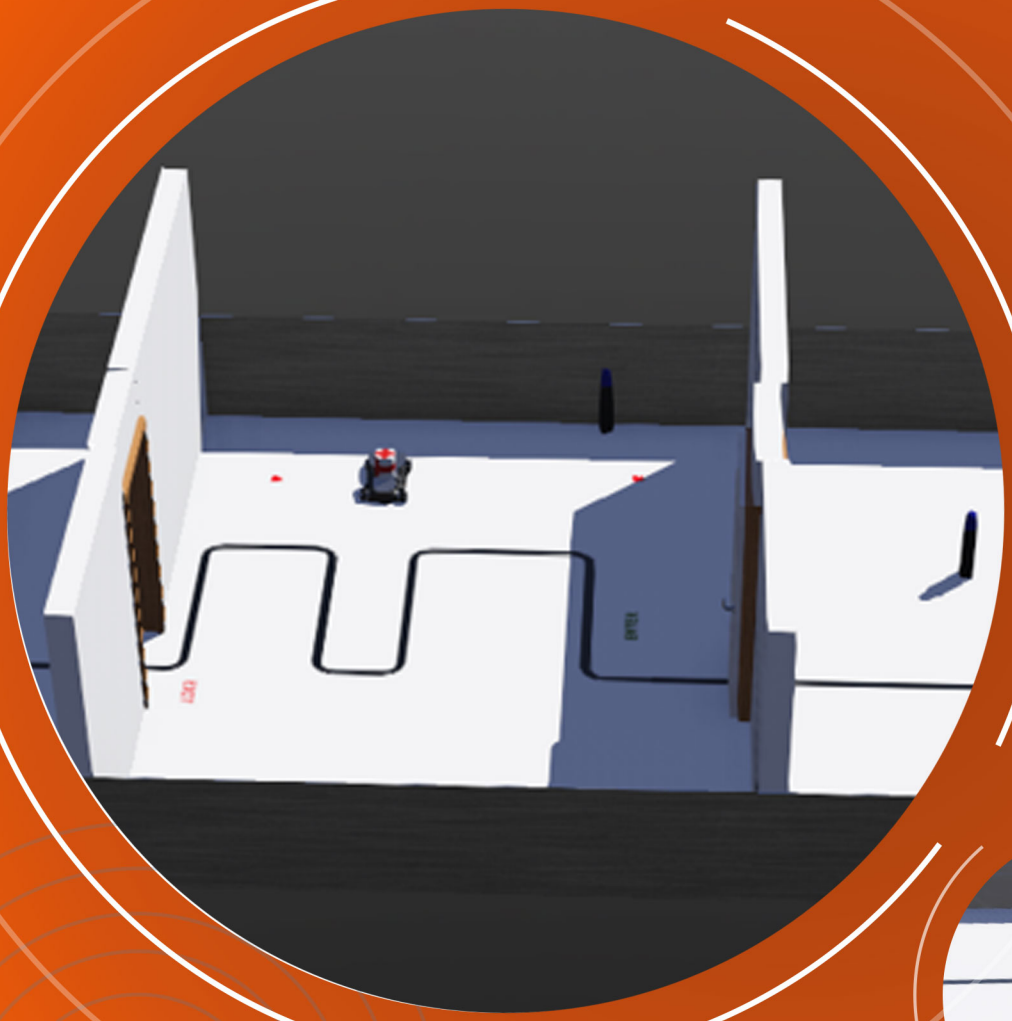
DIGANA



TEAM

GAETANO DI GRAZIA GABRIELE GAMBINO GIUSEPPE NASO

PROFESSORS	COURSE	DEGREE	A.A.
ANTONIO CHELLA ARIANNA PIPITONE	ROBOTICA	L.M. 32	2021/2022

ALMA MATER STUDIORUM PANORMITANÆ VNIVERSITATIS SIGILLVM

scuola politecnica
agroalimentare
architettura
economia
ingegneria

dipartimento
di ingegneria
unipa

Contents

1 Introduzione

Il team ha pensato che in un momento difficile come quello in cui ci troviamo adesso, fosse necessario un sistema che permetta di mantenere sicuro un luogo al chiuso e che quindi possa aiutare a limitare il diffondersi del contagio. Quindi si è pensato di creare un sistema robotico, DiGaNa, che analizzi lo status di un soggetto (sano, contagioso, altamente contagioso) e che blocchi l'accesso alla stanza a quelli contagiosi mediante una rimozione forzata. Il team ha pensato che oltre a fare una scrematura esterna monitorando gli accessi, fosse anche necessario sanificare gli ambienti in cui si muovono i pedestri.
Il sistema robotico è composto dagli e-puck, opportunamente modificati, per avere un pedestre nel turretSlot, due totem per il controllo visivo dei pedestri che vengono generati ed infine uno youbot che procede a sanificare l'ambiente seguendo delle logiche decise dal team.

2 Descrizione del team

Il team DiGaNa è composto da Gaetano Di Grazia, Gabriele Gambino e Giuseppe Naso tre studenti del corso di laurea magistrale di Ingegneria informatica presso l'Università degli studi di Palermo.

3 Descrizione del progetto

Questo progetto vuole realizzare un sistema robotico, chiamato DiGaNa. I pedestri che vengono generati nella sala d'attesa vengono innanzitutto analizzati da un totem esterno. Essi, qualora si trovassero in uno stato di buona salute (colore bianco) o in uno stato di buona salute ma con temperatura limite (colore giallo), verrebbero lasciati proseguire nel loro percorso. Viceversa, quando ci troviamo in presenza di pedestri infetti (colore arancione) o altamente contagiosi (colore rosso), ad essi verrebbe vietato l'accesso e dunque verrebbero eliminati definitivamente.
I pedestri seguiranno un percorso disegnato sul pavimento fino alla porta di uscita. Una volta fuori dal campo visivo del totem, contenente la seconda telecamera, verranno eliminati. Il sistema DiGaNa inoltre vuole mantenere sicuro l'ambiente in cui si muovono i pedestri, quindi al raggiungimento di un certo numero di pedestri all'interno della stanza (questo numero è stato impostato a 4), verrà avviata la sanificazione dell'aria attraverso un ulteriore robot, questa volta uno youbot.

