Documentació Robocode Ex 2

Ixent Cornella, Arnau Roca

Exercici 2:

En aquest segon exercici, on nosaltres haviem de plantejar l'estratègia, abans de començar a programar res vam decidir fer una recerca exhaustiva de com podria ser el millor funcionament per un robot d'equip.

Vam estar mirant el funcionament i estratègies dels millors robots del roborumble (competència entre diferents robots de robocode per veure quin és el millor) i al veure les estratègies i codis ens vam abrumar bastant. Per implementar alguna de les seves estratègies (i entendre-la, que copiar-ho sense entendre no val) ens haguessin estat moltes hores. Amb una mica de desesperació vam decidir provar amb el codi que ja teníem fet del corner team, que passaria si tots fossin kamikazes, ja que funcionava bastant bé el robot kamikaze. No sabem exactament per quin motiu però no acabava de donar els resultats que esperàvem i després de canviar la meitat del codi del kamikaze vam decidir desestimar l'opció. En aquest punt ja se'ns acabaven les idees, i vam decidir provar amb un robot simple (a vegades menys és més) que avancés poc i que la complexitat del codi estigués en el moment de disparar.

TARGETING

Així doncs, en fer un robot senzill per començar a desenvolupar la idea sobre aquest ens vam donar compte d'una cosa, al disparar era molt poc eficient, ja que disparava al primer robot escanejat, i tota la estona al anar escanejant anava disparant a robots diferents (cosa que feia que als que estiguessin més lluny falles moltes bales). Aquí va ser quan vam veure una mica de llum i vam decidir implementar un sistema que escanegés tot l'entorn i trobés el robot que estava més a prop seu, llavors fixés aquell objectiu fins destruir-lo.

Per fer-ho vam implementar un HashMap de string i double, per tal que emmagatzemés el nom del robot i la distància fins a aquest (només dels enemics). Dit així sembla senzill, però com que els robots es van movent, això havia de ser tota la estona. Llavors vam fer que cada cop que s'escaneja un robot enemic, es guardés la distància del robot a ell. Més tard, amb les distàncies obtingudes, cridem la funció pròpia getTarget() que, sabent les distàncies a cada robot enemic, escolleix el que està més aprop. D'aquesta forma, només s'atacava al robot que es té més a

prop. Vam afegir diverses precaucions, ja que sinó el robot podia fallar, per això vam implementar un sistema de reconeixement del primer escaneig senzill (només una variable booleana per saber si estem al primer escaneig), i així descartar errors inicials. Després, de vegades quan ja es moria un robot (onRobotDeath) enemic, s'elimina de la llista de distàncies, però per algun error de paral·lelisme i concurrència, de vegades tornava a afegir al robot a la llista de distàncies després del onRobotDeath event, per lo qual vam afegir una variable booleana "eliminant" que evités errors de paral·lelisme.

.

PRIMERS RESULTATS

En les primeres versions del team, quan estavem decidint que és millor que "tarjetegi" el robot, vam comparar tant la distància com l'energia, fins que vam observar els següents resultats:

ADVERSARIS & ESTRATEGIA	Eficiencia	Wins	Losses	Win Rate
CornerTeam -> Target distancia	75%	866	135	86,60%
CornerTeam -> Target energia	52%	344	656	34,40%
MyFirstTeam -> Target distancia	89%	1000	0	100,00%
MyFirstTeam -> Target energia	86%	999	1	99,90%
CornerTeam -> Target energia i distancia	76%	894	106	89,40%

Figura 1: Comparativa del nostre team al principi vs altres, en funció del targeting.

El resultat era similar quan es tenien en compte les dues variables que quan es tenia en compte només la distància, per tant vam decidir d'usar únicament la distància per a escollir el nostre target, però també tindriem l'energia en compte per a esquivar bales si detectem que el nostre target ha perdut vida (probablement o bé l'han disparat, o bé ha disparat).

FRIENDLY FIRE

Un cop implementat el sistema de selecció de target, vam decidir d'implementar un mètode anti friendly-fire, que s'executava cada cop que s'anava a disparar per evitar impactar a un company. La manera en la que funciona és simple, només es guarda un HashMap nou (igual que el de distàncies) per a saber el angle al que està el robot aliat respecte al propi robot. D'aquesta manera, cada cop que escanegem un

aliat, guardem el angle al que es troba respecte a nosaltres. Quan cridem el mètode friendlyFire(angle), li passem com a paràmetre l'angle al que pensem disparar, i si detecta que hi ha un robot dins del marge de dispar (hem decidit que sigui angle-7º, angle+7º), no es dispara. D'aquesta manera evitem disparar quan un robot es troba directament a la línia de foc del robot, però no podem evitar que un cop ja s'ha disparat el robot aliat es creuï amb la bala igualment.

