posizioni degli ostacoli e dei ladri trovati).

**N.B. Per motivi puramente dimostrativi non è stato introdotto un timer per la vittoria dei ladri in modo da dimostrare la capacità di esplorazione dell’intera mappa correttamente.**

# Caratteristiche Hardware



**Foto guardia**



**Foto ladro**

Entrambi i robot utilizzati sono la versione 1 dei GCTronic’ e-puck opportunamente modificata per ed ottimizzata per lo scopo del progetto.

Sono stati rimossi o disabilitati elementi presenti nell’originale e-puck che portava ad un oneroso costo computazionale (Gyro,Accelerometer) o un elevato costo in termini grafici (Light Sensor, Texture, etc).

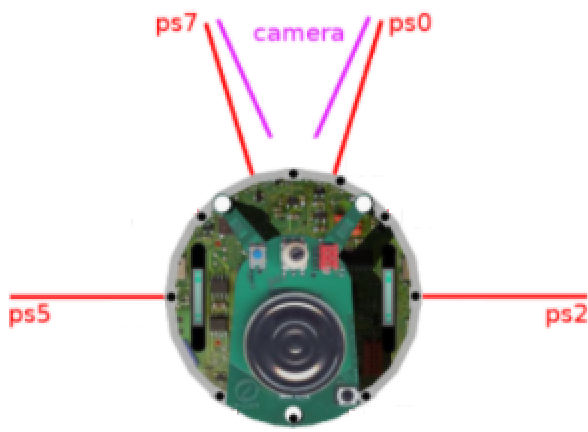
Di seguito riportiamo i dati tecnici dell’e-puck generico (sia guardia che ladro):

Dati Tecnici

|  |  |
| --- | --- |
| Characteristics | Values |
| Diameter | 71 mm |
| Height | 50 mm |
| Wheel radius | 20.5 mm |
| Axle Length | 52 mm |
| Weight | 0.16 kg |
| Max. forward/backward speed | 0.25 m/s |

|  |  |
| --- | --- |
| Device | Name |
| Motors | 'left wheel motor' and 'right wheel motor' |
| Position sensors | 'left wheel sensor' and 'right wheel sensor' |
| Proximity sensors | ‘ps0’, ‘ps2’, ‘ps5’, ‘ps7’ |
| Light sensors ( not used ) | 'ls0' to 'ls7' |
| LEDs | ‘'led\_1’, ‘led\_2’ |
| Camera | 'camera' |
| Accelerometer ( not used ) | 'accelerometer' |
| Gyro ( not used ) | 'gyro' |
| Ground sensors ( not used ) | 'gs0', 'gs1' and 'gs2' |
| Speaker | 'speaker' |

**La risoluzione della camera è stata migliorata fino a 512x206( dai 52x39 di base ).**

**Robot e-puck Generico**

**L’unica differenza tra le guardie ed i ladri è la videocamera utilizzata dalle guardie per identificare i ladri ed evidenti cambiamenti puramente estetici come visto in foto.**

# 

# 

# Ambiente di sviluppo

## Java

Webots, supporta diversi linguaggi di programmazione per la realizzazione dei controllori per i robot, tra questi i più noti sono C++, Python e Java. Dopo che il gruppo si è confrontato valutando pro e contro dei vari linguaggi conosciuti, dato che l’intero il gruppo ha seguito il corso di Programmazione del Prof. La Cascia in cui si è studiato e sviluppato un progetto con pattern MVC con java e data l’esperienza con questo linguaggio per progetti personali, si è deciso di utilizzare quest’ultimo (Java) come linguaggio di programmazione ritenuto altresì ideale per la tipologia di progetto.

## Eclipse

Eclipse è tra i più famosi IDE Java.

Dato che si è deciso l’utilizzo dei controllori esterni, e dato che diversi controllori e quindi, diversi programmi, utilizzano più volte le stesse classi, si è reso necessario l’uso di un IDE in modo da semplificare il più possibile la gestione dei Packages all’interno del progetto. Dalle esperienze passate, l’intero team ha concordato l’utilizzo di Eclipse.

## Webots

Webots è un applicazione open source e multi piattaforma usata per simulare robot. Webots fornisce un ambiente di sviluppo complete per modellare, programmare e simulare dei robot. È stato sviluppato per usi professionali ed è largamente usato in campi aziendali, scolastici e di ricerca. Cyberbotics Ltd. sviluppa Webots come suo prodotto principale dal 1998.

Con un’ampia documentazione, la sua community attiva, tutte le varie tipologie di linguaggi adoperati ed il suo continuo miglioramento, Webots risulta essere uno dei simulatori migliori sia in ambito educativo che lavorativo.

# 

# Struttura del progetto

Eseguendo il main contenuto nella classe “Main” all’interno del package “General”, verrà eseguito un setup iniziale tramite le informazioni contenute all’interno del file “Config.txt”.

Una volta recuperati i parametri presenti nel suddetto file, verrà avviato un Thread che si occuperà di gestire la parte relativa al server, il quale creerà un nuovo Thread per ogni client (un controllore di un robot) che si connetterà. Infine, avvierà il controllore del robot supervisor.