4 Analisi dei requisiti

Di seguito verranno riportati i requisiti funzionali e non funzionali che il team ha elaborato per la realizzazione del progetto.

4.1 Requisiti funzionali

Per prima cosa il team ha cercato di capire quali fossero le tipologie di robot che meglio si adattano a svolgere il task richiesto. Per simulare le persone il team ha deciso di modificare la struttura di un e-puck e porre all'interno del turretSlot un pedestre. Il colore del pedestre permetterà di capire il suo stato di salute e di conseguenza i comportamenti che il sistema assumerà.
Inoltre è stato deciso che vi saranno due totem provvisti di dispositivi di acquisizione video. Il primo dispositivo avrà il compito di riconoscere lo stato dei pedestri che si apprestano ad entrare, mentre il secondo ha come compito quello di contare il numero di pedestri e, una volta raggiunti i parametri limite, far partire la sanificazione dell'ambiente attraverso un ulteriore robot.
È stato deciso che ad occuparsi della sanificazione degli ambienti sia lo youbot della Kuka. Il robot dovrà essere in grado di muoversi per la stanza e simulare la sanificazione dell'ambiente una volta raggiunto il numero massimo di persone che possono transitare all'interno della stanza. Il robot è anche dotato di un controller che gli permette di evitare gli ostacoli.

4.2 Requisiti non funzionali

Per sviluppare il seguente sistema il team ha deciso di utilizzare il simulatore robotico Webots. I controller dei vari robot verranno sviluppati utilizzando un linguaggio di programmazione al livello di astrazione opportuno. Opportuni algoritmi verranno poi implementati nei controller, per consentire ai robot di portare a termine i loro task.
È stato necessario modificare gli e-puck e dotarli di pedestri nel loro turret slot. Essi sono stati inseriti modificando il proto del robot facendo sì che le funzionalità principali degli e-puck non venissero meno. I pedestri seguiranno la linea nera traccia a terra per tutta la durata del loro task. Lasciata la stanza, e quindi superata la seconda porta, i pedestri verranno correttamente eliminati. Occorre assicurarsi che i pedestri che vengono lasciati entrare rispettino le norme previste del team, e che una volta entrati ad essi venga cambiato correttamente il comportamento. Il goal dei pedestri è dunque quello di attraversare correttamente l'anticamera, la camera principale e l'uscita. Per il raggiungimento di questo goal il team non ha imposto limiti di tempo.

