

Treball de Fi de Grau

## **Grau en Enginyeria en Tecnologies Industrials**

# **Disseny del recorregut òptim d'un robot de desinfecció amb UV-C en diversos escenaris**

### **MEMÒRIA**

**Autora:** Cristina Hernandez Miguel  
**Director:** Samir Kanaan-Izquierdo & Gerard Escudero  
**Convocatòria:** Abril 2021



Escola Tècnica Superior  
d'Enginyeria Industrial de Barcelona





## Resum

L'objectiu d'aquest projecte és crear l'algoritme de control d'un robot de desinfecció UV-C, de manera que sigui el més eficient possible.

Per una banda, s'ha dissenyat el comportament del robot utilitzant l'algoritme de *pathfinding*, o cerca de camí, A\*, necessari per evitar la col·lisió amb obstacles o parets i per buscar la manera en què es porta a terme moviments complexos.

També s'ha tingut en compte les característiques tècniques del robot; com per exemple, les dimensions de la base.

Per altra banda, s'ha creat una interfície gràfica utilitzant Unity, de manera que en tot moment es pugui observar de forma senzilla el comportament de l'algoritme i el robot.

El projecte s'ha estructurat en 3 fases, de manera que en cada una d'elles, l'algoritme pren més complexitat. En la primera fase s'ha creat la interfície i s'han programat les bases de l'algoritme, en la segona, s'ha tingut en compte la bateria, la càrrega i l'existència de varies sales i en la tercera, s'ha adjudicat un nivell de brutícia a les cel·les de les sales.

Finalment, s'ha avaluat el rendiment de l'algoritme per cada una de les fases, conclouent en quina de les casuístiques dels escenaris el seu comportament és més òptim.

## Abstract

The objective of this project is to create the control algorithm of a UV-C disinfection robot so that it as efficiently as possible.

On one hand, the robot performance has been designed using the pathfinding algorithm A\*, useful to avoid colliding with obstacles or walls and to find how to make complex movements.

Also, robot technical specifications have been taken into consideration; for example, the base dimensions.

On another hand, a graphic interface has been created using Unity, so that at any moment the behaviour of the robot and algorithm can be analyzed easily.

The project is structured in 3 phases, so in each one, the algorithm becomes more complex. In the first phase, the interface has been created and the bases of the algorithm have been programmed. In the second one, I considered the battery, the charging process and the possibility to have more than one zone. In the third one, each cell has a level of dirtiness attributed.

Finally, in each phase, the performance has been evaluated, which allowed me to conclude in which cases the algorithm behaviour was more optimal.



# Sumari

<b>RESUM</b>	<b>3</b>
<b>ABSTRACT</b>	<b>4</b>
<b>SUMARI</b>	<b>6</b>
<b>1. INTRODUCCIÓ</b>	<b>9</b>
1.1. Descripció del problema	9
1.2. Objectius del projecte	10
1.3. Abast del projecte	10
<b>2. ROBOTS UV-C</b>	<b>11</b>
2.1. Descripció	11
2.2. Usos	11
2.3. Característiques tècniques	15
<b>3. ASPECTES TÈCNICS</b>	<b>19</b>
3.1. Zona de treball	19
3.2. Bases de l'algoritme	20
3.3. Xarxes de cel·les. Creació i importació	21
<b>4. FASE 1</b>	<b>24</b>
4.1. Casuística	24
4.2. Objectius	25
4.3. Desenvolupament	26
4.4. Resultats i avaluació	28
<b>5. FASE 2</b>	<b>31</b>
5.1. Casuística	31
5.2. Objectius	31
5.3. Desenvolupament	32
5.4. Resultats i avaluació	33
<b>6. FASE 3</b>	<b>37</b>
6.1. Casuística	37
6.2. Objectius	37
6.3. Desenvolupament	37
6.4. Resultats i avaluació	38
<b>7. PLANIFICACIÓ TEMPORAL</b>	<b>42</b>

<b>8. ESTUDI ECONÒMIC</b>	<b>43</b>
<b>9. ESTUDI AMBIENTAL</b>	<b>44</b>
<b>CONCLUSIONS</b>	<b>45</b>
<b>VISIÓ DE FUTUR</b>	<b>46</b>
<b>AGRAÏMENTS</b>	<b>47</b>
<b>BIBLIOGRAFIA</b>	<b>48</b>
Referències bibliogràfiques .....	48
Bibliografia complementària .....	48
<b>ANNEXOS</b>	<b>50</b>





# 1. Introducció

## 1.1. Descripció del problema

La neteja i desinfecció dels espais públics, com hospitals, escoles o centres comercials, és molt important per assegurar una higiene correcta i per evitar el contagi i propagació de virus, bacteries i altres patògens entre les persones que utilitzen aquests espais.

Tot i que la majoria d'aquests llocs tenen equips de neteja i es preocupen per la higiene, els mètodes de neteja i desinfecció actuals tenen limitacions, ja que es porten a terme per persones. A més a més, amb el COVID-19 s'ha vist que, fins i tot, aquest feina pot ser perillosa per les persones que la porten a terme, ja que estan en primera línia de contacte amb el virus.

Per aquest motiu, els robots de desinfecció UV-C són tant útils i necessaris, ja que permeten una eliminació de patògens al 99%, tenen baixes probabilitats d'error i permeten que tant les persones encarregades de la neteja com les persones que podrien quedar infectades estiguin més segures.

No obstant, cal que l'algorisme de neteja dels robots UV-C permeti un comportament òptim, que asseguri aquest 99% de desinfecció. Basant-me en aquesta necessitat, he proposat els objectius del projecte.



*Imatge 1. Model C de UVD Robots*

## 1.2. Objectius del projecte

L'objectiu principal del projecte és idear l'algoritme de control d'un robot UV-C, de manera que netegi tota una zona o escenari de forma òptima.

L'objectiu principal està subdividit en els següents sub-objectius:

- a) Crear una interfície gràfica per poder veure gràficament el comportament del robot.
- b) Afegir obstacles i parets.
- c) Integrar l'algoritme de cerca de camí.
- d) Afegir cel·les de càrrega.
- e) Tenir en compte la bateria del robot.
- f) Distribuir l'espai en diferents sales.
- g) Adjudicar un nivell de brutícia a les sales.

Per a crear l'algoritme, s'han repartit els sub-objectius en 3 fases de treball. D'aquesta manera, s'ha pogut desenvolupar i avaluar el comportament de l'algoritme a mesura que anava prenent complexitat.

En la primera fase es contemplen els tres primers sub-objectius. En la segona, s'integren les cel·les de càrrega, la bateria i les sales. Finalment, en la tercera, es té en compte el nivell de brutícia.

## 1.3. Abast del projecte

L'abast del projecte es basa, fonamentalment, en el fet que el robot coneix a priori la distribució de la sala, els seus elements i la brutícia. És a dir, no hi ha cap mena de reconstrucció de l'espai a temps real.

Permetre que el robot no conegués el seu entorn des de l'inici augmentava molt la complexitat del projecte i no estava dins dels objectius.

Un altre aspecte a tenir en compte és que el temps no influeix. Per exemple, el robot es carrega de forma instantània quan es mou a una cel·la de càrrega.

He decidit no incloure el temps en el projecte perquè no influeix en el comportament de l'algoritme; les decisions que prengui no tenen cap component temporal.

Finalment, l'abast també ve definit pel fet que l'escenari és estàtic. És a dir, no es pot canviar la distribució de la sala, la ubicació del robot ni poden aparèixer nous elements un cop el robot s'ha posat en funcionament.

## 2. Robots UV-C

### 2.1. Descripció

Els robots UV-C es basen en l'ús de llum ultraviolada (UV) del grup C per, mitjançant la seva radiació, destruir l'ADN i ARN dels patògens que es puguin trobar en les superfícies de sales i espais i evitar que s'expandeixin o tornin a aparèixer.

Cada cop més, la desinfecció d'espais compartits és vital per a la seguretat de les persones. Ja sigui escoles, hospitals, hotels, zones de lleure o restauració, assegurar una correcta neteja de les superfícies és important per reduir la possibilitat de propagar malalties.

Tot i que es podria pensar que la neteja manual és suficient, no és així. Al ser un treball humà, sempre hi pot haver errors o diferències en l'eficàcia dels treballadors. A més a més, estem parlant d'eliminar organismes que no es poden veure i, per tant, costa més identificar fins a quin punt s'ha aconseguit netejar totes les zones. És per això que els robots UV-C són tant útils i necessaris, ja que aquest mètode de desinfecció permet eliminar fins el 99,99% dels virus, bacteries i fongs d'una manera automatitzada i, per tant, controlable.

Per altra banda, és un aparell segur per als humans. Els robots tenen incorporat un sistema de detecció automàtica de moviment, que els permet desactivar els raigs UV quan s'identifiqui la proximitat d'una persona o un altre ésser viu, i així evitar que rebin radiació ultraviolada.

Alguns dels fabricants i distribuïdors de robots UV-C són SEWERTRONICS™ a Espanya, AIS a Canadà o UVD Robots® a Dinamarca.

### 2.2. Usos

L'ús dels rajos UV per a la desinfecció és una tecnologia que data del segle XX. Es tenen registres que l'any 1937 es va utilitzar per desinfectar els quiròfans del Duke University Hospital, ja que els mètodes tradicionals no eren prou eficaços i, l'any 1972, es va instal·lar en el sistema de ventilació d'una escola i es va aconseguir reduir el contagi de xarampió.

Els primers robots basats en l'ús de la llum UV-C van aparèixer fa 15 anys, pensats per a la desinfecció d'hospitals.

En l'actualitat, el seu ús està molt més estès i especialitzat, cobrint les necessitats de diversos sectors, com escoles, hospitals o aeroports. Algunes de les solucions proposades van des de robots sense mobilitat, fins a robots que fan una detecció de la zona a netejar per calcular el nivell de rajos UV-C, robots amb mobilitat autònoma o sistemes de diversos dispositius que permeten una desinfecció més flexible.

En el cas dels hospitals, per exemple, és molt important assegurar una desinfecció correcta, ja que hi ha un gran nombre de pacients que s'infecten durant la seva hospitalització. A més a més, si s'opta per un robot autònom, no cal que el personal es

posi en perill accedint a les zones infectades, sinó que el robot automàticament portarà a terme la neteja de la manera més eficient possible.

Per altra banda, molts dels robots tenen control remot i càmeres amb les quals es pot monitoritzar el seu funcionament.

L'ús d'aquesta tecnologia ja ha sigut provada en institucions sanitàries importants, com les clíniques dentals Boston Dental a Estats Units o a l'Hospital Abano Veneto a Itàlia.



*Imatge 2. Boston Dental*



*Imatge 3. Hospital Abano Veneto*

Un altre sector altament interessat en assegurar una desinfecció i una neteja adequades dels seus espais és l'hoteler. Ja sigui les habitacions, les zones compartides (gimnasos, sales d'estar, etc) o els restaurants, tenir la certesa que són àrees segures, permet oferir confiança als clients i, per tant, desmarcar-se com una opció millor i més responsable.



*Imatge 4. UV-C Robot per a hotels*

Finalment, els altres sectors que poden veure els robots UV-C com la seva millor opció a l'hora de desinfectar els seus establiments, són les escoles, els aeroports, les estacions de transport públic, les fàbriques o els centres comercials.



*Imatge 5. UV-C Robot per a centres comercials*



*Imatge 6. UV-C Robot per a escoles*

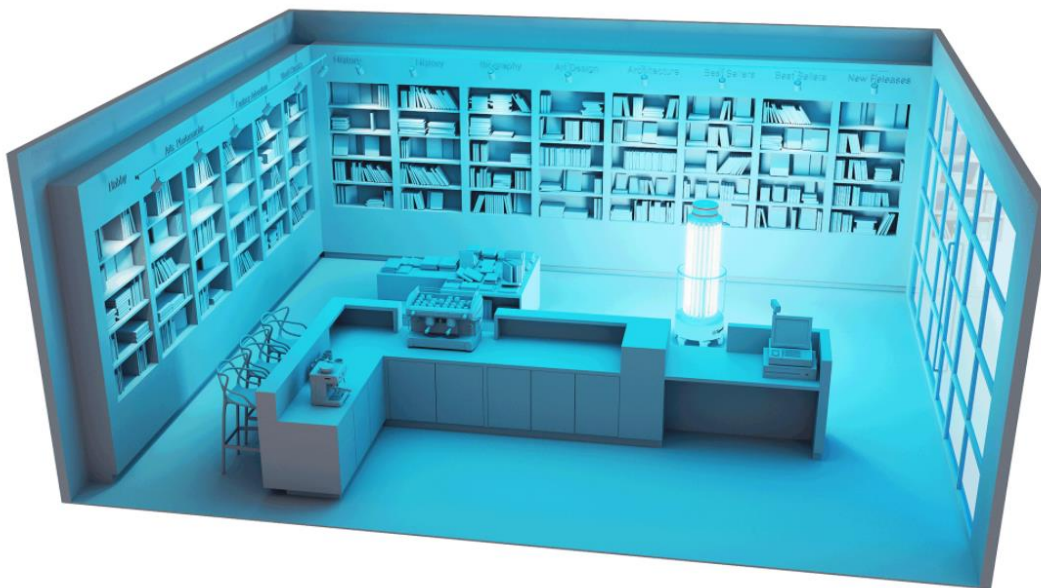




*Imatge 7. UV-C Robot per a aeroports*

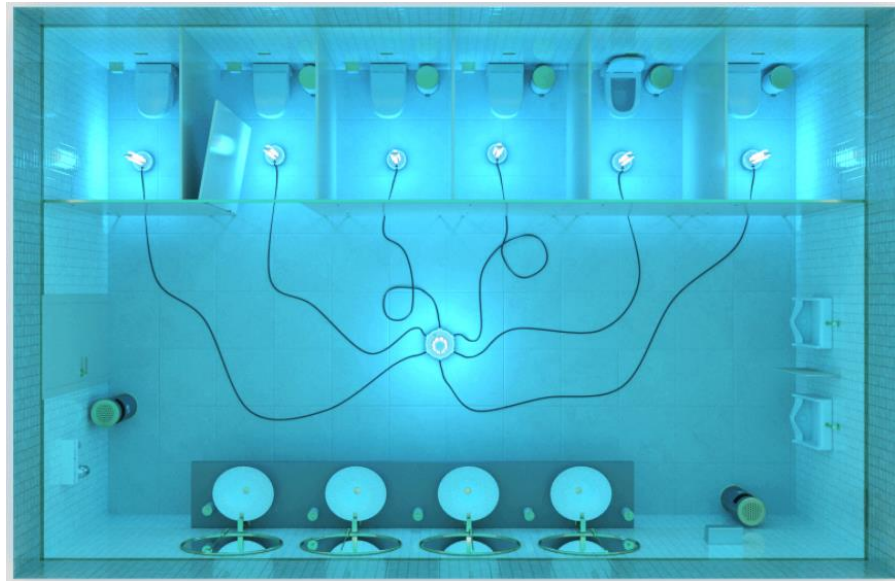
No obstant, no sempre un robot autònom pot ser la millor solució; també existeixen dispositius fixos. Aquests es col·loquen en un punt concret de la sala, i ells mateixos fan una reconstrucció de l'espai i calculen el temps i la dosi de llum necessària per a una correcta desinfecció, tant de les superfícies com de l'aire.