DISPAR CONTROLAT

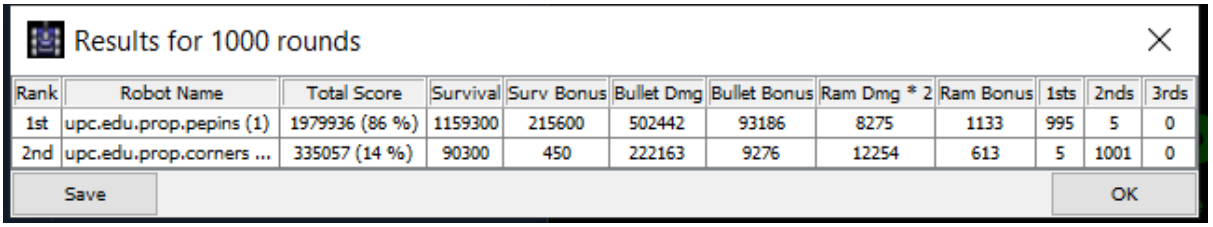
En moltes de les rondes que ens quedàvem mirant, el nostre robot (el qual disparava sempre amb força 3) fallava tantes bales que acaba morint per això. Per tant, vam decidir implementar un sistema que mitjançant la distància amb el seu objectiu decideixi quina potència de dispar farà. Si està més lluny serà una potència més baixa ja que hi ha més probabilitats de fallar. Això ho hem aconseguit mitjançant un càlcul que ens retorna el mínim entre 900 dividit la distància i 3. Això funciona ja que a partir de la distància 300, considerem que ja estem prou prop i podem disparar al màxim i quan està més lluny que faci la 900/distància (un nombre que considerem prou lluny, ja que el màxim es 1200 i segons les diferents proves amb diferents valors que hem fet és el que funciona millor). En cas de que la distància fos 600, dispararem amb una potència de 1,5.

STOP AND GO

Per tal d'anar millorant el codi, vam tornar a fer un búsqueda més per les estratègies dels robots d'altra gent, però aquest cop amb robots més humils. Vam descubrir una manera molt senzilla per esquivar bales dels oponents, que era un dels problemes que teniem, anomenat stop-and-go. Com el seu propi nom indica, el que fa aquest sistema és anar arrancant i parant, però ho fa quan detecta que el rival ha disparat. Això en combats de 1vs1 és meravellós, però en aquest cas no ho era tant, és per això que per evitar que es comencés a moure massa, vam decidir fer que tan sols es mogués quan el rival que té més prop dispara (ja tenim identificat aquest rival). Descobrim si el rival ha disparat o no fent un registre constant de l'energia del rival, una altra vegada amb HashMap on guardem l'energia que té i la comparem amb la que torna a tenir en el següent escaneig, llavors si és més petita la nova, vol dir que ha disparat, en aquest punt ens movem una mica i així esquivem la bala (en cas que ens estigui apuntant a nosaltres. És una molt petita millora, ja que en molts casos de

la batalla no és útil, però precisament en molts dels casos en els que perdiem, eren casos que es quedava el nostre robot 1vs1 contra un altre o que els robots enemics estaven apuntant al robot més proper per casualitat (i per tant disparaven més fort i tenen menys probabilitat de fallar) i amb aquest mètode ens cobrim bastant les esquenes en aquests casos.

Cal destacar que el robot es mou de forma continua amb la variable “dist”, que li diu quant ha d’anar endavant i endarrere, i si l’han impactat, quan ha de moure’s. El valor que més ens funciona per dist és de 750, hem trobat una major eficiència.


Results for 1000 rounds

Rank	Robot Name	Total Score	Survival	Surv Bonus	Bullet Dmg	Bullet Bonus	Ram Dmg * 2	Ram Bonus	1sts	2nds	3rds
1st	upc.edu.prop.pepins (1)	1979936 (86 %)	1159300	215600	502442	93186	8275	1133	995	5	0
2nd	upc.edu.prop.corners ...	335057 (14 %)	90300	450	222163	9276	12254	613	5	1001	0

Save OK

Figura 2: Comparativa del nostre team final vs cornerTeam, 1000 rondes.

El resultat final es un robot capaç de vèncer fàcilment a cornerTeam, amb només una classe de java implementada.

VARIABLES I NOMENCLATURA:

- **dist**: variable que ens indica la distància la qual s’ha de moure el robot endavant i endarrere.
- **firstScan**: variable booleana que indica si el robot escaneja per primer cop un robot enemic.
- **eliminant**: variable booleana que indica si el robot està eliminant un robot de la seva llista de distàncies/energies
- **distancies**: HashMap que conté, per cada robot del team enemic viu, la distància a la que es troba respecte al propi robot.
- **energies**: HashMap que conté, per cada robot del team enemic viu, la seva energia.
- **anglesEquip**: HashMap que conté, per cada robot del team viu, l’angle al que es troba respecte al propi robot.

.

MÈTODES PRÒPIS i MÈTODES MODIFICATS:

- **onScannedRobot(Robot)**: Quan escanegem a un robot es cridarà aquesta funció, aquí és on tenim tota la casúistica dels dispars i el targeting al enemic més proper.
- **onRobotDeath(Robot):** Quan es mori un robot, el treiem dels HashMaps on era i posem la variable booleana eliminant a true per tal que no el torni a afegir mentre el borrem.
- **onHitByBullet(Robot):** Quan ens toca una bala, el que fem és girar i avançar
- **getTarget():** funció auxiliar per definir quin és el robot més proper i per tant el nostre objectiu
- **friendlyFire(angle):** Ens retorna els si hi ha algun robot aliat en l'angle que li donem, així sabem si hi ha foc amic o no
- **calcDist(xOrigen, yOrigen, xDesti, yDesti)**: Càlcula la distància entre dos punts a partir de les coordenades d'aquets.
- **getAngleTo(xOrigen, yOrigen, xDesti, yDesti**): Càlcula l'angle del punt Origen al punt Destí.
- **inici():** Tria el color pertinent per al robot.
- **getAngleMoviment(distància, bearing):** amb aquesta funció podem predir una mica el moviment del rival que estem apuntant.