Robocode bot evolválása genetikus programozással

Pető Zoltán

Konzulens: Kovács Dániel László

Tartalom

Az önálló labor munka áttekintése – a feladat

Az önálló labor munka kezdetén az „Intelligens ágensek optimalizálása evóluciós számítási módszerekkel” c. témakiírásra kerestünk alkalmas feladatot a konzulensemmel. A választás a Robocode keretrendszer felhasználásával, egy Robocode bot optimalizására – ez esetben evolvására – esett.

A témakiírásban szereplő módon, evolúciós módszerekkel kellett megvalósítottam az optimalizálást. Ehhez egy általam korábbról már részben ismert „Watchmaker” evolúciós frameworköt használtam. Ez megkönnyítette az implementációt, és módom volt megismerni a frameworköt nagyobb mélységeiben. Úgy érzem ez is hozzájárult a szemléletmódom változásához.

A Robocode

A Robocode egy IBM fejlesztésű keretrendszer, ami programozók, fejlesztők, IT szakemberek számára teszi lehetővé, hogy –egy mókásabb formában- összemérjék tudásukat.

A feladat, hogy egy Robocode robot program kódját írják meg, amelyek összemérik erejüket. A Robocode keretrendszer többféle játékmódot is támogat. Közös ezekben, hogy cél ez ellenséges robot(ok) likvidálása és saját a robot(ok) túlélésének biztosítása.

A tankok egy bizonyos kezdeti energiaszinttel rendelkeznek és lőhetnek egymásra. Találat esetén az ellenfél energiaszintje csökken. Lényegében az ellenfél lövéseinek kikerülésével, illetve a saját lövésekkel és megfelelő célzással tud a robot nyerni.

A feladat választásának oka, hogy egy nagyobb komplexitású feladatot szerettem volna megoldani, amelyben szerepel valamelyin formában a tér és az idő – amelyek együtt nehezítik a feladatot. A harc „turn”-ökre, azaz diszkrét időegységekre van osztva.

Léteznek „1 az 1 ellen”, 2 a 2 ellen”, illetve „mindenki mindenki ellen” (Free for All) játékmódok, illetve ezeknek további variánsai (különböző, a szabályra, játéktérre vonatkozó paraméterezésekkel). Ezek közül én az 1v1-re fókuszáltam. Ennek oka, hogy a választhatóak közül ez volt a legegyszerűbb, de ez is elég komplex ahhoz, hogy érdemi feladat legyen végezhető.

A robotokat egy API-n keresztül tudja a programozó „irányítani”. Megadhatóak parancsok: mennyit mennyen előre vagy hátra, mennyit forduljon és milyen irányban a robot teste, a robot lövege, illetve a robot radarja, illetve ha lövünk, milyen erősséggel lőjünk. A robotok képesek érzékelni egymást és bizonyos eventeket, amelyek szükségesek a cél teljesítéséhez.

Feladatom volt, hogy megmutassam, hogyan tekinthetünk a Robocode robotokra mint ágensekre. Ehhez a következő tudásmérnöki feladatokat végeztem: a keretrendszer nyújtotta alapokon megterveztem az „evolúciós robot” avagy „evolúciós ágens” (amelyet Evobotnak nevezek) Érzékeléseit, Tudásbázisát, Döntéshozó mechanizmusát és lehetséges Beavatkozásait.

Evolúciós számítási módszerek

Az evolúciós módszerek a természetes evolúciót próbálják lemásolni. Az egyedek populációt alkotnak, populáció pedig generációról generációra változik. Leegyszerűsítve a „rátermett”, avagy „inkább rátermett” egyedek kerülnek az új generációba – fenntartva ezzel egyfajta „fejlődést” a generációk között.