Il server, in maniera pseudocasuale (seguendo le regole e l’algoritmo spiegati successivamente), genera la mappa con le specifiche presenti nel file sopracitato, per poi mettersi in ascolto su una porta TCP specificata nel file “Config.txt”.

Una volta che il controllore **supervisor** sarà correttamente collegato con Webots, il **server** invierà al **supervisor** la mappa e quest’ultimo si occuperà di generare tutti gli ostacoli ed i robot di tipo guardia e di tipo ladro (i cui numeri sono specificati nel file “Config.txt”).

Quando il **supervisor** ha terminato la fase di “spawn”, verrà inviato un pacchetto al **server**, il quale avvierà tutti i controllori necessari in modo che possa avviarsi la simulazione.

## 

## Suddivisione in package

-Controllers

| - GuardiaController

| - LadroController

| - SupervisorController

-General

| - AStarSearcher

| - ControllerExecutor

| - Main

| - SharedVariables

| - StreamPrinter

-Map

| - Mappa

| - MappaInterna

| - Point

| - StartPositions

-Network

| - Packets

| | - ClientToServer

| | | - CTS\_GOAL\_CHANGED

| | | - CTS\_GOING\_TO

| | | - CTS\_LADRO\_FOUND

| | | - CTS\_LADRO\_HIDDEN

| | | - CTS\_NEW\_GUARDIA\_POS

| | | - CTS\_OBSTACLE\_IN\_MAP

| | | - CTS\_PEER\_INFO

| | | - CTS\_WORLD\_READY

| | - ServerToClient

| | | - STC\_SEND\_MAP

| | | - STC\_START\_GUARDIE

| | - Packet

| - Client

| - ClientConnectionHandler

| - Server

| - ServerConnectionHandler

-Robot

| - Sensors

| | - FrontalSensors

| | - LateralSensors

| - GenericRobot

| - GuardiaRobot

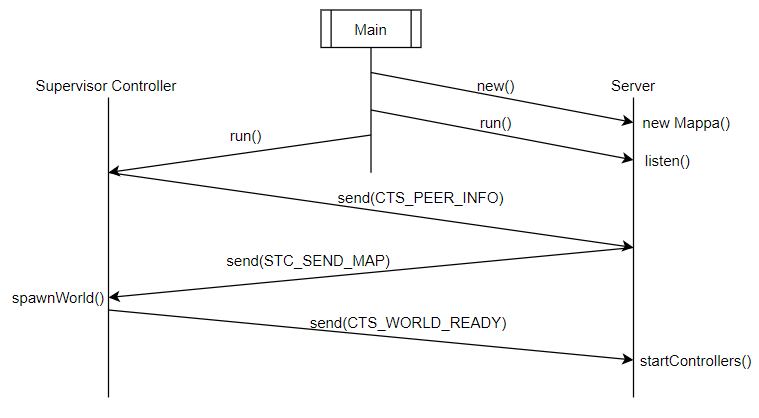
| - LadroRobot

| - Motors

| - SireneThread

| - SupervisorRobot

# Il protocollo di rete



**Fase di avvio**

## La classe “AsynchronousSocketChannel”[1]

Questa classe implementa un canale asincrono per connessioni di tipo stream-oriented. E’ possibile creare un AsynchronousSocketChannel nel seguente modo:

Un nuovo **AsynchronousSocketChannel** viene creato invocando uno dei metodi “open” definiti all’interno della classe. Un nuovo canale viene così aperto ma non ancora connesso. Un **AsynchronousSocketChannel** passa allo stato “connected” quando viene effettuata una connessione ad una socket tramite una funzione definita dalla classe [**AsynchronousServerSocketChannel**](https://docs.oracle.com/javase/7/docs/api/java/nio/channels/AsynchronousServerSocketChannel.html). Un canale appena creato, quindi, si connette invocando il metodo “connect”; una volta connesso, un canale rimane attivo fino alla sua chiusura. Un tentativo di operazione di tipo I/O su un canale non connesso causerà il lancio dell’eccezione“[**NotYetConnectedException**](https://docs.oracle.com/javase/7/docs/api/java/nio/channels/NotYetConnectedException.html)”.   
  
I canali di questo tipo sono sicuri per essere usati su più thread concorrenti. Essi supportano la lettura e la scrittura concorrente facendo in modo che avvenga una sola operazione di lettura/scrittura per volta. Se un Thread inizia un’operazione di lettura prima che una lettura precedente sia stata completata, viene lanciata l’eccezione “[**ReadPendingException**](https://docs.oracle.com/javase/7/docs/api/java/nio/channels/ReadPendingException.html)”.

Allo stesso modo, un tentativo di scrittura prima che la precedente sia stata completata, causa il lancio dell’eccezione “[**WritePendingException**](https://docs.oracle.com/javase/7/docs/api/java/nio/channels/WritePendingException.html)”.

Per questi ultimi due motivi, abbiamo usato il metodo “**get**” bloccante della classe “**Future**” il quale aspetta il completamento dell’operazione di lettura/scritta dal/sul canale.