Per quanto riguarda invece lo Youbot è stato inserito nel bodySlot un cilindro che conterrà il liquido sanificante che viene nebulizzato nell'aria. Il braccio meccanico inoltre è stato rimosso dagli appositi campi di configurazione.

5 Descrizione dei componenti: Robot

5.1 Epuck

E-puck è un robot sviluppato presso l'Istituto Federale Svizzero di Tecnologia di Losanna(EPFL) per scopi didattici, è un robot mobile in miniatura ispirato al robot Khepera. L'hardware e il software di e-puck è completamente open source e fornisce un accesso di basso livello a ogni dispositivo elettronico e offre possibilità di estensione illimitate. Il modello fornito dal simulatore robotico Webots include il supporto per i principali sensori e attuatori del robot, come i motori delle ruote, i sensori a infrarossi per le misure di prossimità e la telecamera. Le funzionalità del robot sono gestite dal controller che permette di accedere alle diverse componenti per implementare i comportamenti desiderati.
Ai fini progettuali l'e-puck è stato modificato con l'aggiunta di un robot pedestre inserito nel turretSlot del robot.
Questa modifica ha permesso al team di poter sfruttare le piene capacità del robot e di evitare di utilizzare principalmente il nodo pedestrian, il quale riscontra delle difficoltà in termini di grandezza e movimento.

5.2 youBot

youBot, kuka, è un braccio robotico mobile sviluppato da KUKA. Il suo braccio ha cinque gradi di libertà e una pinza lineare. La sua base ha quattro ruote Mecanum che consentono il movimento omnidirezionale. Queste ruote sono modellate in modo efficiente utilizzando l'attrito asimmetrico. Ai fini progettuali lo youbot è stato modificato aggiungendo un solido nel bodySlot del robot.
Questa modifica è puramente estetica, infatti il cilindro posto sul robot serve a simulare il contenitore del liquido sanificante che verrà nebulizzato nell'aria durante il processo di sanificazione del locale.

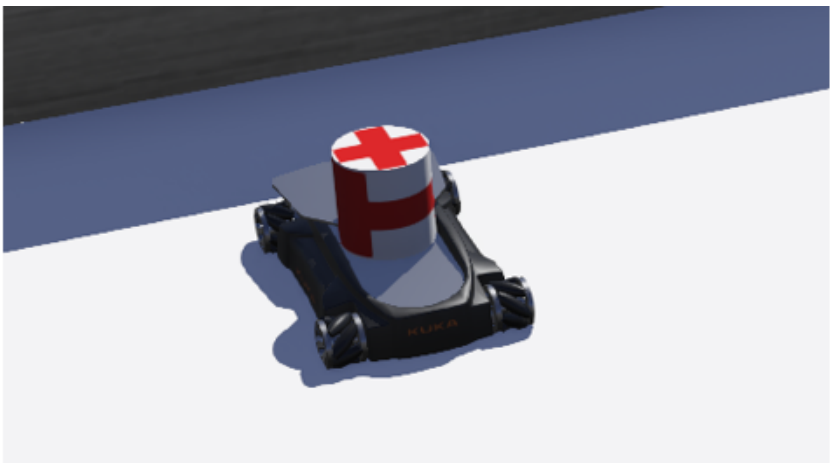
5.3 Telecamere

All'interno del progetto sono state inserite anche due telecamere per il riconoscimento del colore degli oggetti in una scena e l'ingresso/uscita dei pedestri dalla stanza. In Webots il nodo Camera

modella una telecamera esterna che può essere installata su un robot. L'immagine risultante può essere visualizzata su una finestra 3D. A seconda della sua configurazione, il nodo Camera può modellare una telecamera lineare, una tipica telecamera RGB o anche un fish eye. Ai fini del progetto è stata scelta una telecamera RGB con opzione sferica.

6 Albero del robot

Di seguito è riportata la forma finale acquisita dai robot e-puck e youbot, in seguito alle modifiche precedentemente illustrate.