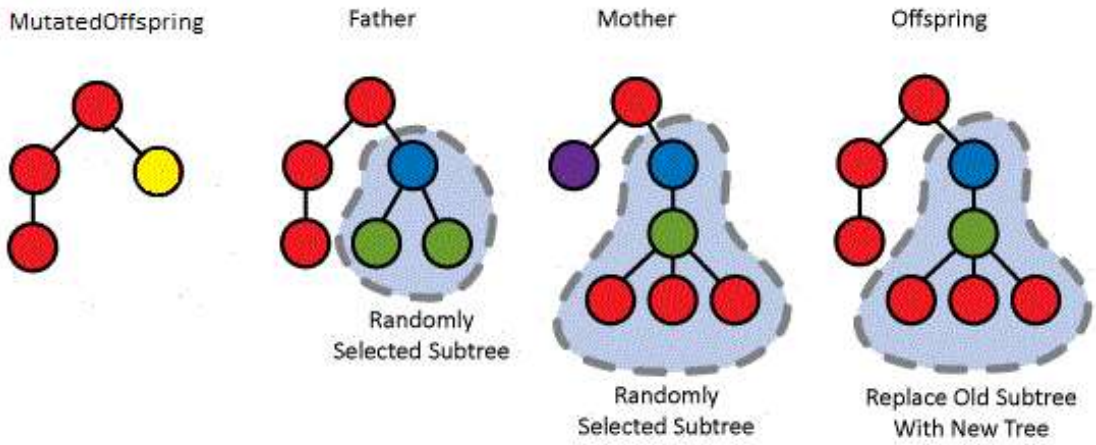
Az evolúciós optimalizálási módszerek globális optimumot meghatározó sztochasztikus keresőeljárások. Az eljárás során egy un. „fitnesst”, egyfajta „jóságéi mércét” használ, ezt közelíti meg. A feladat jó megoldásához megfelelő fitnesst kell találni. Ez alapjaiban határozza meg az evolúció konvergenciáját, „gyorsaságát” és a végeredményt is – azaz hogy mennyire lesznek ténylegesen használhatóak az így talált „optimális” megoldások.

Az általam választott evolúciós módszer a Genetikus Programozás. Ez tekinthető a Genetikus Algoritmusok egy speciális változatának. Míg a Genetikus Algoritmusok esetében a populáció egyedeit bitfűzérek reprezentálják, addig a Genetikus Programozás esetében az egyedek „program-fák”.

Az evolúció lépései evolúciós operátoroknak felelhetőek meg. Ezek az alábbiak: szelekció, mutáció, keresztezés.

A szelekció során az adott generációból kiválasztódnak a következő generáció egyedei – általában a fitnesst értéket valamilyen módon figyelembe véve. A mutáció során

egy egyed (programja, vagy bitfűzére) részben megváltozik. A keresztezés során két ős egyed felhasználásából együttesen keletkezik az utód egyed.



Rendszerterv

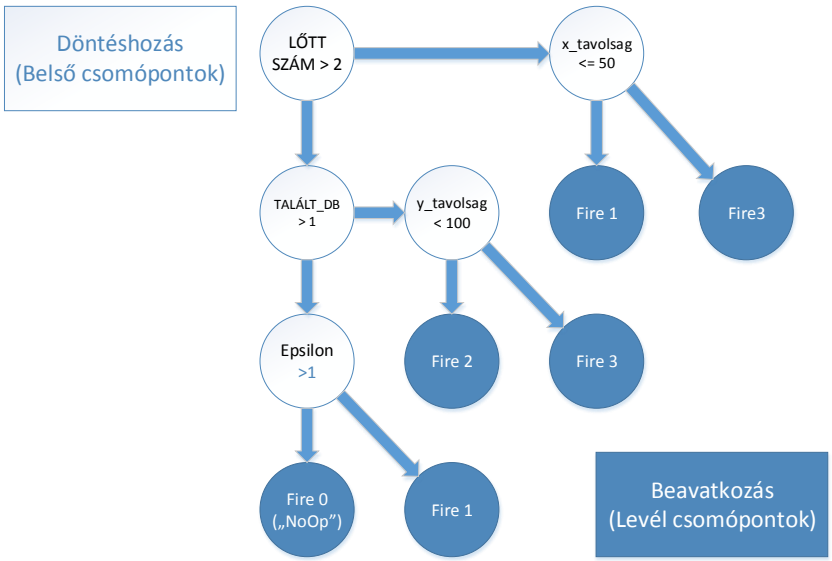
Meg kellett terveznem az ágens Érzékeléseit, Tudásbázisát, a Döntéshozás folyamatát, Beavatkozásokat. Bővebb táblázatokat erről a melléklet tartalmaz.