## I pacchetti

## CTS\_PEER\_INFO

Questo pacchetto viene inviato quando un nuovo client si connette al server. Il pacchetto contiene il tipo di **client** (Guardia, Ladro o Supervisor) che si è appena connesso. Il **server**, quando si connette una nuova guardia o un nuovo ladro, aggiunge questo client nelle liste corrispondenti affinchè possa inviargli successivamente i pacchetti necessari. Quando si connette un **client** corrispondente ad un controllore di tipo “**supervisor**”, il server invia un pacchetto “STC\_SEND\_MAP”.

## STC\_SEND\_MAP

Questo pacchetto contiene un oggetto della classe Mappa. Viene inviato quando il **supervisor** si connette al server e ogni qual volta si connette un nuovo **ladro** o una nuova **guardia**. Qui possiamo distinguere due casi:

* Quando il **ladro** riceve questo pacchetto, Verrà inizializzata una lista contenente tutti i punti ritenuti “validi” per nascondersi.
* La guardia, anche se riceve tutta la mappa contenente anche le posizioni degli ostacoli, estrarrà solamente le informazioni necessarie per creare un nuovo oggetto della classe Mappa con le dimensioni uguali a quella della mappa ricevuta. La guardia, inizialmente, conoscerà anche tutte le posizioni degli ostacoli che costituiscono il bordo della mappa.

## CTS\_WORLD\_READY

Questo pacchetto viene inviato dal robot supervisor quando ha terminato lo spawn dei robot e degli ostacoli e funge da “segnale di start”.

## CTS\_OBSTACLE\_IN\_MAP

Questo pacchetto contiene un oggetto della classe “Point” il quale rappresenta il punto della mappa in cui è presente un ostacolo. Questo pacchetto è inviato al server ogni qual volta che una guardia, durante l’esplorazione, incontra un nuovo ostacolo. Quando il server riceve questo pacchetto non fa altro che inoltrarlo a tutte le altre guardie le quali aggiorneranno la propria mappa.  
L’idea di questo pacchetto è nata dall’approccio alla “swarm robotics” ed è stato utilizzato per rendere collaborativi il team delle guardie ed aumentare la complessità e rendere più intelligente l’intero progetto.

## CTS\_GOING\_TO

Questo pacchetto contiene un oggetto della classe “Point” il quale rappresenta il punto della mappa in cui sta andando il client. Quando una guardia invia questo pacchetto al server, quest’ultimo lo inoltrerà alle altre guardie le quali rimuoveranno il punto in questione dalla propria lista dei punti validi ancora da esplorare. In questo modo più guardie non avranno mai lo stesso punto come destinazione.

Invece, quando un ladro invia questo pacchetto al server, quest’ultimo lo inoltra agli altri ladri i quali rimuoveranno il punto dalla propria lista dei punti buoni per nascondersi. In questo modo più ladri non cercheranno di nascondersi nello stesso punto.

## CTS\_NEW\_GUARDIA\_POS

Questo pacchetto contiene due oggetti della classe “Point”, uno rappresenta la vecchia posizione della guardia, l’altro la nuova posizione. Viene inviato da ogni guardia che ha modificato la propria posizione in modo che le guardie possano tenere traccia delle posizioni reciproche.

## CTS\_GOAL\_CHANGED

Questo pacchetto contiene due oggetti della classe “Point”. Viene inviato da una guardia quando modifica il proprio goal. Questo evento si verifica quando una guardia incontra un’altra guardia di fronte. In questo caso, la guardia cercherà un goal nella direzione opposta e per questo motivo è necessario che la suddetta guardia segnali alle altre che il proprio goal è cambiato. Di conseguenza, quando le altre guardie ricevono questo pacchetto, inseriranno il vecchio goal nella propria lista dei punti da esplorare e rimuoveranno il nuovo goal.

## CTS\_LADRO\_FOUND

Questo pacchetto contiene un oggetto della classe “Point”. Viene inviato quando una guardia ha trovato un ladro. Quando le altre guardie ricevono questo pacchetto, quest’ultime, se non hanno già considerato il ladro in quella posizione, aggiorneranno la propria mappa e incrementeranno un contatore il quale, quando raggiungerà un valore pari al numero di ladri presenti nella simulazione, farà terminare l’esplorazione delle mappa dando così la vittoria alle guardie.

## CTS\_LADRO\_HIDDEN

Questo pacchetto viene inviato da un ladro quando ha raggiunto il punto in cui starà nascosto. Quando il server riceve questo pacchetto, incrementa un contatore che quando raggiungerà il numero di ladri presenti nella simulazione invierà alle guardie il pacchetto “STC\_START\_GUARDIE” dando così ufficialmente via al gioco.

## STC\_START\_GUARDIE

Dopo che tutti i ladri si sono nascosti, le guardie riceveranno questo pacchetto che farà avviare un timer dipendente dall’id associato alla guardia. Una volta terminato il timer, la guardia inizierà ad esplorare la mappa e cercare i ladri.