7 Diagramma a stati finiti

Di seguito il diagramma a stati finiti dello youbot

```
    switch (state)
    {
    case 0:
        //printf("%s\n", "STATO 0 - VADO ALLA BASE DI RICARICA");
        state = move_to_charging_base(position_sensors);
        break;
    case 1:
        //printf("%s\n", "STATO 1 - EFFETTUO LA RICARICA");
        state = recharge();
        break;
    case 2:
        //printf("%s\n", "STATO 2 - ATTENDO CHE PASSINO 4 PERSONE");
        state = wait_for_sanitizing();
        break;
    case 3:
        //printf("%s\n", "STATO 3 - SANIFICO");
        state = goFrom1To2(position_sensors);
        break;
    case 4:
        //printf("%s\n", "STATO 3 - SANIFICO");
        state = goFrom2To3(position_sensors);
        break;
    case 5:
        //printf("%s\n", "STATO 3 - SANIFICO");
        state = goFrom3To4(position_sensors);
        break;
    }
}

wb_robot_cleanup();
return 0;
```

7.1 Moving to charging base

È lo stato iniziale del robot, iniziata la simulazione esso si muoverà verso la base di ricarica, raggiunta la base di ricarica il robot entrerà nello stato di "recharge".

7.2 Recharge

In questo stato il robot ricarica di liquido sanificante il cilindro. Quando il robot ha ultimato la ricarica il robot entrerà in uno stato di "wait_for_sanitizing".

7.3 Waiting for sanitizing

In questo stato il robot, carico di liquido sanificante, aspetta che il numero max di pedestri sia transitato all'interno della stanza prima di iniziare a muoversi verso i landmark.

7.4 Sanifico

In questo stato il robot si muoverà verso i landmark e simulerà la sanificazione dell'ambiente, inoltre grazie al controller di obstacle_avoidance eviterà di andare a sbattere contro i pedestri e quindi interferire col loro task. Una volta terminata la sanificazione il robot entrerà di nuovo nello stato di "moving_to_charging_base".

8 Mondo

Il mondo all'interno del quale si svolge il goal del robot è un'area rettangolare lunga 10 metri x 6 metri, delimitata da muri. Al suo interno troviamo due muri che ci permettono di creare tre ambienti: l'anticamera, la sala principale e l'uscita.
All'interno dell'anticamera troviamo il punto in cui verranno generati i pedestri e la prima telecamera che ha il compito di discriminare il colore dei pedestri e permettere loro di procedere lungo il percorso verso la prossima camera.
All'interno della sala principale troviamo 4 landmark che simboleggiano i goals del nostro youbot che dovrà raggiungere per completare il percorso di sanificazione. Anche in questa sala troviamo a terra la striscia nera che simboleggia il percorso che dovranno seguire i pedestri. Una volta che i pedestri raggiungono la sala d'uscita e escono dal campo visivo della telecamera, dovranno essere correttamente eliminati.

9 Controller

Il controller per l'e-puck è stato realizzato in linguaggio C ed è formato da due file epuck_turret_pedestrian ed e-puck_line_mod. Essi permettono di modulare correttamente i comportamenti dell'e-puck in base alla caso d'uso in cui ci troviamo. Anche il controller del youbot è stato scritto in C, mentre i controller delle due telecamere: totem_controller_py.py e termoscanner.py sono scritti, appunto, in Phyton. Il team ha ritenuto che ibridare lo sviluppo dei controllori in due diversi linguaggi permettesse di risolvere i problemi in maniera più efficiente e dinamica. Infatti nei controllori delle telecamere, che lavorano con array, si è utilizzato il Python poiché utile alla

programmazione procedurale. Mentre per implementare la logica dietro il funzionamento dei robot, e-puck e youbot, è stato utilizzato il C.

9.1 Controller e-puck

Il controller con cui viene gestito il movimento degli e-puck, e quindi simulato il comportamento dei pedestri, è quello che viene fornito da Webots per far seguire la banda nera che si trova a terra. L'algoritmo che permette al robot di seguire la linea è molto semplice. I tre sensori sono in grado di distinguere la linea nera (che riflette poco) dal piano su cui è disegnato il tracciato (che riflette il raggio infrarosso). I motori vengono comandati considerando che:

- se il solo sensore centrale rileva la linea il veicolo può procedere dritto (entrambi i motori alla stessa velocità);
- se il sensore destro rileva la linea il veicolo deve curvare a destra per correggere la traiettoria;
- se il sensore sinistro rileva la linea il veicolo deve curvare a sinistra per correggere la traiettoria.