A célom volt, hogy három – a robot irányítását tekintve egymást kiegészítő – program fát evolváljak ki a „programfa” belső csúcsai és a levelei egyaránt evolválódnak. Ezek a következőket határozzák meg: Lövés és erőssége, Fordulás iránya és mértéke, Gyorsítás iránya és mértéke.

Az ágens érzékeléseihez tartozik az ellenfél sebessége, saját sebesség, ellenféltől való távolság, az ellenfél energiászintja, a saját energiaszint. Illetve megpróbáltam komplexebb, de (egy komplexebb tudásbázishoz releváns) érzékeléseket is kitalálni: pl.: „ellenfél iránya a tank törzséhez képest”

A Tudásbázis az Érzékelésekből származó aggregált tudás. Paraméterezhető a visszaemlékezés mélysége. Ilyenek szerepelnek benne például: ellenfél átlagsebessége, saját átlagsebesség, falaktól való távolság (x és y irányban).

A lehetséges Beavatkozások pedig a fent említett három fa – különböző (érvényes) paraméterekkel.

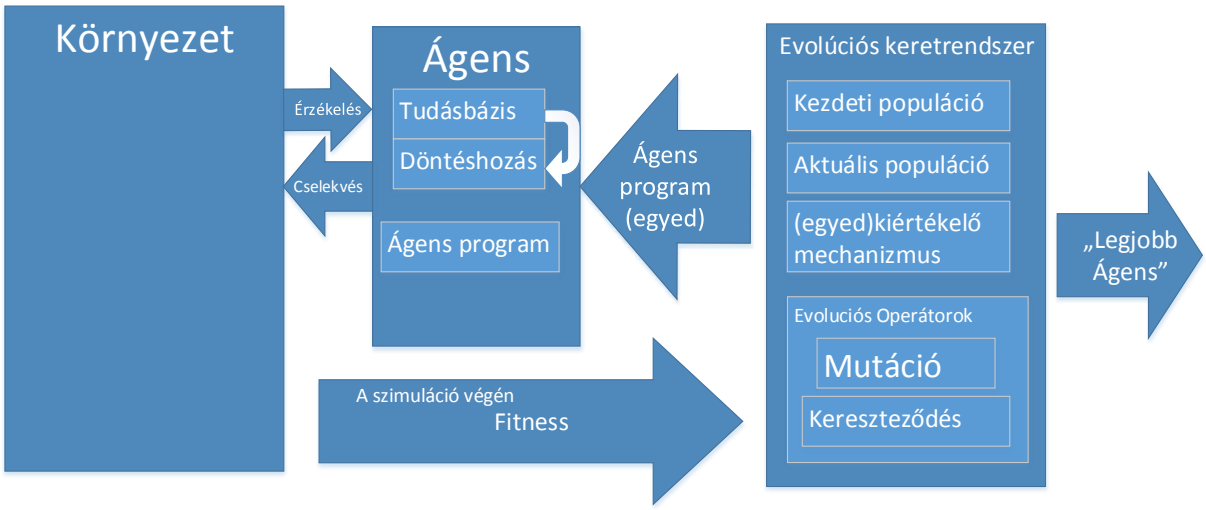


Az alapelv az, hogy egy Evolúciós keretrendszert használok egy Robocode ágens kievolválásához. Ehhez szükség van ezen ágensek kiértékelésére, tehát a fitness értékük meghatározására. Egy Ágenst úgy értékelek ki, hogy szimulációt végzek, az Ágens a környezetébe helyezem. A környezetből az Ágens Érzékeléseket kap, és egy Beavatkozást (Cselekvést) ad vissza a Környezet (Robocode) számára. Az evolúció a korábban leírt módon zajlik.