És una opció ràpida i segura, ja que es pot mantenir la sala buida i tancada i, per tant, evitar una possible interacció amb les persones.



*Imatge 8. SpeedyCare™ 1500*

També existeixen els robots formats per diversos dispositius, permetent una neteja més adaptable a cada tipus d'espai.

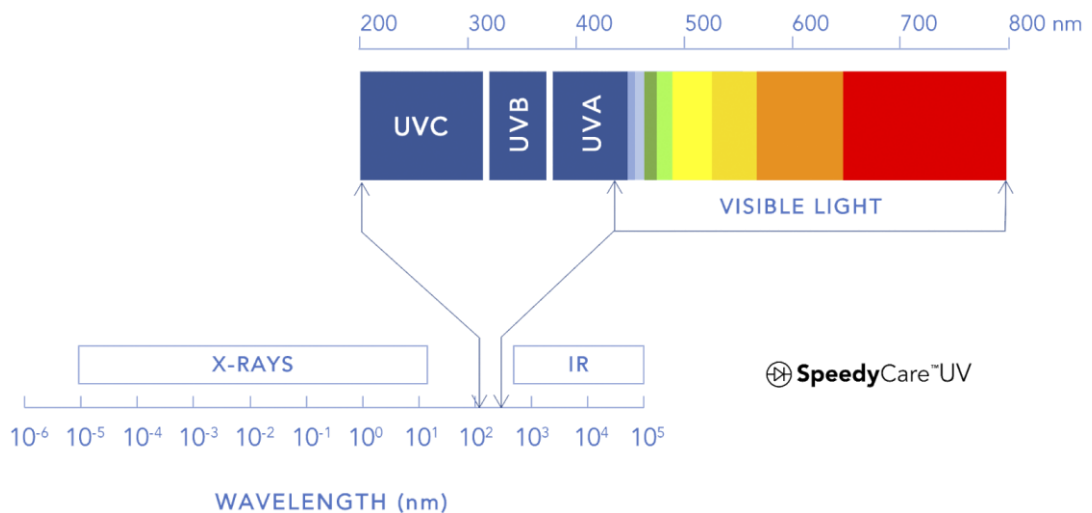


Imatge 9. SpeedyCare™ Satellite

### 2.3. Característiques tècniques

La llum ultraviolada (UV) és una radiació electromagnètica invisible, que es mou entre els valors de longitud d'ona de 100 nm i 400 nm. En l'espectre lumínic es troba entre la llum visible i els rajos X.

A més a més, està subdividida en 3 grups; un d'ells és el grup C, l'utilitzat pels robots de desinfecció UV-C.



Imatge 10. Explicació rang de llum UV-C

La llum ultraviolada natural prové del sol, tot i que l'atmosfera s'encarrega de no deixar passar la major part dels rajos del grup B i cap dels rajos C. La llum del grup A si arriba a la Terra, però no és un problema perquè és l'única d'elles que no és germicida.

La llum del grup B és radiació actínica, és a dir, causa reaccions fotoquímiques, mentre que la del grup C és radiació germicida, això és, elimina patògens causant canvis fotoquímics en els àcids nucleics dels microorganismes.

La manera en què es calcula el temps d'exposició a la llum necessari per eliminar els patògens d'una sala és funció de quatre paràmetres:

- Susceptibilitat del patogen a la llum UV-C
- Factor de reducció logarítmica (LRF)
- Potència de radiació del dispositiu (DRP)
- Distància del dispositiu al patogen que s'està netejant

El factor de reducció logarítmica és el paràmetre matemàtic que expressa el percentatge de microorganismes eliminats en el procés de desinfecció, calculat com el logaritme en base 10 de la relació entre les unitats de patògens abans i després de l'exposició a la llum UV.

Reducció logarítmica	Factor de reducció	Percentatge reduït
1	10	90%
2	100	99%
3	1000	99,9%
4	10000	99,99%

*Taula 1. Reducció logarítmica [1]*

El model de robot que he escollit com a referència de les característiques tècniques per realitzar el projecte és el UVD Robot® Model B.

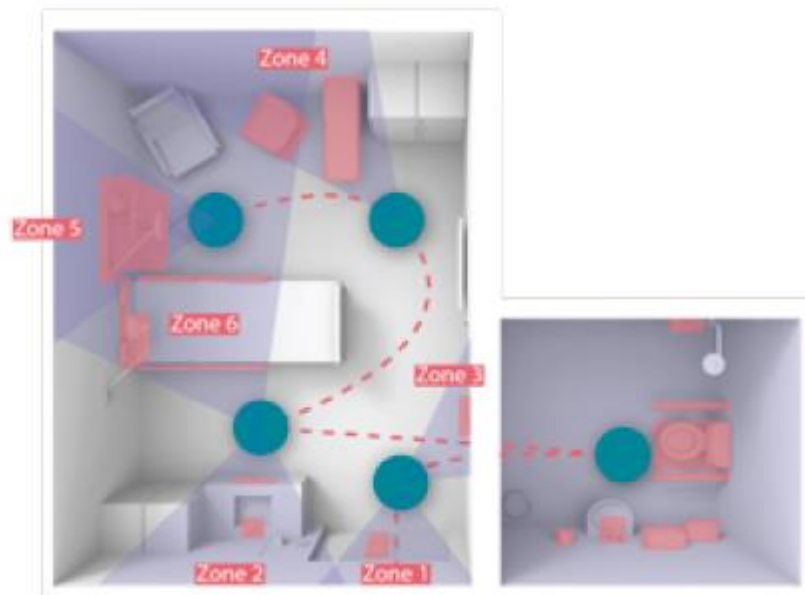
He decidit utilitzar aquest model concret perquè està fabricat per una de les empreses més importants del sector, desinfecta de manera autònoma, és mòbil i permet controlar el seu funcionament de forma remota amb una tableta que porta incorporada.





Imatge 11. Cicle de funcionament UVD Robot Model B


Aquest robot està pensat principalment per utilitzar-se en hospitals, ja que permet desinfectar zones reduïdes com els lavabos de les habitacions dels pacients. No obstant, també pot usar-se en espais grans com aeroports o magatzems.



Imatge 12. Funcionament del robot en una habitació d'hospital

Les especificacions tècniques són les següents:

### UVD ROBOT® MODEL B



DIMENSIONS: L: 93 x W: 66 x h: 171 (cm)

### TECHNICAL SPECIFICATIONS

- TOTAL WEIGHT.....140 kg
- OPERATING TIME..... 2-2.5 hours (disinfects up to 12,000 m<sup>2</sup>)
- BATTERY CHARGING TIME.....4 hours
- DISINFECTION COVERAGE..... 360 degrees
- DISINFECTION TIME.....8 min - Hotel room, office, cabin (25 m<sup>2</sup>)  
30 min - Conference room, warehouse (500 m<sup>2</sup>)
- CONNECTIVITY..... Wireless (Wi-Fi based)
- MAX SPEED.....5.4 km/h
- UV-C WAVELENGTH..... 254 nm (Ozone free)
- UV-C LAMP LIFESPAN.....12,000 hrs of disinfection
- CHARGING REQUIREMENTS....220-240 VAC, 50 Hz, 6 Amps
- SAFETY..... Software & Sensors Based  
Emergency Stop Button

*Imatge 13. Especificacions tècniques UVD Robot Model B*

Per al disseny del meu algoritme i de les àrees de testeig, he tingut en compte:

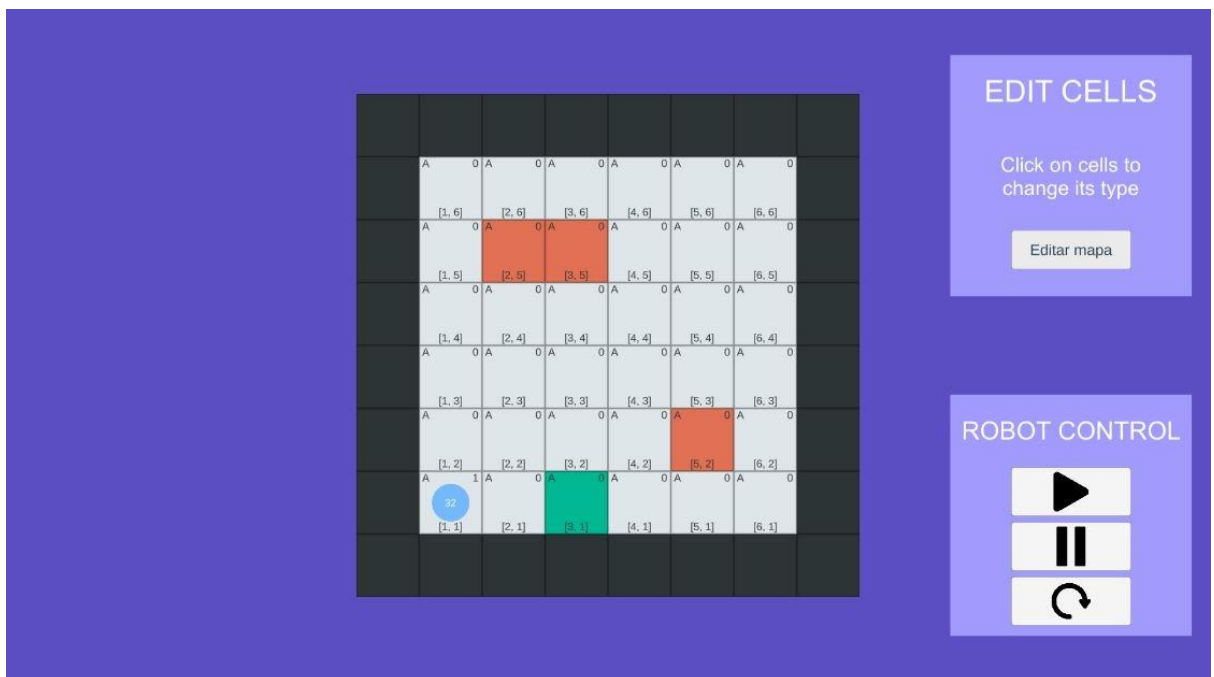
- **Les dimensions de la base.** Per relacionar-les amb les mides de les cel·les de la xarxa que conforma les zones de prova.
- **Temps de servei i velocitat.** Per calcular quantes cel·les pot recórrer per cicle de càrrega.
- **Àrea de desinfecció.** Els graus de la zona que cobreix la llum UV permet saber quines zones s'exposaran a la radiació en cada moment. Com són 360 graus, estigui en el punt de la sala on estigui, afectarà a totes les cel·les adjacents.
- **Sensors.** El robot porta incorporats sensors de moviment i de proximitat, per tal d'aturar-se si detecta que una persona s'hi acosta i per no xocar amb obstacles ni parets.

L'ús que es fa de totes aquestes característiques tècniques s'explica més detalladament en el següent apartat del projecte.

## 3. Aspectes tècnics

### 3.1. Zona de treball

La plataforma on s'ha desenvolupat el projecte és Unity, un motor de videojocs multi plataforma creat per Unity Technologies. Per tal de veure en cada moment el comportament de l'algoritme de control del robot, s'ha usat Unity per crear una interfície gràfica on es pugui observar les sales, les cel·les, el robot i la manera en que aquest interacciona amb l'espai.



Imatge 14. Escena de Unity

Tot i que hi ha alternatives, s'ha escollit Unity perquè facilita molt la creació de la interfície gràfica i és un entorn de desenvolupament complet, que permet provar el codi en cicles de forma senzilla.

Es podria haver fet servir Godot, que és un motor de videojocs que ha aparegut en els últims anys, però no hi ha tanta documentació i no està estandarditzat.

Per altra banda, el codi s'ha desenvolupat amb C#, un llenguatge de programació estandarditzat per Microsoft com a part de la seva plataforma .NET i que és anàleg a Java amb una sintaxis derivada de C.