9.2 Controller youBot

```
    switch (state)
    {
    case 0:
        //printf("%s\n", "STATO 0 - VADO ALLA BASE DI RICARICA");
        state = move_to_charging_base(position_sensors);
        break;
    case 1:
        //printf("%s\n", "STATO 1 - EFFETTUO LA RICARICA");
        state = recharge();
        break;
    case 2:
        //printf("%s\n", "STATO 2 - ATTENDO CHE PASSINO 4 PERSONE");
        state = wait_for_sanitizing();
        break;
    case 3:
        //printf("%s\n", "STATO 3 - SANIFICO");
        state = goFrom1To2(position_sensors);
        break;
    case 4:
        //printf("%s\n", "STATO 3 - SANIFICO");
        state = goFrom2To3(position_sensors);
        break;
    case 5:
        //printf("%s\n", "STATO 3 - SANIFICO");
        state = goFrom3To4(position_sensors);
        break;
    }
}

wb_robot_cleanup();
return 0;
```

Questo costrutto switch è il core del comportamento dello youbot, come detto prima nella sezione dei diagrammi a stati finiti questa logica permette allo youbot di muoversi e di effettuare correttamente i task cui viene chiamato a soddisfare. Nel dettaglio andremo a parlare dei singoli metodi.

```
int move_to_charging_base(WbDeviceTag position_sensors[6]
```

Metodo che fa muovere lo youbot verso la base di ricarica, allo youbot viene passata la posizione del landmark su cui dovrà fermarsi e aspettare.

```
int recharge()
```

Questo metodo consente allo youbot di ricaricare il fluido sanificante all'interno del cilindro, finita questa operazione il robot entrerà in fase di wait.

```
int wait_for_sanitizing()
```

Questo metodo mette in stato di wait il robot. Una volta raggiunta la capienza massima dei pedestri che possono transitare all'interno della stanza verrà avviata la sanificazione.

```
int goFrom1To2(WbDeviceTag position_sensors[6])
```

Questo quando viene invocato, fa muovere lo youbot dal landmark 1, ovvero la base di ricarica, fino al landmark 2.

```
int goFrom2To3(WbDeviceTag position_sensors[6])
```

Questo quando viene invocato, fa muovere lo youbot dal landmark 2, ovvero la base di ricarica, fino al landmark 3.

```
int goFrom3To4(WbDeviceTag position_sensors[6])
```

Questo quando viene invocato, fa muovere lo youbot dal landmark 3, ovvero la base di ricarica, fino al landmark 4.

```
void release_liquid()
```

Questo metodo, quando invocato, simula il rilascio del liquido nell'aria. Verrà dunque decrementato il campo description. Qualora il campo description dovesse arrivare a 0 il robot tornerà alla base per ricaricare il fluido.

```
void avoid_obstacles(WbDeviceTag position_sensors[6])
```

Questo metodo viene invocato all'interno di quasi tutti i metodi sopracitati. Infatti grazie ai sensori presenti nello youbot eviteremo che vada a sbattere contro gli ostacoli presenti nella simulazione.

9.3 Controller Totem

Il controller totem_controller_py .py rappresenta il controllore del totem interno. Abbiamo creato una classe totem_controller contenente tutti i metodi e le proprietà necessarie.

```
def openDoor(self, door):
```

È la funzione che ci permette di aprire una porta, in questo caso gli passiamo una stringa che è poi la definizione della porta che vogliamo aprire

```
def closeDoor(self, door):
```

È la funzione che ci permette di chiudere una porta, in questo caso gli passiamo una stringa che è poi la definizione della porta che vogliamo chiudere.

```
def updateCustomData(self)
```

Funzione che ci permette di fare l'update del campo customData del totem controller mediante questo abbiamo gestito, prima di renderlo manuale, lo spawn dei pedestri

```
def canSanitize(self):
```