# 

# Proto

## Guardia

Nel file proto relativo alla guardia sono stati modificati i parametri per ottimizzare la scena ed inserite features grafiche quali texture, loghi, una telecamera con performance migliori rispetto a quella base (una risoluzione di 512x206 contro i 52x48 nativi, l’antialiasing, è stata modificata l’apertura focale), speaker, dei nuovi led che rappresentano le “sirene” della guardia, ed il campo “recognitionColors” per essere individuato dalla telecamera.

## Ladro

Nel file proto relativo al ladro sono stati modificati i parametri per ottimizzare la scena ed inserite features grafiche come la “gonna” inferiore, delle texture nere, uno speaker ed il campo “recognitionColors” per essere individuato dalla telecamera della guardia.

## WoodenBox

Il file proto relativo alle WoodenBox è simile a quello di base di Webots. È stato modificato per diminuire il carico dovuto alla complessità della geometria dell’oggetto in questione. In particolare, si è modificata la dimensione in “0.995, 0.995, 0.995” per evitare le collisioni con gli angoli con gli altri ostacoli presenti e, di conseguenza, è stato modificato il campo “geometry box”. Infine, il peso è stato modificato con un valore idoneo alla scena (1 kg).

# Algoritmi ed implementazioni

## Creazione della Mappa

Per la creazione della mappa si è voluto sviluppare un algoritmo di generazione pseudocasuale.

Fondamentalmente si sceglie un pattern casuale (3 cubi in ogni posizione cardinale, etc..), per poi scegliere casualmente, da un pool di possibili posizioni (in modo da scartare i punti non validi), un punto di spawn “di partenza” del pattern scelto in precedenza.

Il valore di difficoltà presente nel file “Config.txt” determina il numero di cubi che verranno posizionati all’interno della mappa.

Dopo aver terminato la generazione della mappa, il server la invierà ai vari controllori che si connetteranno. Il “supervisor” sarà il responsabile della spawn degli ostacoli all’interno della simulazione.

## Ricerca del punto

“La ricerca del punto” riguarda la scelta del goal, ovvero del punto nella simulazione in cui il robot, indipendentemente da guardia o ladro che sia, dovrà dirigersi. Dati i diversi scopi delle due tipologie di robot presenti nel progetto, il modo in cui viene scelto il punto in questione presenta delle sostanziali differenze che spiegheremo nei paragrafi successivi:

## Ladri

I ladri devono scegliere una posizione ideale dove nascondersi. Per questo motivo, conoscendo la mappa e di conseguenza la disposizione degli ostacoli, eseguono una ricerca su dei punti ritenuti “validi” dal team di sviluppo. Un punto viene considerato “valido” se questo è circondato da tre ostacoli in modo tale da rendere il ladro visibile dal minor numero di direzioni possibili.

Una volta trovati tutti i punti “validi” presenti sulla mappa, ogni robot ne sceglierà uno in maniera casuale e inizierà la ricerca di un path verso di esso per poi raggiungerlo. Se il path esiste realmente, allora il punto in considerazione verrà notificato agli altri ladri attraverso il server, i quali lo escluderanno dal proprio set dei possibili punti “validi”, altrimenti, verrà scelto un nuovo punto fino a trovarne uno per cui esista un path.

## Guardie

Per quanto riguarda le guardie la situazione è ben diversa: loro, non conoscendo la mappa, avranno il compito di esplorarla.

Per fare ciò, all’avvio, metteranno ogni punto esplorabile della mappa all’interno di una struttura chiamata “openSet” che contiene tutti i punti ancora da esplorare.

Il passo successivo è quello di selezionare un punto casuale dall’openSet, cercare un path verso di esso ed iniziare a muoversi secondo la logica di esplorazione attuando anche il principio di collision avoidance nei confronti di ostacoli ed altre guardie. Ogni punto scelto come destinazione viene notificato alle altre guardie che lo toglieranno dal proprio openSet.

## Ricerca di un path

Per la ricerca di un path è stato utilizzato l’algoritmo informato A\*. Questo è un algoritmo di ricerca su grafi che individua il percorso a costo minimo da un nodo di partenza ad uno di destinazione. Nel nostro caso i nodi del grafo sono i punti della mappa. Per far funzionare bene l’algoritmo è necessario definire una funzione di costo euristica che classifica ogni nodo in base alla stima del costo per raggiungere il goal partendo da quel nodo. Nel nostro caso l’euristica presa in considerazione è stata la distanza euclidea definita come:

Questa risulta essere un’euristica ammissibile e consistente. Ogni nodo è definito da due parametri di costo, uno riguarda il costo per raggiungere il nodo stesso partendo dal nodo iniziale, l’altro riguarda il costo stimato dalla funzione euristica per raggiungere il goal partendo dal nodo in considerazione. Per ogni nodo, quindi, la somma dei due parametri rappresenta il costo stimato per raggiungere il goal passando per esso. Altre strutture fondamentali per l’algoritmo sono l’openSet e il closedSet: il primo comprende i punti che sono raggiungibili e non ancora espansi, il secondo contiene i punti già espansi. L’algoritmo comincia inserendo il punto di start nell’openSet, lo espande e lo sposta nel closedSet mettendo tutti i punti raggiungibili dal punto di start nell’openSet (i 4 vicini). L’algoritmo continua via via esplorando i punti che hanno il costo stimato più basso per arrivare al goal. In questo modo, quando troverà una soluzione sarà sicuramente quella a costo minimo e restituirà la sequenza di punti in successione per raggiungere il goal. Se l’algoritmo espande tutti i punti possibili (dimensione openSet = 0) senza mai riuscire ad arrivare al goal, termina restituendo una lista non inizializzata.