S'ha utilitzat aquest llenguatge en concret perquè és el que es fa servir per desenvolupar amb Unity.

Si no s'hagués escollit Unity, les altres opcions de llenguatge de programació haurien sigut C++, però representa més hores de feina, els requeriments tècnics són més i Python, que hagués sigut una bona opció si no s'hagués hagut de crear una interfície gràfica.

## 3.2. Bases de l'algoritme

L'objectiu de l'algoritme és moure de manera adequada el robot per tota una sala. Per a fer-ho, cal, en primer lloc, saber on s'ha de moure i, en segon lloc, de quina manera.

La primera part consisteix en la cerca de la següent cel·la a netejar, priorititzant en aquest ordre:

- a) La cel·la horitzontal contigua en la direcció òptima.
- b) La següent cel·la netejable de la mateixa fila en la direcció òptima.
- c) La cel·la immediatament superior.
- d) La següent cel·la netejable de la fila superior en la direcció òptima.

Una cel·la netejable és aquella que és de tipus Floor i per la que no s'hi ha passat cap vegada o que encara està bruta; el criteri serà en funció de la fase en la que s'estigui treballant.

La segona part té dues variants, en funció de si el moviment fins la cel·la és simple i en horitzontal (les dues direccions) o vertical (cap a dalt), o si s'ha de fer varis moviments perquè la cel·la no és contigua.

En el primer cas, la manera en que el robot es mourà no té cap secret; el camí serà únicament la cel·la destí.

En canvi, en el segon cas, el camí ja no és trivial i serà necessari fer servir un algoritme de cerca o *Pathfinding*, de manera que es trobi per quines cel·les ha de moure's el robot per a arribar a la cel·la destí el més eficientment possible. És a dir, s'ha de trobar el camí que representi la distància més curta i el menor nombre de moviments o temps de viatge.

Per aquest projecte he utilitzat, concretament, l'algoritme  $A^*$ , un algoritme de *graph search* que es fa servir quan el mapa consisteix en un *graph*, és a dir, quan està format per nodes i arestes, i quan el camí té un únic origen i un únic destí.

He escollit aquest algoritme perquè, tot i que existeixen d'altres semblants, l' $A^*$  és el millor per aquesta casuística, ja que està optimitzat per una sola destinació. Les altres opcions haguessin pogut ser l'algoritme Dijkstra, que afavoreix els camins menys costosos però els busca per a totes les destinacions, i el Breadth-first search, o cerca en amplada, que no li assigna un cost als camins, sinó que els busca tots per igual i, per tant, la solució final pot no ser la més òptima.

El funcionament de l' $A^*$  consisteix en trobar el millor camí entre un node inicial i un node final partint d'una xarxa de nodes, en aquest cas, una sala. Les direccions en les que es pot moure el robot des d'una cel·la són les quatre principals: amunt, avall, dreta i esquerra; no es pot moure en diagonal perquè podria haver col·lisions amb parets, portes o obstacles.

A cada cel·la se li adjudiquen 3 pesos o costs:

- Cost G. És el cost d'arribar a aquella cel·la des del node inicial. El cost de desplaçar-se en diagonal és de 14, mentre que el de moure's de costat, és de 10.
- Cost H. És el cost heurístic d'arribar fins la cel·la final, obviant obstacles i altres impediments. És la distància entre un node i el node final. En aquest cas, s'ha utilitzat la distància Manhattan, ja que no hi ha moviment en diagonal.

- Cost F. És la suma del cost G i el cost H. És el valor que s'utilitza per prioritzar un node; com més baix sigui el cost F, més probable és que el node estigui a prop del destí.

Per començar, es creen dues llistes, la llista oberta on estan els nodes que s'han de mirar i la llista tancada, on estan els nodes que ja s'han cercat. La cerca acaba quan el node actual és el node final o quan la llista oberta es queda sense nodes i, per tant, no hi ha camí possible.

Partint de la cel·la inicial, que s'afegeix a la llista oberta, primer s'agafa la cel·la amb menor cost F de tota la llista oberta com a cel·la actual. Si no és la cel·la final, s'elimina de la llista oberta i s'afegeix a la llista tancada.

Després, es miren els veïns del node (els que queden a dalt, a baix, a la dreta i a l'esquerra) un per un. Si el veí ja està dins la llista tancada es passa al següent i si és una paret o un obstacle, s'afegeix a la llista tancada. Però si no és cap d'aquests casos, es calcula el seu cost G hipotètic sumant el cost G de la cel·la actual i la distància Manhattan entre la cel·la actual i el veí. Si aquest cost hipotètic és menor al cost G del veí, se li adjudica el cost hipotètic com a cost G, es recalculen els costos H i F i es diu que el veí prové de la cel·la actual. Per últim, si el veí no està dins la llista oberta, se l'afegeix. I així successivament.

Finalment, un cop s'arriba al node final, com cada cel·la del camí coneix de quina cel·la prové, es refà el camí a la inversa.

	42 0 42	38 10 48	42 20 62			
	38 10 48	28 14 42	24 24 48	28 34 62		
	42 20 62	24 24 48	14 28 42	10 38 48	14 48 62	
		28 34 62	10 38 48	A	10 52 62	
			14 48 62	10 52 62	14 56 70	

Imatge 15. Funcionament algoritme A\*

### 3.3. Xarxes de cel·les. Creació i importació

Com s'ha vist a l'apartat anterior, per poder fer servir l'algoritme A\* cal treballar sobre un mapa que es correspongui a un *graph*. Per aquest motiu, la sala està formada per cel·les, que representen els nodes. Les cel·les són quadrades, per tal de reproduir la forma i la mida de la base del robot.

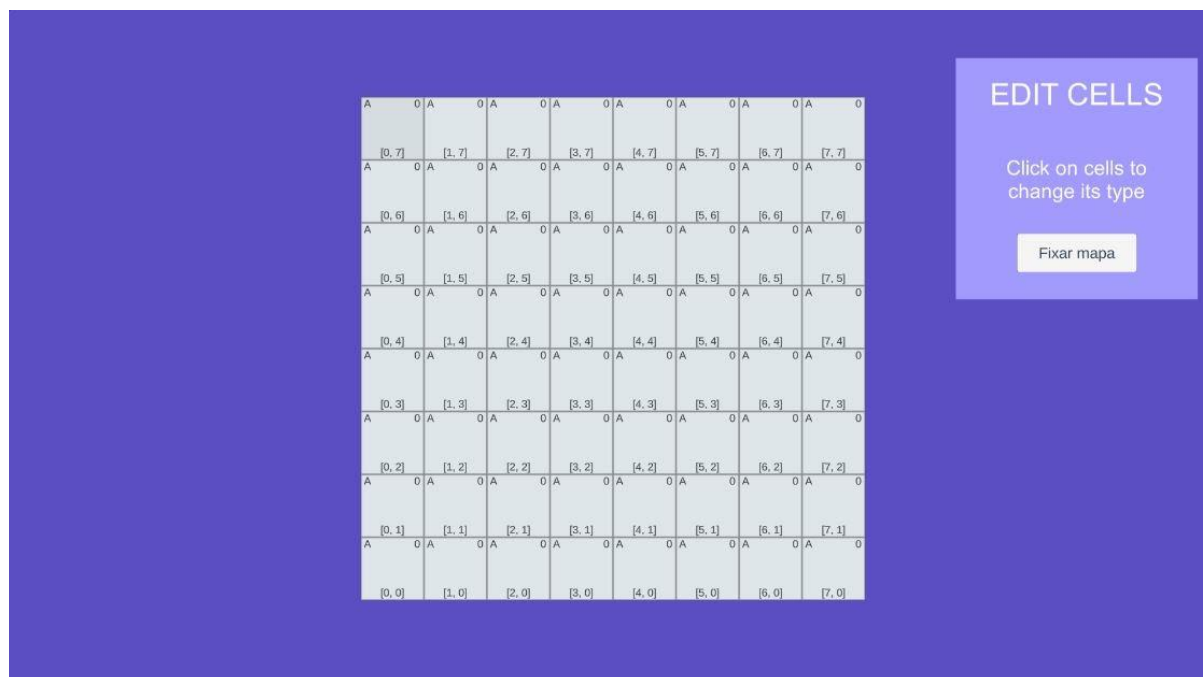
El mapa és una xarxa de cel·les, on cada una d'elles té la mateixa estructura i el mateix comportament, però que prenen valors diferents. Per aquest motiu, les cel·les s'han creat com un Prefab de Unity.

Un Prefab de Unity és un tipus d'asset que permet emmagatzemar un objecte GameObject de manera completa, amb components i propietats. Actua com una plantilla que es pot copiar varies vegades a l'escena. A més a més, qualsevol edició que es faci al Prefab afectarà a totes les seves instàncies, però aquestes també poden ser modificades individualment.

La informació que conté el Prefab Cell són les coordenades X i Y, un booleà que indica si el robot està en aquella cel·la, el nivell de brutícia, el tipus de cel·la, la sala, els costos G, H i F, el comptador de quantes vegades s'hi ha passat, tant el que es mostra com el que s'utilitza pels càlculs i, finalment, de quina cel·la prové, dada que només s'utilitza a l'A\*.

La classe que s'encarrega de crear, importar i editar les xarxes de cel·les és el GridManager. Està creada com a singleton, de manera que la classe que controla el robot RobotManager o qualsevol altre li pugui demanar informació en qualsevol moment, ja sigui sobre la xarxa o sobre les cel·les que la formen.

S'ha creat una interfície per facilitar la interacció amb les sales i el funcionament del robot. És lineal, i permet crear, importar i editar una xarxa i controlar el robot.



Imatge 16. Exemple xarxa de cel·les

Per tenir escenaris ja creats i poder-los importar, s'ha utilitzat Json.

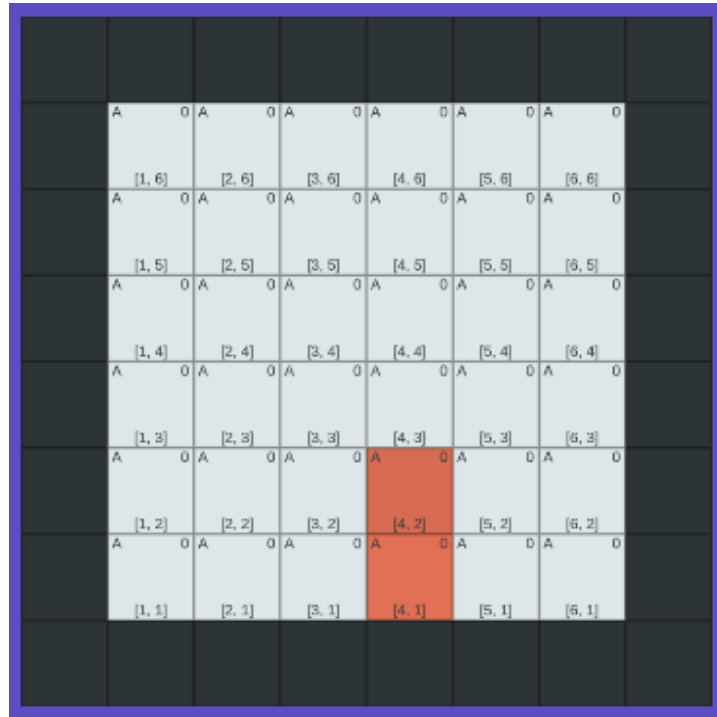
Json és un format lleuger d'intercanvi de dades, independent del llenguatge de programació. La seva estructura està formada per un conjunt de parelles clau/valor, organitzat com una llista ordenada de valors.

Quan es crea una xarxa, algunes de les dades de les cel·les es serialitzen en format Json, de manera que es puguin emmagatzemar con un string. Les dades són les coordenades X i Y, el booleà que indica si el robot està sobre la cel·la, el nivell de brutícia, el tipus de cel·la i la sala.

Per altra banda, quan s'importa una xarxa, el procés és l'invers: el Json es deserialitza. És a dir, es converteix l'estructura de dades Json en instàncies de cel·la que s'ajunten formant una xarxa.







*Imatge 18. Escenari 8x8 amb objecte en vertical*

## 4.2. Objectius

L'algorisme de control del moviment del robot ha de complir aquests objectius per a la fase 1:

- Recórrer la sala passant per les cel·les de tipus Floor, ja que són les úniques per les que es pot caminar, de manera que es passi mínim una vegada per cada una d'elles. D'aquesta manera, s'assegura que s'hagi cobert la sala sencera.
- Poder col·locar el robot en una posició inicial qualsevol de la fila inferior, sempre partint d'una cel·la de tipus Floor, ja que començar per una d'una altra mena no tindria cap sentit. Això permet que si a les cantonades de la fila inferior hi ha obstacles, no sigui un impediment per iniciar el moviment del robot.
- Prohibició del moviment en diagonal. El robot només es pot moure en les quatre direccions principals, és a dir, a les cel·les superiors i inferiors i a les de la dreta i l'esquerra. La decisió de no fer vàlid el moviment diagonal és deguda a que podria provocar la col·lisió del robot amb parets, portes o obstacles. Tot i que això significa que per rodejar un obstacle s'ha de fer més moviments, és un comportament més segur i més escalable, ja que no s'haurà de tenir en compte una possible col·lisió a l'hora de dissenyar jocs de proves i permetrà assegurar un bon funcionament en qualsevol escenari.
- Ús de l'A\* quan no es pot fer un moviment senzill, és a dir, moure's a una cel·la contigua. D'aquesta forma, sempre es trobarà el camí òptim per rodejar un obstacle o per fer un canvi de fila.

