SERVER

1. main()

Nella funzione main è inclusa la creazione del socket **UDP** che resta in loop,
in ascolto di messaggi da parte dell' **HttpHandler** .
Ricevuto un messaggio, esso viene "Parsato" e incapsulato in un oggetto di tipo **"Message".**
Il messaggio viene inserito in una "Concurrent_Queue" ossia la coda messaggi.
La coda verrà consumata da Thread dedicati (1-4) e inseriti i relativi dati nell'albero robot.

La coda messaggi, diventerà fondamentale nel momento in cui il sistema si troverà in **Update,** dal momento che, essendo basato su un'esecuzione parallelizzata**,** il thread principale dovrà comunque continuare la ricezione dei messaggi, mentre gli inserimenti sull'albero saranno sospesi sino a fine update.

2. queueConsumer()

Consuma la coda messaggi condivisa affidandosi a **Thread dedicati**.
Ad un intervallo di **tempo prestabilito** sarà invocato un aggiornamento del sistema, che **metterà in pausa il consumo della coda** in attesa del suo completamento.

3. consumeMessage() [1-4]

Task affidato al **Thread**.
Si occuperà di controllare **l'esistenza** di un robot nell'albero:

Se **esiste**, effettuerà l'inserimento del **cambio di stato**.

Se **non esiste**, effettuerà **l'inserimento del robot**.

ROBOT-TREE

1. getCurrentTime()

Restituisce il timestamp corrente espresso in millisecondi (Since Unix Epoch).

2. addCluster()

Aggiunge un cluster all’albero.

3. addArea()

Aggiunge un’ area all’albero.

4. updateRobot()

Aggiunge un cambio di stato ad un robot.

5. printRobot() printRobotObj()

Stampa le informazioni relative al robot.

6. createDummyTree()

Crea un alberto parametrico a fini di test.

UPDATE

1. systemUpdate()

Aggiornamento generale del sistema: **per ogni area e per ogni cluster** vengono **calcolati gli IR di tutti i robot presenti** (tramite getBotIr() in “irEngine.h”) e **inseriti nel Json corrente.**

Ogni volta che un cluster viene completato, viene generato il rispettivo Json e il processo si ripete per tutti gli altri cluster del sistema**.**

2. createJson(Json json)

Genera il Json passatogli come parametro, impostando il path e il nome del file.

IR-ENGINE

Calcolo IR

1. **getBotIr()**
2. **upDown()**
3. **secDuration()**

L**' IR** sarà così calcolato:

Passo 1 : Entry point **getBotIr**(sigMap currentBot,long istant)

Il metodo prenderà come parametro il robot corrente, esso sarà costituito da una mappa di tipo **sigMap** ossia **std::map<string,sigLineMap>** E.g. <"s1",sigLineMap>
dove **sigLineMap** è **std::map<long,char>** E.g. <1294738574827,'1'>

Nel sistema i nodi dell'albero (Robots) saranno appunto delle **sigMap**, cioè mappe che avranno 7 chiavi (Segnali) e per ognuna di esse sarà associata un'ulteriore mappa (**SigLineMap**) che avrà come chiavi i **timestamps dei cambiamenti di stato** e come **valore '0' oppure '1'** ad identificare rispettivamente **"UP" e "DOWN".**

Vengono **lette le mappe di tutti i segnali** ed inserite in 7 vettori distinti le chiavi (**timestamps**) **maggiori o uguali** al valore di **ISTANTE_UPDATE - FINESTRA_TEMPORALE.**
Inoltre **tutte le chiavi trovate** vengono **inserite** in un vettore denominato "**allKeys**".

I vettori vengono **ordinati** in ordine crescente.
Vengono **rimossi i duplicati** dal vettore "**allKeys**".

Attraverso il vettore "**allKeys**" vengono **individuati dei settori**, dove inizio e fine sono timestamp di cambiamenti di stato oppure i valori degli estremi della finestra temporale, **inclusi**.

Si andrà adesso a determinare i valori **up e down di ogni segnale**
in ciascuno dei settori.

Up down sarà **chiamato in totale 7 volte, restituendo** 7 array di char.

Passo 2 : Chiamata di **upDown**(vector<long> allKeys,
int secNumber,
sector* sec,
sigMap currentBot,
vector<long> v,
string sig,
long istant)

Prenderà come parametri:

Il numero di settori **"secNumber".**

Puntatore all'array di struct contenente i settori **"sec".**

Robot (sigMap) corrente **"currentBot".**

Vettore contenente le chiavi del segnale **"v".**

Chiave(codice) del segnale corrente **"sig".**

Timestamp dell'istante in cui verrà richiesto l'update **"istant".**

Il metodo restituirà un array di char contenente in sequenza i valori di up e down per ogni settore.

L'array conterrà valore "1" per i settori up "0" viceversa.

Il metodo ritorna l'array a **"getBotIr() .**

Passo 3 : ritorno a **getBotIr()**

Si **cerca nei vettori appena ottenuti da upDown()** per ogni settore **la presenza di <u>almeno</u> un valore “Down” (0)** .

Se esso viene trovato si effettua una **chiamata a secDuration(sector* s, int i)** che restituisce la durata in minuti del settore di indice i .

Questo valore viene quindi **aggiunto alla variabile “downTime”** inizializzata a 0 .

In caso invece **non viene trovato alcun segnale di “Down”** , il settore viene considerato up e calcolata **la sua durata, viene aggiunta ad “upTime”** anch'essa inizializzata a 0 .

Viene assegnato alla variabile iR :

la **parte intera** di (**downTime / FINESTRA-TEMPORALE**)
arrotondando **per eccesso.**

L’ IR viene restituito a “systemUpdate()” **in** ‘update.h’ **.**

PARAMETERS

Contiene dei valori configurabili per cambiare il comportamento del sistema.

```
/* DEBUG MODE */

#define DEBUG 0 // 0 disable 1 enable
#define VERBOSE 0 // display messages on console

/* PERFORMANCE TWEAKS */

#define THREAD_COUNT 1 // How many thread will consume the queue
#define QUEUE_SLEEP 500 // How much time Queue Consumer will wait in each loop with "Empty Queue"

/* SETTINGS */

#define UPDATE_INTERVAL 15 // Time in sec between every update
#define BUFSIZE 2048
#define SERVICE_PORT 222

/* IR */
#define TIME_WINDOW 60 // Time in minutes | Calc IR based on (downtime/time-window)*100


/* SIMULATION */

//file: robotTree.h method: generateDummyTree()

#define BORN_TIME 15 //time in minutes used to calc: ARRIVE_ISTANT = (CURRENT_TIME - BORN_TIME)
#define N_CLUSTERS 100 //cluster number
#define N_ROBOTS 900 //robot number per cluster
```

È necessario ricompilare il sorgente perché le modifiche abbiano effetto.