## Navigazione

Come detto nel paragrafo precedente, il path calcolato tramite l’algoritmo A\* restituisce una lista di punti della mappa da seguire per raggiungere il goal prefissato. Prima di iniziare il vero e proprio movimento però, il path deve subire una “traduzione”. Questo perché i movimenti base codificati per il robot sono “gira a destra” (90°), “gira a sinistra” (90°) e “vai avanti” (di un passo). Quindi in funzione della posizione attuale il “prossimo punto” viene tradotto in una direzione (NORD, SUD, EST, OVEST). A sua volta, dipendentemente dalla direzione in cui è girato il robot, la prossima direzione viene tradotta in rotazione e movimento (es: direzione attuale: NORD, prossima direzione: EST, questo viene tradotto in una rotazione verso destra e un passo avanti).

## Correzione del Moto

Venendo a fare i conti con il vero e proprio moto dei robot, e con l’imperfezione dei sensori, sono state riscontrate delle leggere imprecisioni nell’attuazione del moto, piccole imperfezioni che accumulandosi durante il movimento hanno portato a degli errori non indifferenti. Per limitare questo fenomeno è stata necessaria l’implementazione di algoritmi e metodi (basilari) per la correzione del moto durante l’attuazione dello stesso sfruttando dei punti di riferimento. Innanzitutto bisogna distinguere i due tipi di movimenti di base dei robot:

* Movimento rotazionale: implementato tramite odometria, facendo riferimento alle letture dei sensori riguardo l’angolo di rotazione della ruota.
* Movimento rettilineo: implementando sperimentalmente un metodo basato sul tempo impiegato dal robot ad effettuare uno spostamento di una cella sulla matrice rappresentante la mappa.

Sono state necessarie delle implementazioni per la correzione di entrambi i tipi di movimento sfruttando gli unici punti di riferimento disponibili che sono gli ostacoli della mappa. La correzione del moto fa unicamente uso dei sensori frontali (ps0 e ps7) per la correzione del movimento angolare, mentre utilizza i sensori laterali (ps2 e ps5) per la correzione del movimento rettilineo.

Per quanto riguarda la correzione del movimento rotazionale, viene applicata quando un robot si avvicina frontalmente ad un ostacolo. Il robot esegue la lettura dei sensori e se la loro differenza supera una certa soglia significa che evidentemente non è in linea con l’ostacolo. Tale differenza, indica quanto il robot sia fuori asse rispetto all’ostacolo frontale, quindi, in base al segno di tale differenza è possibile capire se il robot debba girare verso destra o verso sinistra. Per correggerlo quindi si effettua una rotazione in senso opposto all’errore fino a quando la differenza tra le letture dei due sensori non risulti essere minore della soglia scelta.

Per quanto riguarda la correzione del movimento rettilineo, viene applicata quando il robot cambia direzione per poi eseguire un moto rettilineo. Durante il moto il robot esegue una continua lettura dai sensori laterali e se uno di questi supera una certa soglia significa che il robot è decentrato rispetto alla posizione attuale, quindi al prossimo moto rettilineo dovrà essere applicata la correzione. Per rendere chiaro il funzionamento si pensi ad un moto rettilineo del robot in cui viene notato un ostacolo troppo vicino al sensore sinistro; il tipo di correzione dipenderà dal prossimo cambio di direzione: se il cambio di direzione sarà verso sinistra allora il prossimo moto rettilineo dovrà essere meno lungo, viceversa, dovrà essere più lungo.

## Esplorazione (Guardie)

La logica di esplorazione prevede il movimento da un punto di start verso un goal seguendo un path. Una volta calcolato il path, il robot comincia il suo percorso di navigazione, togliendo dal proprio openSet ogni posizione attraversata dal robot (in quanto se un punto è attraversato è già esplorato) e segnalandolo alle altre guardie che a loro volta lo rimuoveranno dal proprio openSet. Se durante il tragitto rileva la presenza di un ostacolo che impedisce il proseguimento lungo il path, aggiornerà la mappa con la posizione dell’ostacolo appena rilevato, lo comunicherà alle altre guardie e cercherà un nuovo path verso la stessa destinazione. Le conclusioni possibili sono due:

1. La guardia riesce ad arrivare alla destinazione prefissata: in questo caso selezionerà un nuovo punto da esplorare dall’openSet e ricomincerà l’esplorazione da capo.
2. La guardia non riesce a raggiungere la destinazione prefissata: in questo caso significa che con i successivi aggiornamenti della mappa non è più riuscita a trovare un path verso la destinazione, quindi il punto è irraggiungibile. Se questo si verifica, toglierà il punto in questione dal proprio openSet, lo segnerà come un ostacolo e lo notificherà alle altre guardie, che lo segneranno a loro volta come ostacolo e lo rimuoveranno dal loro openSet.