### 4.3. Desenvolupament

La manera en que l'algoritme busca el camí, és a dir, la llista de cel·les, per les que ha de passar per netejar la sala sencera complint els objectius, està programada en la funció `searchPath`, pròpia de la classe `RobotManager`.

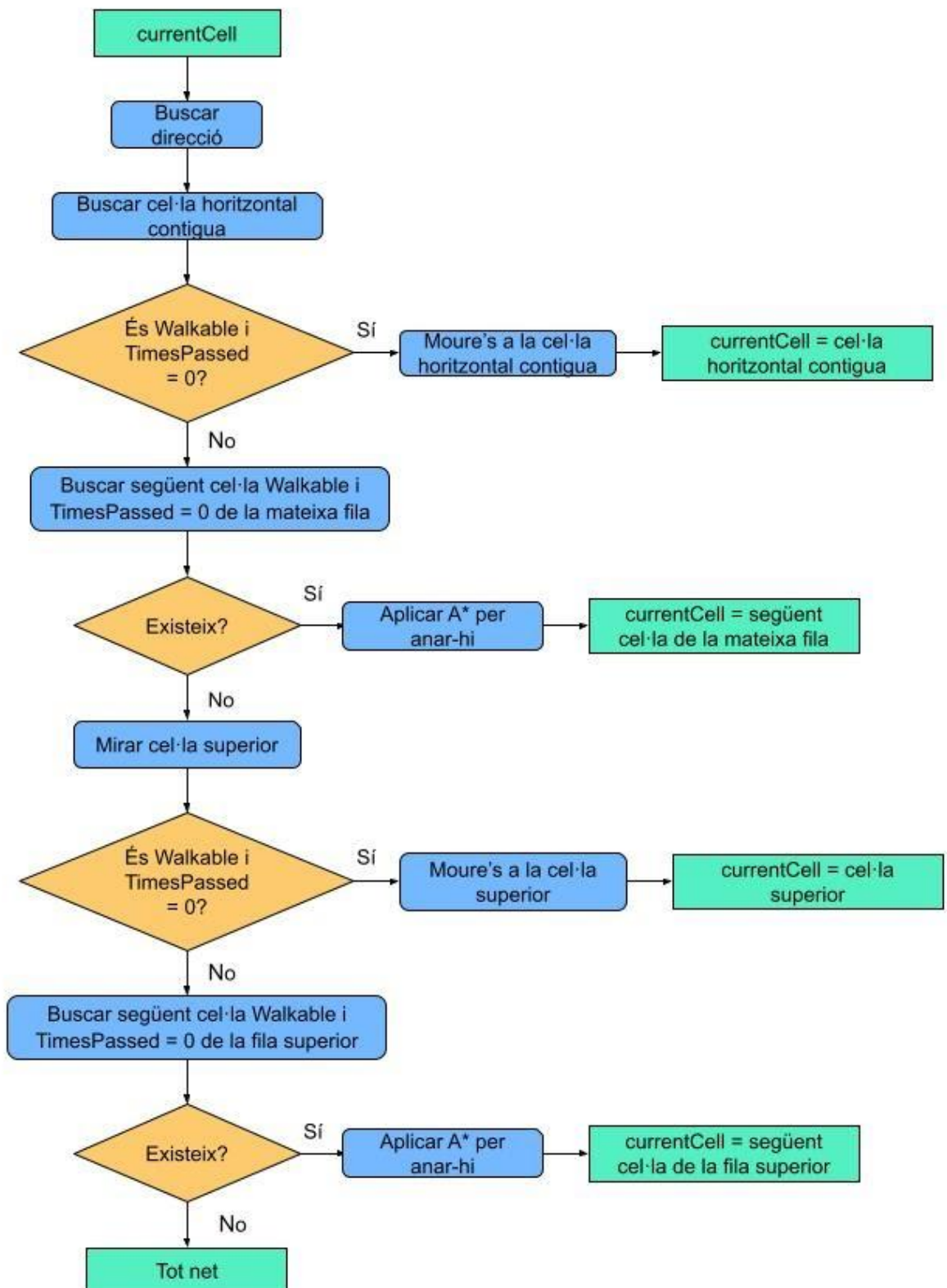
El primer que fa és buscar quina és la millor direcció amb la que recórrer la fila en la que es troba el robot, per tal de començar a moure's cap a on menys cel·les sense netejar hi hagi.

Després d'això, es busca en aquesta direcció, la cel·la en horitzontal contigua. Si és "Walkable", és a dir, no és de tipus `Wall` ni de tipus `Obstacle`, i no s'hi ha passat encara, és a dir, `TimesPassed = 0`, el robot afegeix la cel·la a la llista de cel·les que conformen el camí a seguir.

Si la cel·la en horitzontal no compleix cap d'aquestes condicions o és nul·la (es surt de la xarxa), l'algoritme busca la cel·la de la mateixa fila a la que sí es pot moure. Un cop la té, calcula amb l'algoritme  $A^*$  el camí a fer per anar-hi.

Un cop ja no queden més cel·les a netejar o a les que es pot moure en la fila, el robot es mou a la fila superior. Per fer-ho, l'algoritme mira si la cel·la just sobre la que està el robot és `Walkable` i no és nul·la. Si es compleix, l'afegeix al camí. Sinó, busca la cel·la de la fila superior a la que sí es pot moure i calcula amb  $A^*$  el camí per arribar-hi.

Un cop l'algoritme ha calculat tot el camí que el robot ha de recórrer per netejar la sala sencera, s'inicia la corutina de moviment, que és la que realment fa moure al robot per totes les cel·les que conformen el camí.



Imatge 19. Diagrama de flux de la fase 1

## 4.4. Resultats i avaluació

Per avaluar el comportament de l'algoritme de control del robot i si els objectius de la fase s'han satisfet, s'utilitza el paràmetre de rendiment següent:

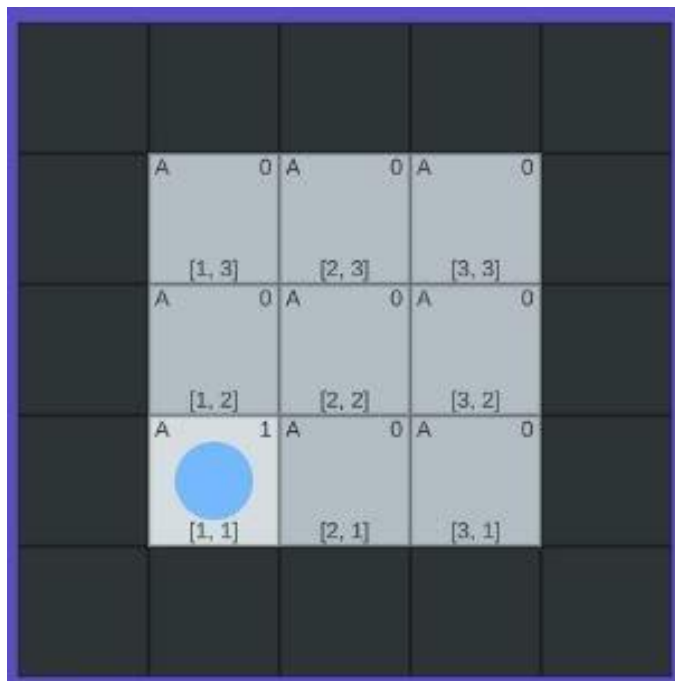
$$\text{Rendiment} = \frac{\# \text{ de passos}}{\# \text{ de cel·les Floor}}$$

El valor mínim del rendiment és la unitat, ja que els objectius dicten que el robot passi mínim una vegada per cada cel·la de tipus Floor per a que la sala quedi completament neta.

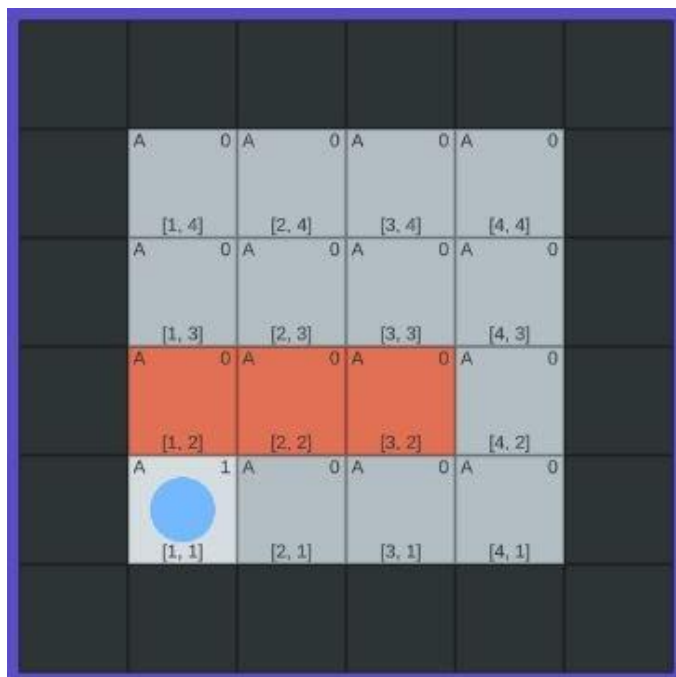
Per altra banda, un valor de rendiment dolent serà aquell major a 2, ja que encara que s'hagi de passar per una mateixa cel·la més d'una vegada a l'hora de rodejar un obstacle, la majoria de cel·les no es veuràn afectades i, per tant, es compensarà.

Els resultats a avaluar són el comportament de l'algoritme de neteja en 4 sales diferents, comparant si és millor fer servir `searchDirection()` per decidir en quina direcció avançar, donada la cel·la on es troba el robot, abans de calcular el seu següent moviment o si no és necessari.

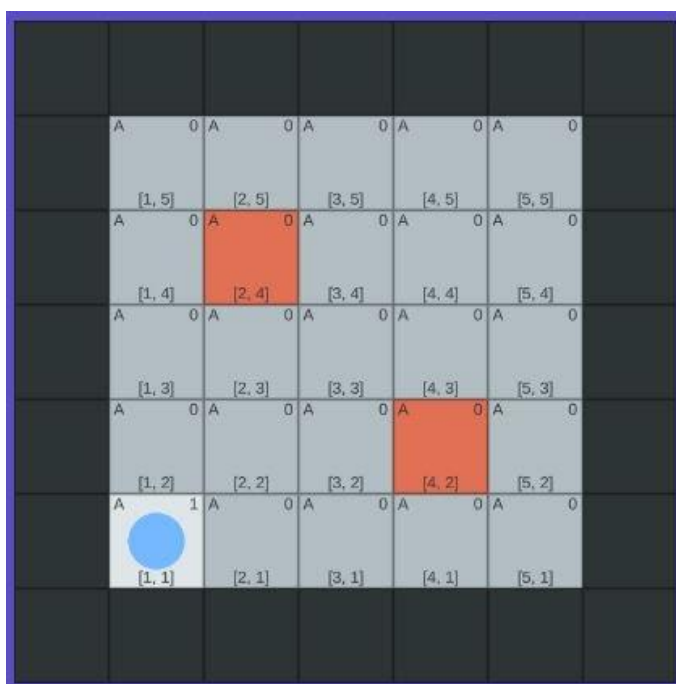
Els escenaris estudiats són:



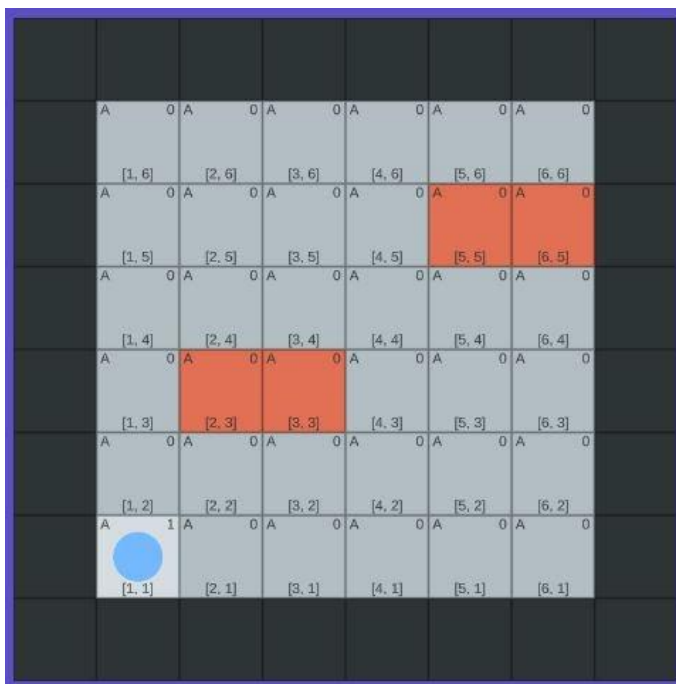
Imatge 20. Escenari 5x5 bàsic EF1.1



Imatge 21. Escenari 6x6 amb canvi de fila limitat EF1.2



Imatge 22. Escenari 7x7 amb obstacles separats EF1.3



Imatge 23. Escenari 8x8 amb obstacles junts EF1.4

Els resultats i el valor del rendiment són:

Escenari	searchDirection	# de passos	# de cel·les	Rendiment
EF1.1	Si	9	9	1
EF1.2	Si	13	13	1
EF1.3	Si	29	23	1,26
EF1.4	Si	38	32	1,19
EF1.1	No	9	9	1
EF1.2	No	13	13	1
EF1.3	No	29	23	1,26
EF1.4	No	39	32	1,22

Taula 2. Resultats fase 1

Com es pot observar, no hi ha gaire diferència entre els resultats amb i sense buscar la direcció òptima. Sí que en l'últim escenari hi ha una lleugera millora quan es fa servir el searchDirection, però és mínima.