È la funzione che modificando il campo check comunica con lo youbot permettendo che questo inizi la sanificazione

```
def getOldestIndoorPedestrian(self):
```

È la funzione che ci restituisce il primo elemento dell'array indoorPed che corrisponde al pedestre che è da più tempo all'interno della stanza.

```
    def getNumberOfVisitors(self):
```

È la funzione che restituisce il numero di pedestri riconosciuti all'interno della stanza

```
    def buildDefUseString(self, pedIndex):
```

È la funzione che ci permette di costruire la defuse dei nodi epuck/pedestre. La def di questi pedestri è composta da E più un indice incrementale

```
    def changeEpuckBehaviourToLineFollowing(self, robot_def):
```

È la funzione che ci permette di cambiare il comportamento degli epuck una volta entrati nella stanza.

```
def incrementCrossed(self):
```

È la funzione che incrementa i pedestri che sono passati dalla stanza e sono usciti; quando si arriva a 4 pedestri passati si avvia la sanificazione.

```
def getEpuck(self, id):
```

È la funzione che ci restituisce un nodo epuck a partire dall'ID randomicamente generato e assegnato da webots ad ogni singolo pedestre.

```
def getNode(self, id):
```

È la funzione che ci restituisce un generico nodo a partire dall'ID randomicamente generato e assegnato da webots agli oggetti.

```
def aPedestrianHasLeft(self):
```

È la funzione che controlla quando un pedestre è uscito mediante un controllo dei pedestri attualmente rilevati e la lunghezza dell'array

```
def aPedestrianEntered(self):
```

È la funzione che controlla quando un pedestre è entrato mediante un controllo dei pedestri attualmente rilevati e la lunghezza dell'array.

```
def registerNewPedestrian(self, id):
```

È la funzione che incrementa il contatore total spawned e aggiunge l'id del pedestre appena entrato nella stanza all'array indoorPed.

```
def kickOut(self):
```

È la funzione che rimuove fisicamente i pedestri dal mondo

```
def passive_wait(self, sec):
```

È la funzione che permette di attendere per sec secondi

```
    def check_keyboard(self):
```

È la funzione che gestisce il controllo da tastiera e che in base ai tasti premuti esegue determinate funzioni

9.4 Controller TermoScanner

Il controller termoscanner.py rappresenta il controllore del totem esterno. Abbiamo creato una classe Termoscanner contenente tutti i metodi e le proprietà necessarie.

```
def __init__(self)
```

Questo metodo è il costruttore di classe e si occupa di definire le variabili di istanza, quali il time step ad esempio.

```
def buildDefUseString(self, pedIndex):
```

Il metodo builDefUseString si occupa della costruzione delle stringhe, utilizzate nelle DEF degli E-Puck in Webots. L'indice pedIndex, che è un indice cumulativo dei pedestri che hanno visitato l'ambiente, viene associato alle suddette stringhe.

```
def incrementVisitor(self):
```

È il metodo che si occupa di incrementare l'indice dei pedestri che hanno visitato l'ambiente.

```
def getDefUse(self, i, x):
```

Il metodo getDefUse si occupa di costruire le stringhe per effettuare successivamente lo spawn di epuck e pedestri. L'indice i rappresenta l'indice cumulativo dei pedestri, x rappresenta invece la traslazione sull'asse x.

```
def generate_color(self):
```

Il metodo generate_color sceglie un colore random tra rosso, arancione, giallo e bianco, che rappresenta lo stato di salute del pedestre.

```
def newVisitor(self, index, x):
```

Il metodo newVisitor permette di aggiungere all'albero dello scenario di Webots una coppia (pedestre, e-puck), e quindi effettua lo spawning. Ottenuto il riferimento alla radice dell'albero, viene aggiunto un e-puck, e dopo aver ottenuto il riferimento al suo campo turretSlot, si aggiunge a questo un nodo Pedestrian con il relativo colore randomico; inoltre, al relativo campo groundSensorsSlot viene aggiunta l'estensione E-puckGroundSensors.proto che fornisce tre sensori infrarossi orientati verso il terreno di fronte al robot. Viene infine associato il controllore e-puck_line_mod fornito da Webots.

```
def spawnSingle(self, index, pos):
```

Chiama i precedenti metodi spawnSingle e incrementVisitor.

```
def getEpuck(self, id):
```

