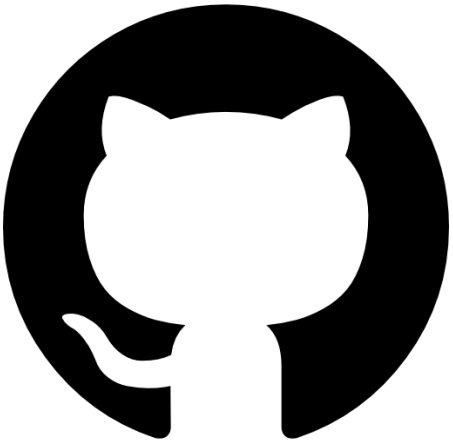
DOCUMENTACIÓ ROBOCODE EX1
TIMIDÍN ROBOT

Denis Vera Iriyari



GitHub Projecte

Plantejament

Vaig començar testejant i trastejant amb les eines que es necessiten pel projecte fins sentir que entenia correctament totes les aplicacions i possibilitats que tinc a mà. Un cop entès tot això i assegurant-me amb un prototip bàsic funcional(vaig utilitzar un Robot Patrulla Programat de les transparències) que funcionaba tot correctament, és a dir, que l'aplicació de RoboCode llegia bé la classe vaig començar a plantejar la lògica i l'estructura amb la qual implementar el Timidín Robot.

Vaig decidir dividir les diferents accions del robot en 3 estats: {SEARCHING, MOVING, SHOOTING}. On 'SEARCHING' fa referència a la primera fase o fase 0 de l'enunciat, que bàsicament cerca a l'enemic i estableix la cantonada més llunyana a aquest. En l'estat 'MOVING' ens mourem a aquesta cantonada en línea recta solucionant alguns bugs que surten quan el robot xoca amb una paret propera a la cantonada (segons la posició inicial del robot i tenint en compte el seu grossor o 'hitbox' a vegades té problemes per arribar correctament a la cantonada (sobretot en angles tancats), per tant quan xoca amb algúna de les parets properes, simplement recorre, ja sigui vertical o horitzontalment, els pocs píxels de recorregut que li falten). També solucionant possibles col·lisions amb enemics, si el Robot Timidín es troba en aquesta situació, disparará a l'enemic amb el que ha col·lisionat amb potència 20 (per exemple, ja que és un dispar que quasi segur no fallarà) i acte seguit modificarà la trajectòria per tal d'arribar a la cantonada desitjada correctament. Per últim, la fase 2 on ja estem situats a la cantonada més llunyana al robot rival, canviem l'estat a 'SHOOTING' on bàsicament estem constantment detectant la posició del rival i adjustant el nostre canó per disparar-li constantment amb una potència de dispar inversament proporcional a la distància.

Amb aquest plantejament, simplement ha faltat implementar les funcions necessàries a base de prova i error per tal de que el Robot actuès acord a l'enunciat de l'exercici.

Variables

-private enum StateRobot {SEARCHING, MOVING, SHOOTING};
-private StateRobot state;

Són les variables que permeten manipular el codi diferenciant segons l'estat al que es troba el nostre robot, això facilita molt la comprensió del codi per entendre quan i per què es fa cada acció.

-private double targetDirection;

Variable on guardarem la direcció del robot escanejat en primera instància per calcular la cantonada més llunyana amb aquesta data i la distància entre el nostre robot i el robot enemic que hem detectat.

-private double furthestX, furthestY;

Variables on guardarem les coordenades de la cantonada més llunyana al robot detectat per tal de facilitar la informació alhora de moure al robot a aquesta cantonada, ens servirà també per resoldre els bugs esmentats anteriorment i per evitar les col·lisions amb robots amb èxit.

Funcions

-public void run()

Simplement selecciona els colors del robot, inicial·litza ‘state’ amb ‘SEARCHING’ (fase 0) i comença el bucle on el nostre robot prendrà accions segons l’estat al qual sigui assignat seguint les fases establertes.