## Collision Avoidance (Guardie)

Per far si che le guardie siano capaci di esplorare la mappa senza scontrarsi tra loro è stato implementato un meccanismo di collision avoidance. Ogni guardia è costantemente aggiornata sulle posizioni attuali delle altre e prima di effettuare un passo in avanti, verifica se qualche altra guardia si trova in una cella vicina secondo lo schema seguente:

|  |  |  |
| --- | --- | --- |
|  | X |  |
| X | X | X |
|  | Guardia |  |

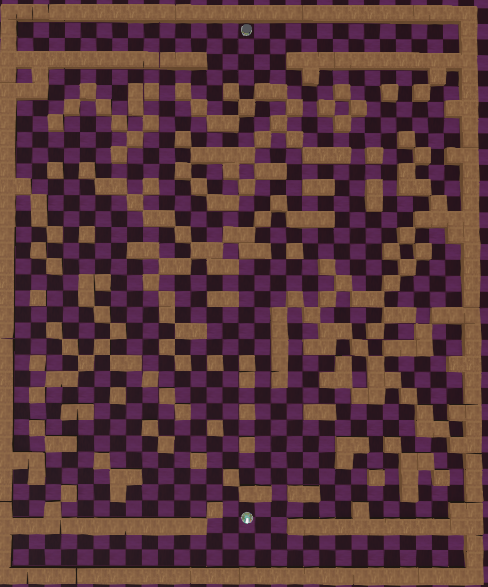
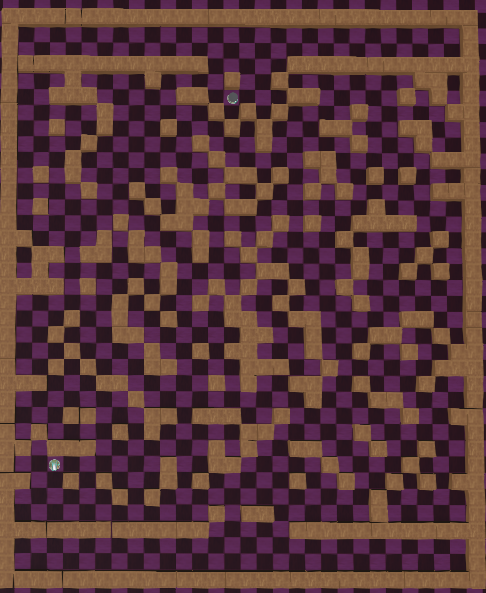
Se si verifica questo evento, entrambe le guardie si fermano e cercano un nuovo goal in una posizione tale che si trovi nella parte di mappa alle loro spalle. Se non viene trovato all’interno dell’openSet, viene scelto un punto già esplorato, quindi, le guardie cercheranno di raggiungerlo.

## 

# Ambiente di simulazione

L’ambiente, creato attraverso l’algoritmo spiegato nel paragrafo 8.1, risulta essere composto da ostacoli posizionati pseudocasualmente all’interno di una matrice quadrata di dimensione specificata nel file di configurazione collegata con due spawn di dimensione anch’esse definite all’interno del file.

Di seguito due prove di spawn con difficoltà: “facile” ovvero 100 blocchi.



**A nord è presente l’area di spawn dei robot di tipo guardia, mentre a sud è presente l’area di spawn dei robot di tipo ladro. Possiamo notare come sia notevole la differenza tra le due mappe con solo due casi consecutivi.**

# 

# Possibili implementazioni future

Sono state pensate molteplici funzionalità che il team ha intenzione di implementare.  
1) Si punterà ad ottimizzare la scena diminuendo il TimeStep da 16, in cui è attualmente, a 4 in modo da rendere più precisa la simulazione. Ciò però gioca un importante ruolo in fase di requisiti minimi del sistema che aumenta vertiginosamente e, quindi, necessita di un’ulteriore ottimizzazione nel codice che richiede uno studio più approfondito e tempo di analisi.

2) Si implementerà un’interfaccia web che permetterà all’utente di giocare come Guardia o come Ladro contro l’intelligenza artificiale.

3) Si implementeranno nuovi modelli e nuovi pattern per i labirinti.

4) Si implementerà la possibilità di creare un labirinto su più livelli.

5) Si implementeranno nuove tipologie di robot, come ad esempio droni.

6) Si implementeranno diverse condizioni meteorologiche, come ad esempio la nebbia , che ridurrà la precisione di rilevamento da parte delle guardie.

# Guida all'installazione

Scaricare e installare Java 15. [Link download.](https://www.oracle.com/it/java/technologies/javase-jdk15-downloads.html)

Scaricare e installare Eclipse. [Link download](https://www.eclipse.org/downloads/download.php?file=/oomph/epp/2021-03/R/eclipse-inst-jre-win64.exe).