Restituisce il nodo dell'e-puck associato all'id del pedestre visto dal totem. Questo metodo è invocato da kickOut al momento dell'eliminazione della coppia (pedestre, e-puck).

```
def kickOut(self, id):
```

Il metodo kickOut rimuove, dall'albero dello scenario di Webots, i pedestri "infetti", cioè i pedestri di colore arancione o rosso.

```
def openDoor(self, door):
def closeDoor(self, door):
```

Sono i due metodi che si occupano di settare rispettivamente a -1.5 e 0 il campo position della porta, la cui DEF è specificata dalla stringa door.

```
def isInfected(self, id):
```

È il metodo che determina se un pedestre è "infetto" o meno sulla base del colore della pelle.

```
def check_keyboard(self):
```

Metodo che si occupa di rilevare la pressione del tasto "S" sulla tastiera. In caso positivo, viene effettuato lo spawning di un pedestre.

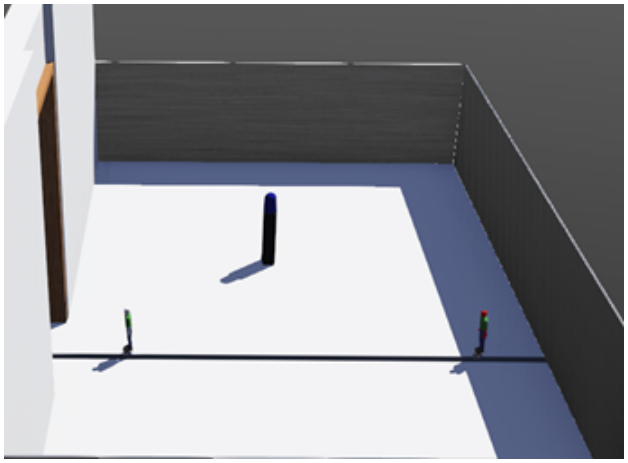
```
def run(self):
```

Rappresenta il main del controllore. In questo metodo viene abilitata la telecamera del termoscanner, denominata “camera_totem_2”. Nel ciclo while interno al metodo, ad ogni ciclo, si rilevano l’eventuale pressione del tasto S della tastiera, e inoltre gli eventuali pedestri che fanno parte del campo visivo della telecamera: se questi sono “infetti” vengono quindi eliminati.

10 Casi d’uso

Di seguito riportati i 4 casi d’uso che mostrano il comportamento dei robot in tre situazioni diverse.

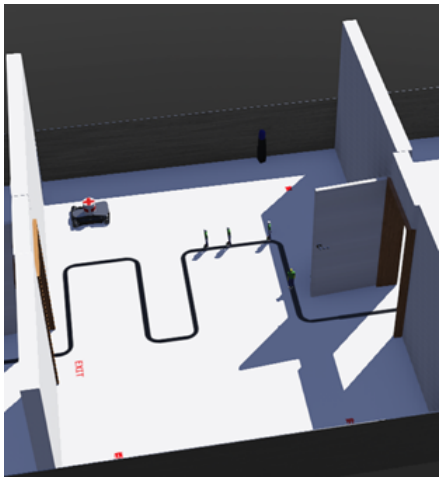
10.1 Caso d’uso 1: efficienza



Con questo caso d’uso si vuole dimostrare il corretto rilevamento dello status dei pedestri. Il primo (in figura a sx) verrà lasciato entrare, mentre il secondo (a dx) verrà eliminato. A questo punto verrà generato randomicamente un pedestre di un altro colore, qualora questo venisse lasciato passare continuerebbe il suo percorso lungo la stanza, viceversa verrebbe eliminato e lascerebbe spazio ad un nuovo pedestre.

La difficoltà in questo caso d’uso è stata classificata come facile-medio, ci si aspetta che in circa 30 secondi almeno due pedestri abbiano varcato la soglia della stanza principale. Lo scenario proposto è composto da un pedestre bianco, che quindi potrà entrare, e un pedestre rosso che verrà eliminato una volta raggiunto il campo visivo della telecamera.