Això és degut a la simplicitat dels jocs de proves de la fase 1, ja que encara que hi hagi obstacles, aquests no modifiquen en gran manera com es comporta l'algoritme a l'hora de canviar de fila.

No obstant, s'ha seguit treballant amb la funció searchDirection i s'ha vist que és altament necessària quan les sales tenen parets internes, passadissos i més obstacles.

## 5. Fase 2

### 5.1. Casuística

Les característiques de les zones de prova de la fase 2 són les següents:

- **Més d'una zona**, que tant pot ser un passadís com una sala, **fins un límit de 3 zones**. La primera zona o sala A sempre contindrà la cel·la inicial [1,1] i sempre es començarà a netejar des d'ella. Un cop la sala A hagi quedat completament neta, es passarà, si existeix, a la sala B. Quan la segona sala també s'hagi acabat de netejar, es passarà, si existeix, a la sala C. L'activitat del robot no finalitza fins que totes les sales no estiguin netes.
- La **bateria del robot** es té en compte a l'hora de calcular el camí a realitzar per netejar una sala. A partir de les característiques tècniques del UVD Robot® Model B, i tenint en compte que les cel·les fan 100 cm x 100 cm, la quantitat de cel·les que es podrien recórrer en un cicle de càrrega, és:

$$2 \text{ h de feina continua} * \frac{2,7 \text{ km}}{\text{h}} \text{ de velocitat mitja} * \frac{10^4 \text{ cm}}{1 \text{ km}} * \frac{1 \text{ cel·la}}{100 \text{ cm}} = 540 \text{ cel·les per cicle de càrrega}$$

Degut a que l'espai de treball és molt més reduït, la bateria màxima és de 32 cel·les. Considerant que quan resti un 25% de la bateria total cal moure el robot a una cel·la de càrrega, quan el comptador de la bateria arribi a 8 cel·les, es considerarà que la bateria està baixa.

- **Una o varies cel·les de càrrega**, repartides per les diferents sales. El robot pot passar-hi, ja sigui com a part del seu camí o per anar a carregar-se. Quan el robot s'hi col·loqui a sobre, es considerarà que la bateria es recarrega completament.
- Com en la fase 1, hi pot haver obstacles repartits per les sales, ubicats de diverses maneres, però mai obstaculitzant el pas completament de forma que s'eviti poder arribar a la següent sala.  
Per altra banda, ara sí pot haver-hi cel·les de tipus Wall dins la sala, de manera que delimitin les diferents zones.

### 5.2. Objectius

Els objectius que ha d'assolir l'algoritme de control del robot a la fase 2 són:

- Recórrer totes les sales passant per les cel·les de tipus Floor, de manera que es passi mínim una vegada per cada una de les cel·les. Això farà que totes les sales quedin completament netes i, per tant, també el conjunt.
- Netejar les diferents zones en ordre, tal que la primera sala a netejar sigui l'A, després la B i, finalment, la C. Això serà així sempre i quan existeixen 3 sales; en cas que n'hi hagués menys, l'ordre seria el mateix però hi hauria menys sales a completar.
- Identificar la cel·la inicial de cada sala, per tal de començar a recorre-la des d'allà. D'aquesta manera s'assegura que la neteja sigui contínua, ja que un cop s'acaba de netejar una sala, el següent moviment serà moure's a la cel·la inicial de la zona següent; amb això s'evita que hi pugui haver errors. La cel·la inicial d'una sala sempre serà aquella que es trobi més a prop de l'origen de coordenades [0,0].

- Calcular quanta bateria té el robot en cada moment per tal de saber quan cal anar a una cel·la de càrrega, ja que un cop es detecti que el robot té menys de 8 cel·les de càrrega, es considerarà que està baix de bateria. Quan el robot es mogui a una base de càrrega, la bateria tornarà al seu valor màxim de 32 cel·les.
- No permetre que el robot es quedi sense bateria en un lloc que no sigui una cel·la de càrrega, ja que sinó la rutina de neteja quedaria aturada i caldria intervenció humana.

### 5.3. Desenvolupament

Les tres diferències principals entre el funcionament de l'algoritme de neteja de la fase 1 respecte el de la fase 2 són, en primer lloc, la condició que dona per acabat el procés de desinfecció, en segon lloc, la rutina de càrrega i, en tercer lloc, el canvi d'una sala a una altra.

En la fase 1, la neteja finalitzava quan no es trobava cap cel·la de tipus Floor per la que no s'hagués passat cap cop (`TimesPassed = 0`); d'avaluar aquesta condició s'encarrega la funció `FinishedCleaning()`.

No obstant, ara, a l'haver més sales, es necessita una altra condició que indiqui que ha finalitzat la neteja d'una sala concreta. D'aquesta manera, es calcularà el camí total, el del conjunt de totes les sales, abans de començar la rutina de moviment, és a dir, no es netejarà una sala fins que l'anterior no estigui completament neta. La funció `RoomCleaned()` avalua aquesta condició.

A més a més, ara la funció de neteja que s'executa al donar Play està partida en dues parts: la funció de neteja global, que està en funcionament mentre totes les sales no estan netes, i la funció de neteja d'una sala concreta, que s'encarrega d'executar el codi de neteja d'una sala.

Una altra diferència amb l'algoritme de neteja de la fase 1 és el control de la bateria. Abans de decidir el següent moviment del robot, es comprova que la bateria actual sigui menor o igual al valor de bateria baixa. En cas que ho sigui, es busca la cel·la de càrrega més propera (utilitzant la distància Manhattan entre la cel·la actual i totes les cel·les de càrrega existents) i es calcula el camí amb A\* necessari per moure's-hi. Un cop el robot es col·loca sobre la cel·la de tipus Charge, la bateria es restableix al seu valor màxim.

Finalment, es busca el camí amb A\* de tornada a la cel·la de la que s'havia partit. No obstant, el camí es retalla en funció de si les cel·les més properes a l'inicial ja estan netes, ja que no tindria cap sentit tornar-hi a passar. Això sí, si hi ha una cel·la en una fila inferior que no s'ha netejat encara, la cel·la a la que es tornarà no pot ser d'una fila superior, ja que el moviment vertical del robot només pot ser positiu, és a dir, cap a files superiors.

Si la bateria no és inferior o igual al valor de bateria baixa, es calcula el següent moviment amb `searchPath()`.

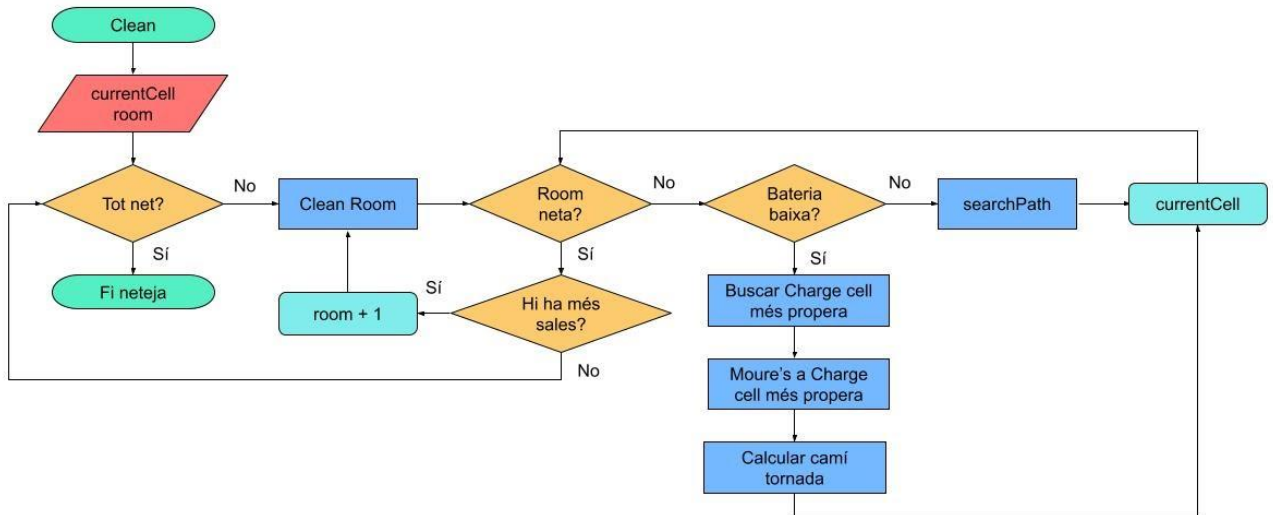
Per altra banda, cada cop que es decideix a quina cel·la cal moure's i de quina manera amb la funció `searchPath()`, les cel·les del camí calculat s'afegeixen al camí global. Al fer



això, es calcula quantes cel·les s'afegeixen i el comptador de la bateria es redueix en el mateix nombre. D'aquesta manera, la bateria sempre està actualitzada.

L'última diferència és el canvi de sala. Un cop es detecta que la sala actual ha quedat completament neta, es busca la cel·la inicial de la següent i es calcula el camí amb A\* per moure's-hi.

En cas que no hi hagués següent sala, voldria dir que ja s'ha finalitzat la neteja de totes les zones.



Imatge 24. Diagrama de flux de la fase 2

## 5.4. Resultats i avaluació

L'avaluació del comportament de l'algorisme de neteja de las fase 2 té dues parts. Per una banda, es calcula el paràmetre rendiment descrit a l'avaluació de la fase 1, ja que és el valor principal a tenir en compte. Per l'altra banda, s'avalua el paràmetre rendiment de càrrega, calculat de la següent forma:

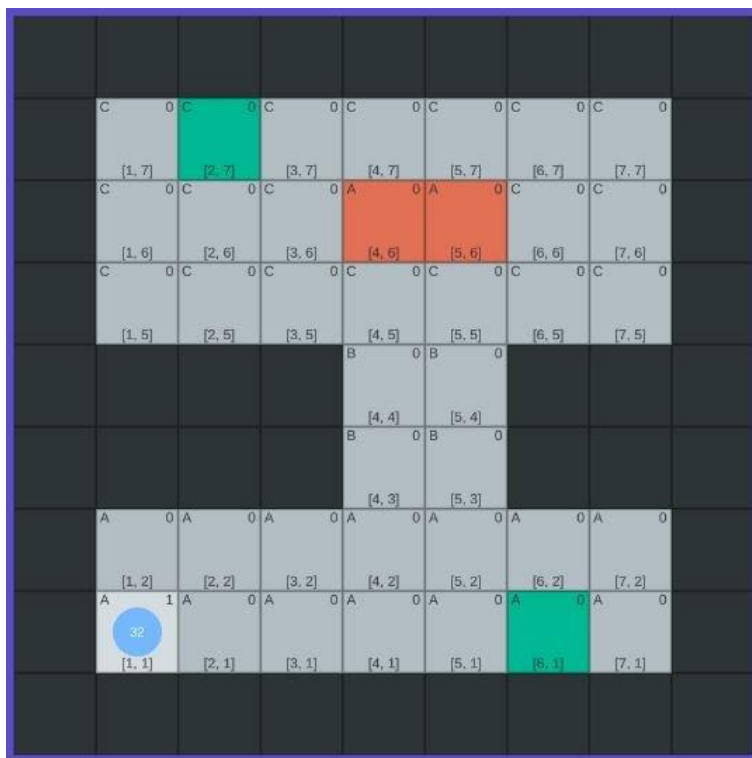
$$\text{Rendiment de càrrega} = \frac{\# \text{ de càrregues}}{\# \text{ de cel·les Charge}}$$

Tenint en compte que el nombre de cel·les que es poden recórrer per cicle de càrrega és 32, els escenaris de prova contindran una cel·la de càrrega per cada 32 cel·les de tipus Floor. D'aquesta manera, hi ha més flexibilitat a l'hora de trobar la cel·la de tipus Charge més propera quan calgui carregar el robot.

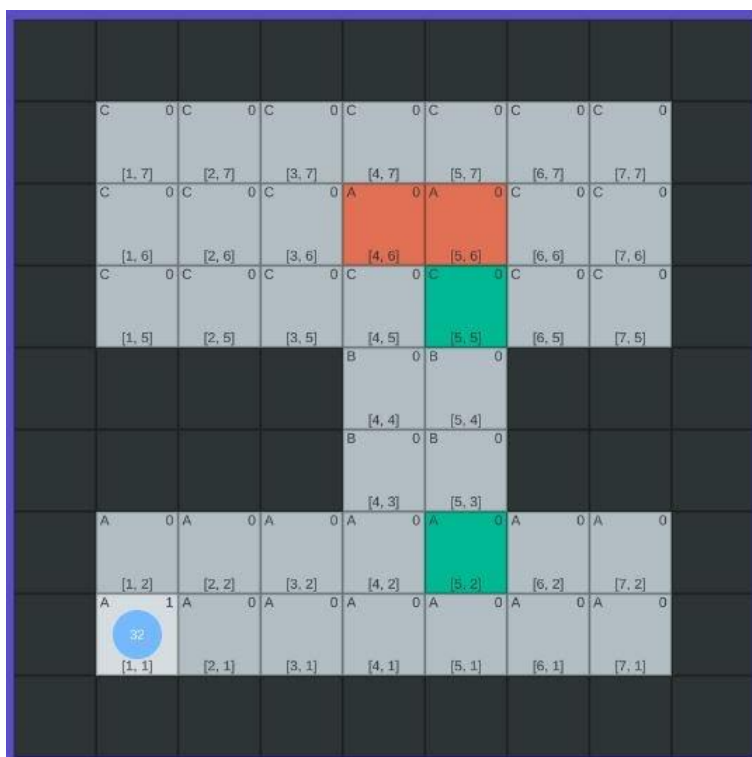
El valor òptim per al rendiment de càrrega és la unitat, ja que si les cel·les de càrrega estiguessin posades en la direcció de neteja cada 32 cel·les, seria el nombre de càrregues que realitzaria el robot fins a netejar completament una sala.