-private void lookForEnemy()

Rota el nostre robot cercant escanejar un robot rival.

-public void onScannedRobot(ScannedRobotEvent event)

Un cop escanejar un robot, si estem a la fase 0 calcule la cantonada més llunyana i canviem a l’estat ‘MOVING’, és a dir, a la fase 1. En canvi si escanejem a un robot estant a la fase 2, vol dir que el nostre robot ha arribat a la cantonada desitjada, per tant, començarem a disparar al robot amb els requisits ja esmentats sense parar de buscar-ho amb ‘lookForEnemy()’.

-private void FurthestCorner(double direction, double distance)

Assigna a ‘furthestX’ i ‘furthestY’ les coordenades de la cantonada més llunyana a la posició del robot escanejat calculant la posició exacta del rival i comparant-la amb els límits de l’arena.

-private void moveToCorner()

Calcula l’angle al qual ha de situar-se per anar fins a la cantonada en línia recta, fa aquest gir i després comença el seu camí fins al destí. En cas que col·lisioni amb un robot, la funció ‘onHitRobot()’ interromprà i després de sol·lucionar aquesta topada amb el rival tornarem a aquesta funció ‘moveToCorner()’ per tal de recalcular el camí que haurem de seguir per arribar al nostre objectiu. També solucionarà els bugs citats abans trucant a la funció ‘handleWallCollision()’. Per últim, si tot ha anat correctament, estarem situats a la cantonada i canviarem l’estat a la última fase, la fase 2 o ‘SHOOTING’.

-private void handleWallCollision()

Quan el robot s’estanca a prop de una paret propera a la cantonada, ho arregla avançant només en la coordenada que ha estat bloquejada.

-private double CalculateDirection(double targetX, double targetY)

Retorna l’angle al qual el robot ha de rotar per arribar a les coordenades del paràmetre (‘targetX’,’targetY’).

-public void onHitRobot(HitRobotEvent event)

Quan un robot col·lisiona amb nosaltres, el primer que fem és apuntar-li i disparar, després si el robot està a una posició que bloqueja la nostra trajectòria, donem uns passos enrere i truquem a la funció ‘moveAroundEnemy()’, després d’aquesta interrupció tornarem a ‘moveToCorner()’ per seguir amb el nostre objectiu de posicionar-nos en la cantonada més llunyana a l’enemic.

-public void moveAroundEnemy(double distance)

Mètode per moure's al voltant del robot rival per tal d’evitar bloquejos mentre intentem anar a la cantonada, ‘distance’ és la distància que volem recórrer rodejant a l’enemic.

-public void onPaint(Graphics2D g)

Mostra radi del robot a la interfície del RoboCode.

-public double normalizeBearing(double angle)

Funció per assistir la normalització de l’angle, és a dir per buscar la menor distància a la qual hem de girar per posicionar-nos en la direcció ‘angle’.

Observacions

Tot i que el plantejament del projecte i la majoría de codi ho he pensat, processat i implementat jo, algunes funcions les he fet amb ajuda de chatGPT (no las he simplement copiat, si no que les he redissenyat per implementar–les al meu projecte ja que directament les funcions generades per IA no solen estar 100% correctament lligades al teu programa, però ha sigut de gran ajuda per facilitar funcions secundàries que simplement assisteixen).

Aquestes han sigut les següents:

- -FurthestCorner()
- -calculateDirection()
- -moveAroundEnemy()
- -normalizeBearing()

Com pots veure són funcions de càlculs ràpids o simplement d'assistència a les altres funcions principals que són les que realment tenen pes.

Aquesta classe no em sembla una bona classe per a lluitar contra altres robots ja que si ens cenyim a l'enunciat i el TimidinRobot es va a una cantonada i comença a disparar al seu enemic estant completament quiet i el seu enemic s'està constantment movent, encara que agafi constantment les seves coordenades fallarà la majoria de dispars i sumat a això és un blanc molt fàcil degut a que no es mou de la cantonada.