Scaricare e installare Webots. [Link download](https://cyberbotics.com/).

Da Eclipse, cliccare su “**File**”🡪”**Open Projects from File System**” e selezionare il path in cui si trova il progetto: “**guardieEladri/controllers/WeBotsEclipse**”.

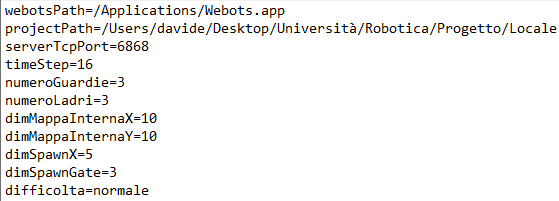
Dal PackageExplore di Eclipse, aprire il file: “**src/Config.txt**”.

Modificare i valori di “webotspath” e “projectpath” specificando rispettivamente, il path in cui è installato webots e il path in cui si trova il progetto. Modificare le altre impostazioni con i valori desiderati.

**N.B. il path del progetto deve fermarsi una cartella prima della cartella “guardieEladri”.**

Inoltre, le seguenti sono le opzioni modificabili della simulazione direttamente dal file “Config.txt”:

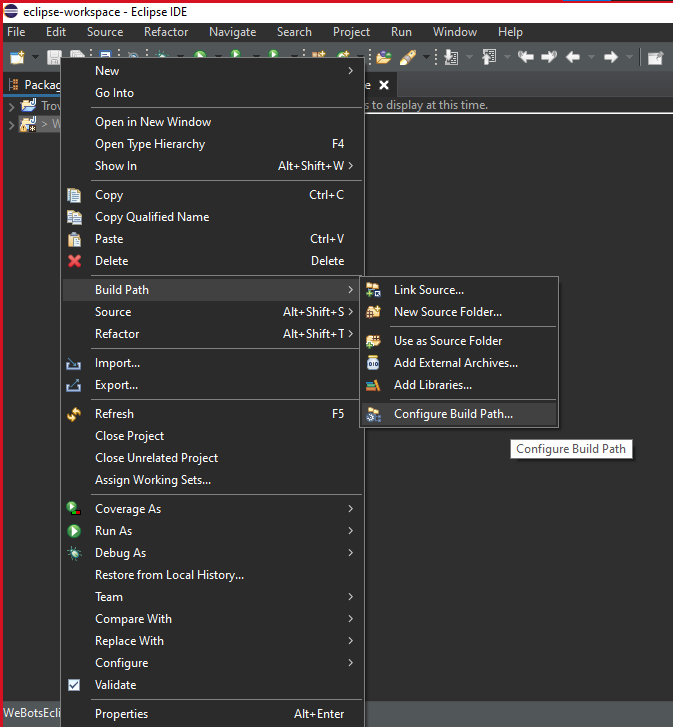
* La porta TCP in cui sta in ascolto il server.
* Il numero di robot (numeroGuardie, numeroLadri).
* Il time step utilizzato nella simulazione
* Le caratteristiche della mappa di gioco tra cui:
  + Dimensione X e Y della mappa di gioco.\*
  + Dimensione dell’altezza dell’area di spawn
  + Dimensione delle porte dello spawn.
  + Difficoltà di gioco equivalente al numero massimo di ostacoli che verranno creati; i possibili valori sono: “**facile**”, “**normale**” e “**difficile**”.

.

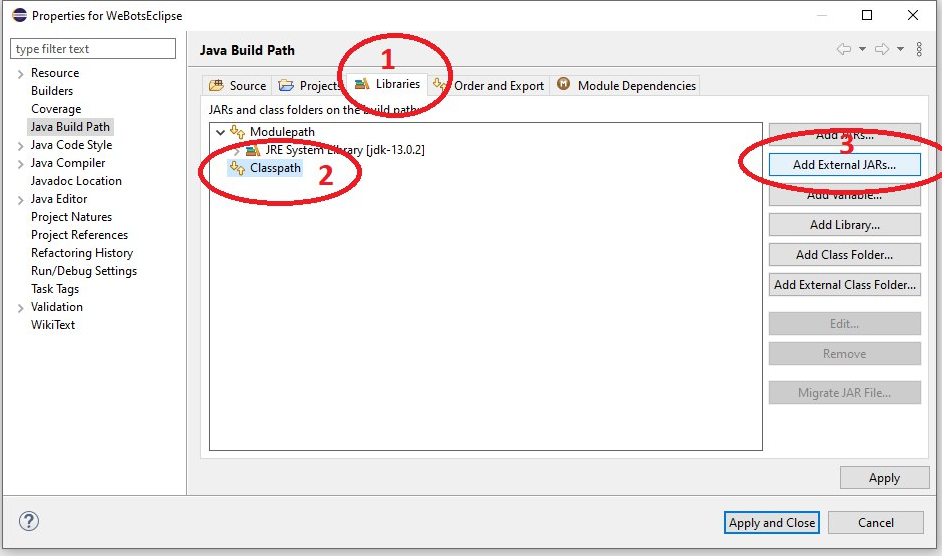
**Un’ esempio di file di configurazione**

\* La mappa è attualmente funzionante nella versione quadrata, x = y, per implementazioni future si è deciso di mantenere separati questi due valori.