Els resultats que s'avaluen són els obtinguts en comparar el comportament de l'algorisme de neteja en el cas que les cel·les de càrrega estiguin a prop o lluny les unes de les altres, i en el cas que el passadís o sala B estigui en un lateral o en el centre.

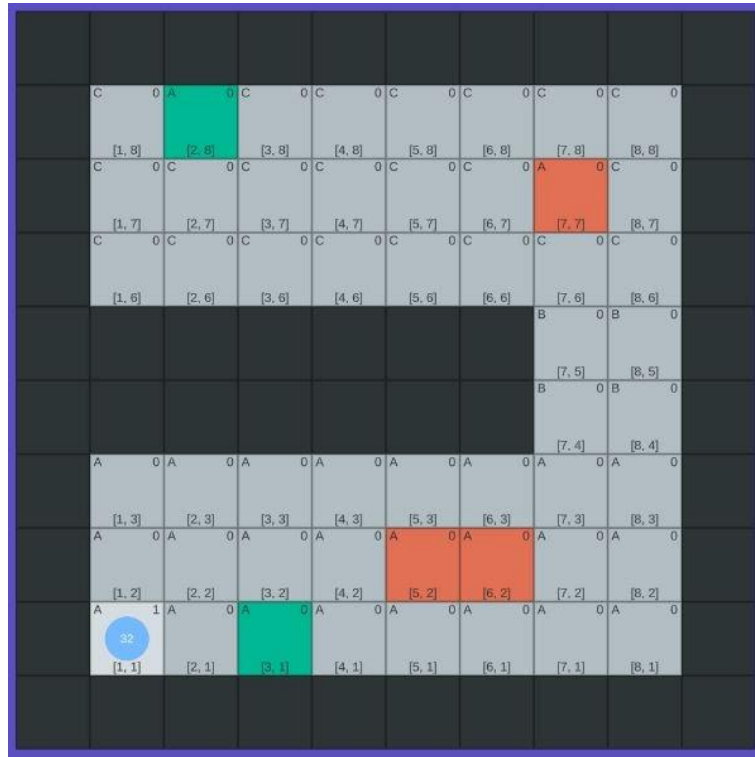
Els escenaris de proves estudiats són:



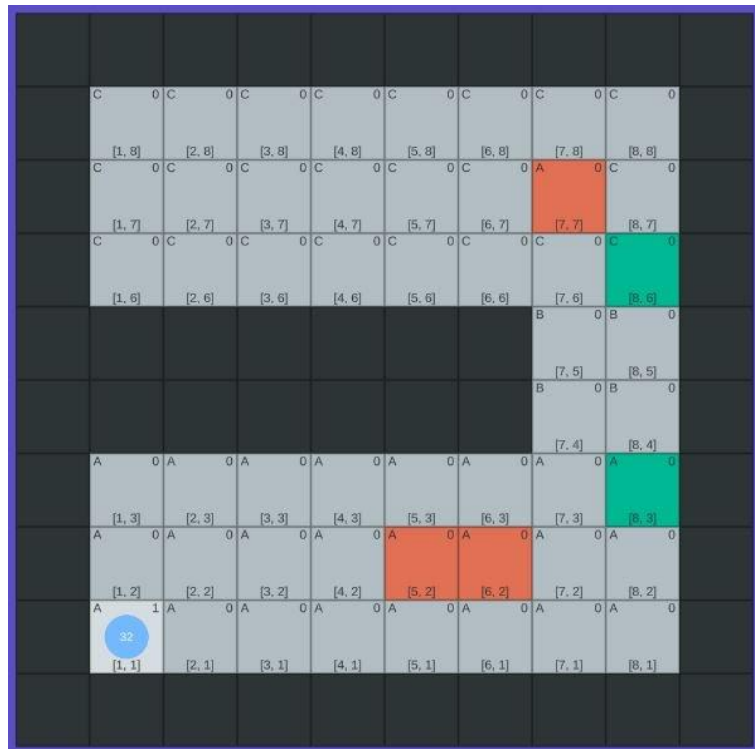
Imatge 25. Escenari 9x9 passadís central amb cel·les de càrrega llunyanes EF2.1



Imatge 26. Escenari 9x9 passadís central amb cel·les de càrrega properes EF2.2



Imatge 27. Escenari 10x10 passadís lateral amb cel·les de càrrega llunyanes EF2.3



Imatge 28. Escenari 10x10 passadís lateral amb cel·les de càrrega properes EF2.4

Els resultats i els valors del rendiment són:

Escenari	Passadís	Charge Cells	# de passos	# de cel·les Floor	Rendiment
EF2.1	Central	Llunyanes	56	35	1,6
EF2.2	Central	Properes	44	35	1,26
EF2.3	Lateral	Llunyanes	97	64	1,52
EF2.4	Lateral	Properes	73	64	1,14

*Taula 3. Resultats rendiment fase 2*

Escenari	Passadís	Charge Cells	# de càrregues	# de cel·les Charge	Rendiment de càrrega
EF2.1	Central	Llunyanes	3	2	1,5
EF2.2	Central	Properes	3	2	1,5
EF2.3	Lateral	Llunyanes	3	2	1,5
EF2.4	Lateral	Properes	2	2	1

*Taula 4. Resultats rendiment de càrrega fase 2*

Observant els resultats, es pot concloure que el paràmetre rendiment és major al de la fase 1. Això ve donat pel fet que hi ha més moviments que poden provocar que el robot passi més vegades de les necessàries per una mateixa cel·la. Per exemple, les anades i tornades a les cel·les de càrrega o el camí a recórrer fins la cel·la inicial de la sala següent.

Tot i així, el valor màxim del rendiment no arriba a ser massa alt i no s'apropa al màxim permès. És a dir, el comportament de l'algoritme segueix sent adequat.

Per altra banda, els valors del rendiment de càrrega són pràcticament estables en 1,5. És un resultat positiu, ja que vol dir que el nombre de càrregues és quasi independent de la distribució de les sales i de les cel·les de càrrega. En l'últim escenari, però, el rendiment de càrrega és 1, és a dir, el valor òptim.

## 6. Fase 3

### 6.1. Casuística

La casuística de la fase 3 és la mateixa que la de la fase 2 afegint:

- Les cel·les de tipus Floor tenen un nivell de brutícia, que pot ser 0, 1 o 2. El nivell 0 és el valor per a una cel·la completament neta. El nivell correspon a quantes vegades ha de passar el robot per la cel·la per a que quedi neta.
- Inicialment, totes les cel·les Floor tenen un nivell de brutícia diferent a 0, és a dir, no n'hi ha cap de neta.
- La brutícia no segueix cap patró. Els valors de cada cel·la són adjudicats a l'atzar a l'hora de crear les sales.
- Les cel·les dels altres tipus, ja sigui Wall, Obstacle o Charge, no tenen nivell de brutícia.

### 6.2. Objectius

Els objectius de la fase 3 són:

- Passar per totes les cel·les de tipus Floor tantes vegades com el nivell de brutícia que tinguin, de manera que el nivell final de cada una d'elles sigui 0. D'aquesta manera, les sales quedaran completament netes.
- Quan el següent moviment del robot sigui a una cel·la amb nivell de brutícia 2, romandre-hi el doble de temps fins deixar-la neta. Amb això s'estalviarà moviments.
- Quan el següent moviment del robot formi part del camí fins la cel·la inicial de la següent sala o del recorregut fins una cel·la de càrrega, no romandre a la cel·la fins tenir un nivell de brutícia 0, sinó quedar-se només una vegada.
- Netejar completament una sala abans de passar a la següent. És a dir, canviar de zona només quan totes les cel·les tinguin nivell de brutícia 0.

### 6.3. Desenvolupament

La integració del nivell de brutícia al comportament de l'algoritme ha tingut dues parts importants, l'atribut dirtLevel de la classe cel·la i el canvi de la condició de neteja completa.

En primer lloc, la cel·la té un indicador del nivell de brutícia, que és fix i s'adjudica al crear la xarxa de cel·les. És a dir, ara la cel·la té dos indicadors, el del nivell de brutícia, amb valors entre 0 i 2, i el de quantes vegades hi ha passat el robot, que no té valor màxim.

En segon lloc, mentre que en les fases 1 i 2 la condició de finalització del procés era que totes les cel·les Floor tinguessin TimesPassed > 0, ara la condició és que s'hagi passat més vegades per la cel·la que el seu nivell de brutícia, això és, TimesPassed >= dirtLevel.

Si aquesta condició es compleix per totes les cel·les de tipus Floor, es considerarà que les sales han quedat netes.

Per a veure-ho de manera gràfica, a cada nivell de brutícia se li ha adjudicat un to de gris diferent, de forma que es pugui veure de manera actualitzada quin és el valor de brutícia de la cel·la en funció de quantes vegades hi ha passat el robot.



Imatge 29. Colors per a cada nivell de brutícia

Per a fer això, el color s'escull en funció del següent càlcul:

$$\text{Max}(0, \text{dirtLevel} - \text{TimesPassed})$$

## 6.4. Resultats i avaluació

El comportament de l'algoritme de control del robot durant la fase 3 s'avalua amb el paràmetre de rendiment següent:

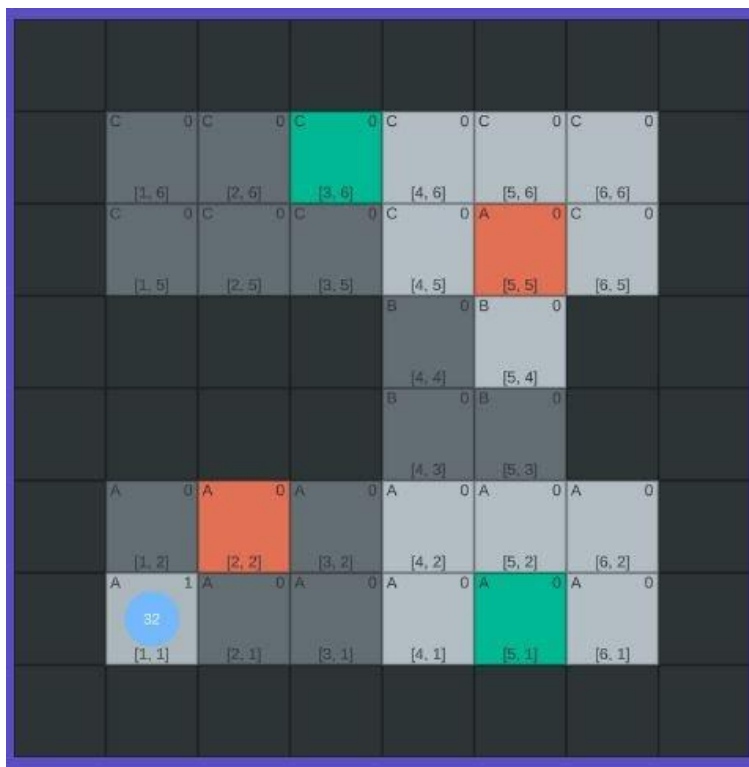
$$\text{Rendiment} = \frac{\# \text{ de passos}}{\# \text{ de brutícia total}}$$

Com es pot veure, és molt semblant al rendiment usat a la fase 1, però amb la diferència que es té en compte la suma dels nivells de brutícia de totes les cel·les Floor. Això és degut a que ara no s'ha de passar només una vegada per cada cel·la per netejar-la, sinó que, en alguns casos, s'hi ha de passar dos cops.

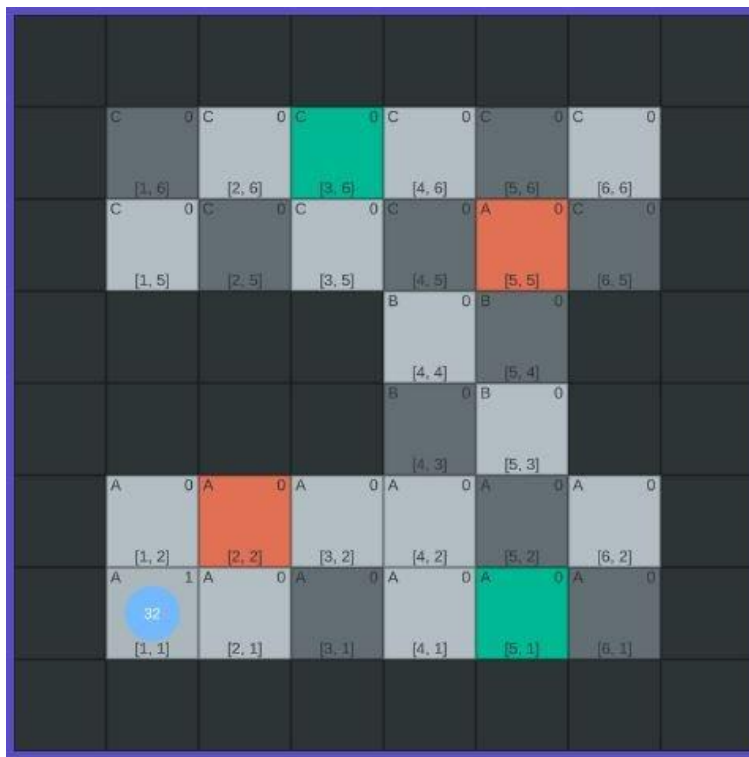
D'aquesta manera també s'aconsegueix que els valors òptims i no acceptats del rendiment siguin els mateixos; rendiment unitat com a millor valor possible i rendiment 2 com a valor màxim permès.