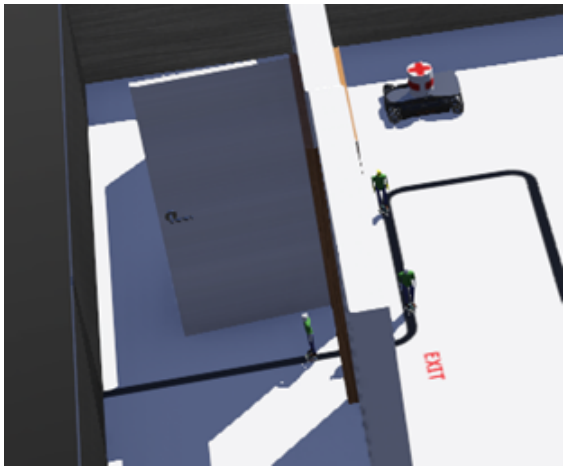
10.2 Caso d’uso 2: Navigazione all’interno della stanza



Con questo caso d’uso si vuole dimostrare il corretto percorrimento della stanza da parte dei pedestri. Inoltre quando viene raggiunto il limite massimo di persone che possono transitare nella stanza, verrà avviata la sanificazione da parte dello youbot.

Il percorso che lo youbot effettuerà è definito dai landmark presenti nella stanza e visibili in figura. La difficoltà di questo caso d'uso è stata classificata come facile-medio, infatti ci si aspetta che i pedestri transino correttamente all'interno della stanza e che lo youbot sanifichi la stanza senza andare a interferire col percorso dei pedestri. Nello scenario proposto vediamo lo youbot fermo nella base di ricarica e i pedestri che percorrono il percorso prestabilito dalla linea nera.

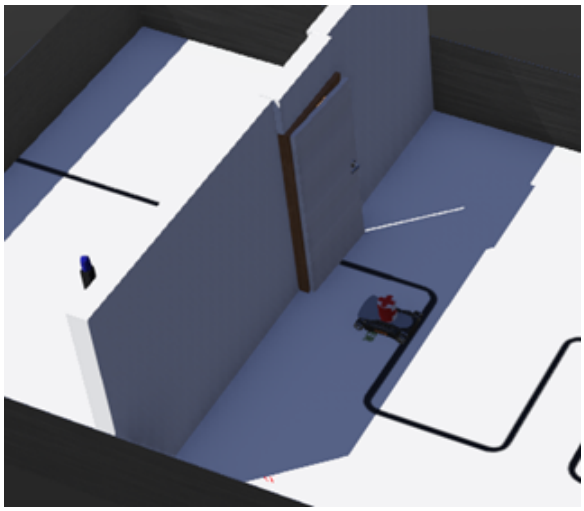
10.3 Caso d'uso 3: Eliminazione dei pedestri



Con questo scenario si vuole dimostrare la corretta eliminazione dei pedestri da parte del totem nella sala. Una volta che i pedestri sono usciti dal campo visivo del totem devono essere eliminati dalla simulazione. La difficoltà di questo caso d'uso è stata classificata come difficile.

Infatti il team ha dovuto gestire correttamente gli ID dei pedestri che vengono generati con quelli che vengono lasciati passare dal primo totem ed infine da quelli che vengono riconosciuti dal totem interni. Nello scenario proposto notiamo come il pedestre ha varcato la soglia della porta di uscita e sta per essere eliminato dalla simulazione.

10.4 Caso d'uso 4: Sanificazione



Con questo caso d'uso si vuole dimostrare il corretto funzionamento della sanificazione operata dallo youbot. Ogni volta che il numero di persone che transitano per la stanza centrale (quindi hanno varcato la porta d'uscita) raggiunge un valore multiplo di 4, lo youbot comincerà la sanificazione della stanza.

A partire dal landmark identificato come base, si sposterà verso i 3 landmark successivi percorrendo un quadrato. Durante l'esecuzione di questo caso d'uso il quantitativo di sanificante verrà decrementato fino all'esaurimento, lo youbot dunque tornerà di nuovo nella base di ricarica aspettando di ripetere il ciclo. La difficoltà che il team ha associato a questo caso d'uso è facile medio.Il team ha dovuto gestire la collision avoidance coi pedestri e il raggiungimento dei landmark da parte del robot. Nello scenario proposto notiamo lo youbot che si sta spostando dal landmark 3 al landmark 4