Dal PackageExplore di Eclipse cliccare con il tasto destro sul nome del progetto “**WeBotsEclipse**” e andare su “**Build** **Path**”🡪”**Configure Build Path…**”:



Dal menu “**Libraries**” cliccare su “**Classpath**” e cliccare su “**Add External JARs…**”. Andare nella cartella “**/lib/controller/java**” situata nel path di installazione di Webots e selezionare il file “**Controller.jar**”.



Chiudere la finestra cliccando su “**Apply and Close**”.

Aggiungere le 3 variabili d’ambiente:

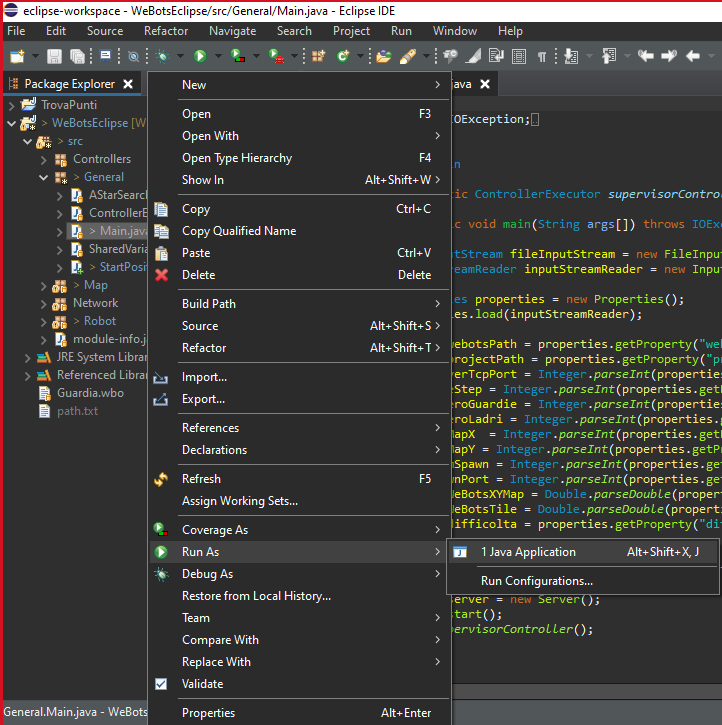
|  |  |
| --- | --- |
| WEBOTS\_HOME | { path }\Webots |
| JAVA\_HOME | { path di installazione di java } |

Aggiungere alla variabile d’ambiente **path**:

|  |
| --- |
| %WEBOTS\_HOME%\lib\controller |
| %WEBOTS\_HOME%\msys64\mingw64\bin\cpp |
| %WEBOTS\_HOME%\msys64\mingw64\bin |

Aprire il mondo da “**guardieEladri/world/guardieEladri.wbt**” tramite webots.

Tornare su Eclipse e cliccare con il tasto destro sul file relativo alla classe “**Main**” contenuto all’interno del package “General” e cliccare su “**Run As**”🡪”**Java Application**”:



**N.B: È necessario aprire sia a Webots sia Eclipse con i permessi di amministratore; inoltre, è necessario accertarsi che Eclipse stia usando Java 15 sia per la compilazione sia per l’esecuzione.**

# Conclusioni

Durante lo sviluppo della tesina sono state affrontate numerose tematiche trattate durante il corso di Robotica. Questa esperienza ci ha permesso di intrecciare le nozioni teoriche con una significativa esperienza pratica che ci ha permesso di vedere nel suo complesso l’intera fase di sviluppo di un robot a partire dalla progettazione fino alle rifiniture pratiche.

L’approccio alla parte pratica della progettazione ci ha fatto scontrare con problematiche che non avevamo preso in considerazione al tempo della progettazione iniziale come, ad esempio, la correzione del moto. Tutto ciò ha permesso di metterci a confronto con problematiche realistiche che hanno fatto sì che prendessimo più consapevolezza delle fasi di sviluppo di un sistema complesso come quello di un robot.

Gli obiettivi prefissati dal team sono stati tutti correttamente conseguiti con dei risultati soddisfacenti, l’ambiente di sviluppo Webots ci ha permesso inoltre l'utilizzo di diverse tipologie di sensori concedendoci così lo sviluppo senza alcun vincolo di budget, seppur alla fine stimato tramite vari siti online[5] a **~550 €**  per ogni e-puck con tutte le varie modifiche apportate durante lo sviluppo del progetto.

# Bibliografia

1. <https://docs.oracle.com/javase/7/docs/api/java/nio/channels/AsynchronousSocketChannel.html>
2. <https://www.baeldung.com/java-nio2-async-socket-channel>
3. <https://cyberbotics.com/doc/guide/index>
4. [Berg, C. 2001. *Amazeing Art: Wonders of the Ancient World*. Harper Collins.](https://dl.acm.org/doi/abs/10.1145/1275808.1276414)
5. <https://www.gctronic.com/e-puck.php>