En aquesta fase s'avaluen els resultats obtinguts de comparar, en dos escenaris diferents, la influència de la distribució de la brutícia, en funció de si està concentrada en petites zones o escampada per les sales.

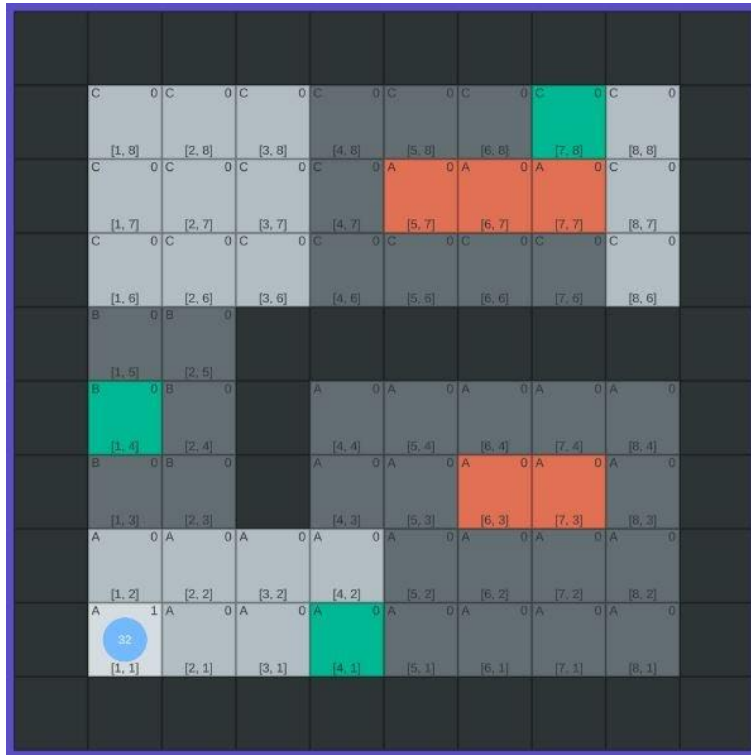
S'han estudiat els següents escenaris de prova:



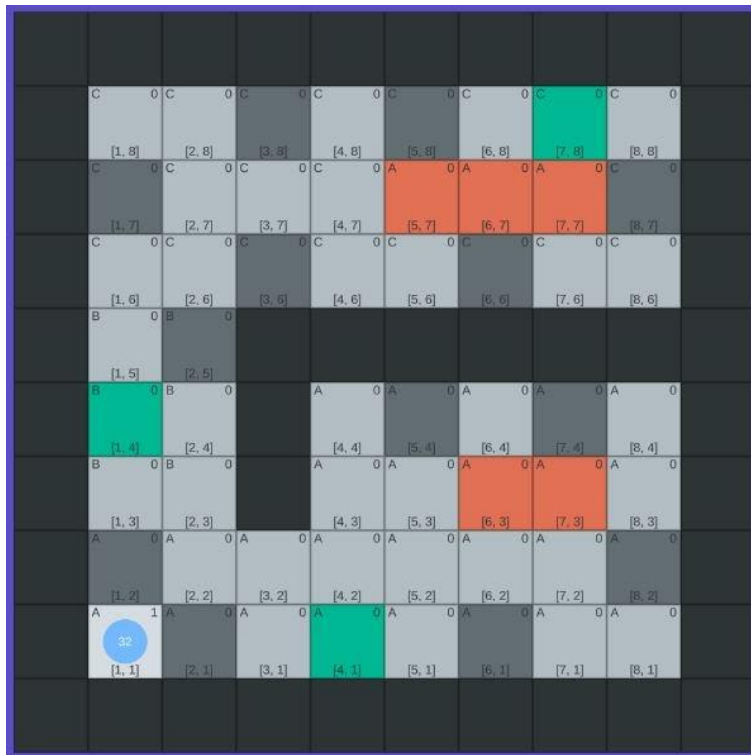
Imatge 30. Escenari 8x8 amb brutícia concentrada EF3.1



Imatge 31. Escenari 8x8 amb brutícia escampada EF3.2



Imatge 32. Escenari 10x10 amb brutícia concentrada EF3.3



Imatge 33. Escenari 10x10 amb brutícia escampada EF3.4



Els resultats i el rendiment obtinguts són:

Escenari	Brutícia	# de passos	# de brutícia	Rendiment
EF3.1	Concentrada	51	37	1,38
EF3.2	Escampada	50	35	1,43
EF3.3	Concentrada	100	77	1,30
EF3.4	Escampada	99	61	1,62

*Taula 5. Resultats fase 3*

Com es pot observar, els valors del rendiment estan dins dels permesos, al voltant de 1,5.

També es pot veure que el rendiment és lleugerament menor quan la brutícia està concentrada, que quan està escampada.

Per altra banda, en els escenaris 8x8 hi ha menys diferència entre els valors de rendiment en funció de la brutícia que en els escenaris 10x10. Per tant, sembla que la ubicació de la brutícia té més influència en jocs de proves més grans.

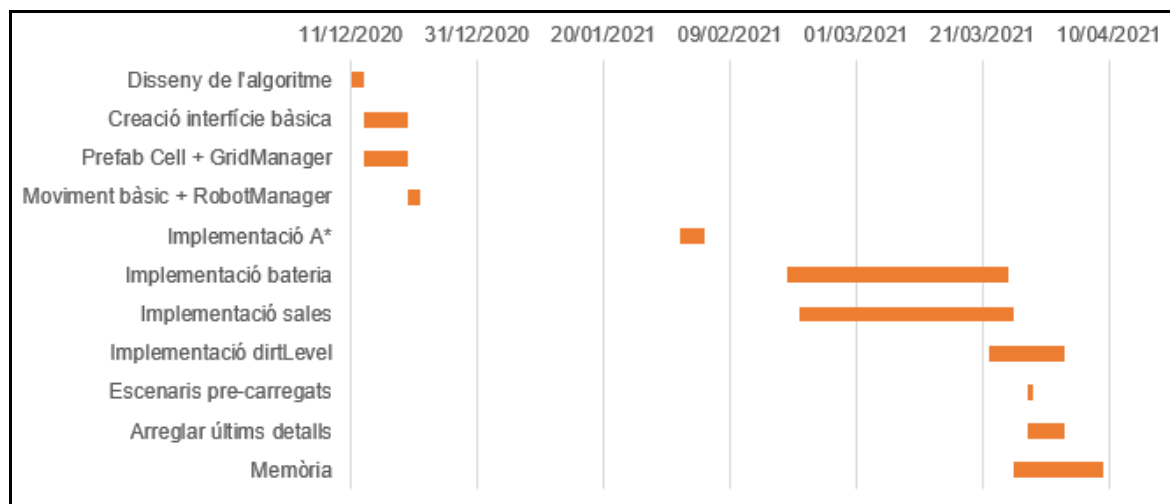
## 7. Planificació temporal

El projecte es va iniciar al desembre del 2020 i s'ha finalitzat a l'abril del 2021. En total, han calgut 360 hores de feina.

La planificació ha sigut la següent:

Activitat	Data inici	Duració en dies	Data fi
Disseny de l'algoritme	11/12/2020	2	13/12/2020
Creació interfície bàsica	13/12/2020	7	20/12/2020
Prefab Cell + GridManager	13/12/2020	7	20/12/2020
Moviment bàsic + RobotManager	20/12/2020	2	22/12/2020
Implementació A*	01/02/2021	4	05/02/2021
Implementació bateria	18/02/2021	35	23/03/2021
Implementació sales	20/02/2021	34	24/03/2021
Implementació dirtLevel	22/03/2021	12	03/04/2021
Escenaris pre-carregats	28/03/2021	1	29/03/2021
Arreglar últims detalls	28/03/2021	6	03/04/2021
Memòria	26/03/2021	14	09/04/2021

Taula 6. Planificació projecte



Imatge 34. Diagrama de Gantt del projecte

## 8. Estudi econòmic

Degut a que el projecte és només el disseny de l'algorisme i la creació d'una demostració gràfica del seu comportament, el cost és molt reduït.

A més a més, els dos programes que he fet servir per portar-lo a terme, Unity i Visual Studio Code, són gratuïts.

Si fas servir Unity a nivell professional cal pagar-lo, però no és el cas.

Per tant, doncs, el pressupost és el següent:

Concepte	Quantitat	Preu	Cost
Salari	360 hores	25 €/hora	9.000,00 €
Unity	-	Gratuït	0,00 €
Visual Studio Code	-	Gratuït	0,00 €
TOTAL			9.000,00 €

*Taula 7. Pressupost detallat*

Probablement, un estudi de disseny com el que he realitzat, a nivell professional i fet per persones amb més experiència, podria valorar-se més alt, ja que seria més extens, més acurat i més fiable.

## 9. Estudi ambiental

L'impacte ambiental del projecte té dues parts: en primer lloc, l'efecte de les hores de feina del projecte en si, i, en segon lloc, les conseqüències que pugui tenir l'ús d'un robot UV-C.

La feina realitzada durant el treball s'ha portat a terme utilitzant un ordinador portàtil, pel que hi ha hagut consum elèctric i, per tant, emissions de CO<sub>2</sub>. El paràmetre de conversió entre gCO<sub>2</sub> i KWh s'anomena mix elèctric, i el seu valor pel 2019 segons l'Oficina Catalana del Canvi Climàtic és de 241 gCO<sub>2</sub>/KWh. Un ordinador portàtil consumeix 0,11 KWh a l'hora, que equival a 26,51 gCO<sub>2</sub>. Per les 360 hores de feina, el total d'emissions seria 9543,6 gCO<sub>2</sub> per tot el projecte.

Els mètodes de neteja i desinfecció tradicionals fan un ús molt gran de materials i productes tòxics pel medi ambient, com el lleixiu i altres químics, i d'un sol ús o no reciclables, com les baietes i altres elements. Per tant, tenen molt impacte ambiental negatiu. A més a més, al ser productes barats, es produeixen en massa i no hi ha consciència de l'ús que se'n fa.

En canvi, els robot UV-C, al tenir una intel·ligència programada per fer una neteja molt eficient, ja està pensat per consumir el mínim imprescindible. A més a més, els robots no creen residus durant el seu funcionament i l'únic consum que tenen és el de carregar-se.

Per altra banda, el temps de vida de les làmpades UV-C és de 12000 hores, és a dir, 500 dies. Per tant, el robot no necessitarà canviar les llums durant molt de temps i els residus seran molt menors.

Un cop una làmpada UV-C deixa de funcionar, s'ha de reciclar igual que un llum fluorescent, ja que estan fabricades amb els mateixos elements. La majoria de centres sanitaris o centres comercials tenen programes de reciclatge de làmpades fluorescents, pel que també estan preparats per les d'UV-C.

El principal problema ecològic és que les làmpades necessiten mercuri per funcionar, i aquest element és altament tòxic i perillós per a les persones i altres éssers vius. La preocupació més important és que es pogués escapar mercuri de la làmpada i contaminar el seu entorn, pel que l'Agència de Protecció Ambiental d'Estats Units (EPA) el considera un residu perillós.

Els fabricants de robots UV-C UVD Robots®, dels quals jo he agafat el model i les característiques tècniques, van resultar guanyadors del premi a la solució més sostenible dins el sector sanitari, entregat per l'organització danesa Sustainary **[2]**.

En conclusió, tot i que la presència de mercuri dins les làmpades pot suposar un perill, els beneficis ambientals que aporta l'ús de robots UV-C són majors i, per tant, és millor opció.

## Conclusions

L'objectiu del projecte era crear un algoritme de control d'un robot de neteja UV-C. A mesura que s'ha anat avançant al llarg de les fases, l'algoritme ha anat guanyant complexitat, per tal de poder satisfer els requeriments.

A partir de l'avaluació de les diferents fases, es pot veure que l'algoritme no és completament òptim, però que ho és en gran part. És a dir, el valor del rendiment no sempre és el mínim, però la sobrecàrrega no és alta. Els casos on té millor comportament són en la fase 1, degut a la simplicitats dels escenaris, en la fase 2, quan el passadís que comunica les sales està en un lateral i les cel·les de càrrega estan properes, i en la fase 3, quan la brutícia està concentrada en zones concretes de la sala.

L'ús de l'algoritme A\* només quan el robot s'havia de moure a una cel·la no contigua, enlloc d'utilitzar-lo per a qualsevol moviment, ha estalviat molts recursos de computació, ja que tot i que es podria haver fet servir, hagués sigut una manera innecessàriament complexa de calcular el camí.

En canvi, diferenciar els dos tipus de moviment ha permès simplificar l'algoritme.

Per altra banda, també s'ha arribat a la conclusió que fer servir Unity ha sigut una bona decisió, ja que ha permès estructurar la interfície de forma senzilla utilitzant Prefabs.

El que és més, gran part del projecte es basa en poder veure gràficament el comportament del robot en els diversos escenaris, pel que tenir una bona eina per fer-ho era fonamental.

Finalment, tot i que l'algoritme final satisfà tots els objectius proposats, hi ha algunes coses que es podrien millorar per a fer-lo encara més òptim i eficient, és a dir, tenir uns rendiments més propers al valor mínim.

Aquesta conclusió queda detallada en l'apartat de visió de futur.

## Visió de futur

Els següents passos que m'hagués agradat donar amb el projecte són aquells que permetessin expandir l'abast inicial.

En primer lloc, m'agradaria poder tenir en compte el temps. Els robots UV-C del mercat tenen inclòs un sensor que els permet apagar-se en cas que es detecti moviment, ja que la llum ultraviolada és perillosa pels éssers vius. No obstant, en el meu treball no s'ha tingut en compte aquesta possibilitat, ja que l'algoritme no està preparat.

En segon lloc, la cel·la inicial on es col·loca el robot sempre ha de formar part de la sala A i trobar-se en la fila inferior. Clarament, això presenta moltes limitacions de cara a la vida real. Per tant, una altra cosa que afegiria seria la possibilitat de moure's en vertical i poder començar en qualsevol sala.

Finalment, m'agradaria que el robot no necessites tenir tota la informació del seu entorn a priori. Per al projecte no hi hagut problema perquè els escenaris de prova són petits i de poques sales, però en el moment en que el robot hagi de netejar un espai gran, guardar tanta informació serà molt costós i no serà escalable.

A més a més, això permetria que l'espai canviés i el robot tingués prou flexibilitat per adaptar-se. Per exemple, moure de lloc un obstacle d'una sala que encara no s'ha netejat.

## Agraïments

En primer lloc, m'agradaria donar les gràcies al Samir i al Gerard per la seva ajuda i els seus consells durant tot el treball.

En segon lloc, gràcies a l'Adrià Puigdellívol pel seu suport durant tot el temps que he estat amb el projecte, des de l'inici fins al final. Gràcies per ajudar-me amb Unity i C#, i pels consells que m'has anat donant.

En tercer lloc, gràcies a l'Amit Patel per escriure un blog sobre AI, heurística i algorismes de *pathfinding*.

Per últim, gràcies a la meva família per estar al meu costat durant tots els anys a l'ETSEIB, per animar-me a seguir endavant passi el que passi.

## Bibliografia

### Referències bibliogràfiques

- [1] SpeedyCare™UV, *Reducció logarítmica de desinfecció*. [https://www.speedycare.es/sobre-uvc.php#log, 12 de novembre de 2020]
- [2] UVD ROBOTS®, *We are awarded as Denmark's most sustainable solution in the healthcare sector*. [https://www.uvd-robots.com/blog/we-are-awarded-as-denmarks-most-sustainable-solution-in-the-healthcare-sector, 9 d'abril de 2021]

### Bibliografia complementària

SpeedyCare™ [https://www.speedycare.es, 10 de novembre de 2020]

UVD Robots [https://www.uvd-robots.com, 10 de novembre de 2020]

Code Monkey. (2019, 20 d'octubre). A\* Pathfinding in Unity [Arxiu de vídeo]. Recuperat de <https://www.youtube.com/watch?v=alU04hvez6L4>. Consultat el 12 de novembre de 2020.

Matachana [https://www.matachana.com/es/robot-uv-promo?utm\_source=gads&utm\_medium=search&utm\_campaign=robot-uvc&gclid=CjwKCAiA17P9BRB2EiwAMvwNyBAylJv\_uP1hVQvALXSM9BI6k\_fy1ftV859A3u9\_4oJgrlUqhhuEaBoC1Y8QAvD\_BwE, 12 de novembre de 2020]

Wikipedia, *Nearest neighbour algorithm* [https://en.wikipedia.org/wiki/Nearest\_neighbour\_algorithm, 13 de novembre de 2020]

Wikipedia, *Upper and lower bounds* [https://en.wikipedia.org/wiki/Upper\_and\_lower\_bounds, 13 de novembre de 2020]

Wikipedia, *Dynamic programming* [https://en.wikipedia.org/wiki/Dynamic\_programming, 13 de novembre de 2020]

Wikipedia, *Travelling salesman problem* [https://en.wikipedia.org/wiki/Travelling\_salesman\_problem, 13 de novembre de 2020]

Code Monkey, (2019, 4 d'octubre). Grid System in Unity [Arxiu de vídeo]. Recuperat de <https://www.youtube.com/watch?v=waEsGu--9P8&list=PLzDRvYVwI53uhO8yhqxcyjDIImRjO9W722&index=3&t=14s>. Consultat el 13 de desembre de 2020.

Jesse Glover, StackExchange, *Calculating distance with Euclidean, Manhattan and Chebyshev in C#* [https://codereview.stackexchange.com/questions/120933/calculating-distance-with-euclidean-manhattan-and-chebyshev-in-c, 1 de febrer de 2021]

Patel, Amit. *Heuristics* [http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html, 1 de febrer de 2021]





Patel, Amit. *Introduction to the A\* Algorithm* [<https://www.redblobgames.com/pathfinding/a-star/introduction.html>, 1 de febrer de 2021]

AIS [<https://www.ai-systems.ca/custom-robotics-company-ais/>, 8 de març de 2021]

Unity, *Documentation Prefabs* [<https://docs.unity3d.com/es/530/Manual/Prefabs.html>, 13 de desembre de 2020]

Desconegut, *Introducción a Json* [<https://www.json.org/json-es.html>, 7 d'abril de 2021]

“Tdykstra” & “Olprod”, *Procedimiento para serializar y deserializar JSON en .NET* [<https://docs.microsoft.com/es-es/dotnet/standard/serialization/system-text-json-how-to?pivots=dotnet-5-0>, 7 d'abril de 2021]

Gencat, *Factor de emisión de la energía eléctrica: el mix eléctrico* [[https://canviclimatic.gencat.cat/es/actua/factors\\_demissio\\_associats\\_a\\_lenergia/](https://canviclimatic.gencat.cat/es/actua/factors_demissio_associats_a_lenergia/), 9 d'abril de 2021]

Eco, *Uso del ordenador portátil vs ordenador sobremesa* [<http://www.ecorresponsabilidad.es/fichas/portatil.htm>, 9 d'abril de 2021]

UVR, *Lights Out: How to replace and dispose of your UV lamps* [<https://uvresources.com/lights-out-how-to-replace-and-dispose-of-your-uv-lamps/>, 9 d'abril de 2021]

## Annexos

### CleanRoom

```

public void CleanRoom(Cell.RoomType roomtype) {

    while(!GridManager.Instance.RoomCleaned(roomtype)) {
        // Buscar camí
        if(battery <= LOW_BATTERY) {
            var nearestChargeCell =
GridManager.Instance.NearestChargeCell(currentCell);
            var goChargeCell = searchPathA(currentCell,
nearestChargeCell);
            goChargeCell.RemoveAt(0);
            addCellsToPath(goChargeCell);
            var comeBack = searchPathA(nearestChargeCell,
currentCell);
            comeBack.RemoveAt(0);
            for(int i = comeBack.Count - 1; i >= 0; i--) {
                if (comeBack[i].GetTimesPassed() <
comeBack[i].GetDirtLevel() &&
GridManager.Instance.LowerCellsClean(comeBack[i].Y)) {
                    if(i < comeBack.Count - 1) {
                        comeBack.RemoveRange(i + 1,
comeBack.Count - i - 1);
                    }
                    break;
                }
            }
            addCellsToPath(comeBack);
            currentCell = comeBack[comeBack.Count - 1];
        }
        else {
            var nextCells = searchPath(roomtype);
            addCellsToPath(nextCells);
            currentCell = nextCells[nextCells.Count - 1];
        }
    }
    if((int) roomtype < (GridManager.Instance.GetRoomCount() -
1)) {
        var nextRoomStartCell =
GridManager.Instance.StartCell(roomtype + 1);
        var nextCellsToNextRoom = searchPathA(currentCell,
nextRoomStartCell);
        nextCellsToNextRoom.RemoveAt(0);
        addCellsToPath(nextCellsToNextRoom);
        currentCell =
nextCellsToNextRoom[nextCellsToNextRoom.Count - 1];
    }
}

```

### searchPath

```

private List<Cell> searchPath(Cell.RoomType roomtype) {
    var direction = searchDirection(currentCell, roomtype);

```

```

var llista = new List<Cell>();
if(currentCell.GetDirtLevel() > currentCell.GetTimesPassed())
{
    llista.Add(currentCell);
    return llista;
}
var horizontalcell =
GridManager.Instance.GetCell(currentCell.X + direction,
currentCell.Y);
if(horizontalcell != null && horizontalcell.GetCellType() ==
Cell.CellType.Floor && horizontalcell.GetRoomType() == roomtype &&
horizontalcell.GetDirtLevel() > horizontalcell.GetTimesPassed()) {
    llista.Add(horizontalcell);
    return llista;
}

var nextRowCell =
GridManager.Instance.GetNextRowCell(currentCell.X, currentCell.Y,
roomtype);
if(nextRowCell != null) {
    llista = searchPathA(currentCell, nextRowCell);
    llista.RemoveAt(0);
    return llista;
}

var verticalcell =
GridManager.Instance.GetCell(currentCell.X, currentCell.Y + 1);
if(verticalcell != null && verticalcell.GetCellType() ==
Cell.CellType.Floor && verticalcell.GetRoomType() == roomtype &&
verticalcell.GetDirtLevel() > verticalcell.GetTimesPassed()) {
    llista.Add(verticalcell);
    return llista;
}

var nextColumnCell =
GridManager.Instance.GetNextRowCell(currentCell.X, currentCell.Y + 1,
roomtype);
if(nextColumnCell != null) {
    llista = searchPathA(currentCell, nextColumnCell);
    llista.RemoveAt(0);
    return llista;
}
return llista;
}

```

### searchPathA

```

private List<Cell> searchPathA(Cell startCell, Cell lastCell) {

    var openList = new List<Cell> {startCell};
    var closedList = new List<Cell>();

    GridManager.Instance.ResetCosts();

    GridManager.Instance.GetCell(startCell.X,
startCell.Y).SetGCost(0);

```

```

    GridManager.Instance.GetCell(startCell.X,
startCell.Y).SetHCost( calculateDistance(startCell, lastCell));
    GridManager.Instance.GetCell(startCell.X,
startCell.Y).CalculateFCost();

    while (openList.Count > 0) {
        Cell currentC = getLowestFCostCell(openList);
        if (currentC.X == lastCell.X && currentC.Y ==
lastCell.Y) {
            //Reached final node
            return CalculatePath(lastCell);
        }

        openList.Remove(currentC);
        closedList.Add(currentC);

        foreach (Cell neighbourCell in
GetNeighbourList(currentC))
        {
            if (closedList.Contains(neighbourCell))
            {
                continue;
            }
            if (!neighbourCell.IsWalkable()) {
                closedList.Add(neighbourCell);
                continue;
            }

            int tentativeGCost = currentC.GetGCost() +
calculateDistance(currentC, neighbourCell);
            if (tentativeGCost < neighbourCell.GetGCost()) {
                neighbourCell.CameFromCell = currentC;
                neighbourCell.SetGCost(tentativeGCost);
                neighbourCell.SetHCost( calculateDistance(neighbou
rCell, lastCell));
                neighbourCell.CalculateFCost();

                if (!openList.Contains(neighbourCell)) {
                    openList.Add(neighbourCell);
                }
            }
        }
    }

    // Out of nodes on the openList
    return null;
}

```

### searchDirection

```

private int searchDirection(Cell celaInici, Cell.RoomType roomtype) {
    var rowCells = GridManager.Instance.GetRowCells(celaInici.Y);
    var celesDreta = rowCells.FindAll(cell => cell.X >
celaInici.X && cell.GetDirtLevel() > cell.GetTimesPassed() &&
cell.GetCellType() == Cell.CellType.Floor && cell.GetRoomType() ==
roomtype);
}

```

```

        var celesEsquerra = rowCells.FindAll(cell => cell.X <
celaInici.X && cell.GetDirtLevel() > cell.GetTimesPassed() &&
cell.GetCellType() == Cell.CellType.Floor && cell.GetRoomType() ==
roomtype);
        if (celesEsquerra == null || celesEsquerra.Count == 0) return
1;

        if (celesDreta == null || celesDreta.Count == 0) return -1;
        if (celesEsquerra.Count >= celesDreta.Count) return 1;
        else return -1;
    }

```

### NearestChargeCell

```

public Cell NearestChargeCell(Cell currentCell) {
    var allChargeCells = listCell.FindAll(cell =>
cell.GetCellType() == Cell.CellType.Charge);
    Cell nearestChargeCell = allChargeCells[0];
    for (int i = 1; i < allChargeCells.Count; i++) {
        int xDistancei = Mathf.Abs(currentCell.X -
allChargeCells[i].X);
        int yDistancei = Mathf.Abs(currentCell.Y -
allChargeCells[i].Y);
        int distanciai = xDistancei + yDistancei;
        int xDistancen = Mathf.Abs(currentCell.X -
nearestChargeCell.X);
        int yDistancen = Mathf.Abs(currentCell.Y -
nearestChargeCell.Y);
        int distancian = xDistancen + yDistancen;
        if (distanciai < distancian) {
            nearestChargeCell = allChargeCells[i];
        }
    }
    return nearestChargeCell;
}

```

### StartCell

```

public Cell StartCell(Cell.RoomType roomtype) {
    var celesRoom = listCell.FindAll(cell => cell.GetRoomType()
== roomtype && cell.GetCellType() == Cell.CellType.Floor);
    Cell startCell = celesRoom[0];
    for (int i = 1; i < celesRoom.Count; i++) {
        if(celesRoom[i].X <= startCell.X && celesRoom[i].Y <=
startCell.Y) {
            startCell = celesRoom[i];
        }
    }
    return startCell;
}

```