Egy leállási feltételt teljesítve – az evolúció megáll (pl.: egy egyed elért egy megfelelő fitness értéket) , és a kievolvált „Legjobb Ágenst" megkapom.

A fitness értéket az ellenfél megmaradt energiaszintje jelentette. Ennek oka: egyszerű meghatározni, kellőképp jutalmazzuk így a találatokat és az ellenfél lövedékeinek kikerülését, illetve győzelmet is jól reprezentálja.

A szelekció rulett alapú – tehát a fitness értékek alapján, egy bizonyos eséllyel élnek túl az egyedek. Elitizmus is tervezve volt. Ennek lényege, hogy a generáció „n" darab legrátermettebb elitje változtatás nélkül kerül a következő generációba.



Megvalósítás, felhasznált eszközök

Mivel a Robocode (v 1.9.0.0.) Java alapú keretrendszer, így adta magát, hogy a fejlesztést Java nyelven végezzem. Az általam használt evolúciós framework, a Watchmaker (v.0.7.1) szintén Java alapú.

A fejlesztés során a IntelliJ Idea (13.1.1 Community Edition) IDE-t használtam. Ez a második legelterjedtebb Java ide az Eclipse után, ám az internetes fórumok nyomán fejlesztése ma sokkal jobban halad.

További paraméterek: Windows 8 (64 bit), Intel Core 2 Duo 2.26 Ghz, 4GB RAM.

A szimuláció során az egyes egyedeket reprezentáló programfákat más botok ellen értékeltem ki. Ilyen –kevésbé erős – beépített botok voltak a: „Walls”, „Spinbot”, „VelociRobor” és a „Fire”. Egy versenyző bot ellen is futtattam: ez a „DrussGT” volt, amelynek egy újabb verziója a 2013-mas 1v1 bajnok.

A célzás és a radar használatát nem evolúciós módon programoztam be. A radar használata 1v1 szituációkban egyértelmű – megfelelő működést produkálunk, ha a radarunk végig az ellenfélre néz.

A célzás egy úgynevezett „Head-On targeting” technika amit használtam. Lényege, hogy oda lövünk, ahol az ellenfél épp tartózkodik. Ez egy nagyon egyszerű technika, de ezzel valamelyest biztosítottam, hogy legyenek találatai az evolválódó botnak.

Mivel a keretrendszerben nem volt benne a Genetikus Programozás mutációja és kereszteződése általánosan, így egy ezeket én implementáltam.

Kiértékelés, futási eredmények

A futások során az emlékezés mélységét nem változtattam, ez mindenhol 100 „turn” volt. A beépített botok ellen nagyon könnyen győzedelmeskedik a robot, ami betudható annak, hogy „Head-On” targeting mechanizmus megfelelő ellenük, nem mozognak jól.

A következő táblázatban a DrussGT elleni kiértékelés eredményei találhatók.

pm	Max mélység	Belső Node esély	Körök száma/Meccs	Populáció méret	Elitek	Generáció	Futási idő (min)	Normált fitness
0.1	4	0.75	10	20	10	25	25	51.9
0.2	6	0.82	10	20	10	31	22	68.7
0.3	3	0.65	10	10	3	26	21	73.6
0.2	6	0.65	10	20	5	28	22	**12.4 !**
0.2	10	0.85	5	20	5	80	20	64.2

Látható, hogy Druss GT ellen nagyon kis esély van győzedelmeskedni (hiszen a versenyrobotok ellen a „Head-On” mechanizmus nem eredményes, hiszen szinte mindenképp kitérnek a pontosan feléjük lőtt lövedékek elől) – a fitness többnyire 50 fölött van az egyes populációkban.

Látható azonban hogy van fejlődés. A jóval 100 alatti fitnnesek azt jelentik, hogy egészen jó mozgáskultúra fejlődött ki a DrussGT ellen.

Érdekesség található az utolsó előtti sorban: kifejlődött –majd elterjedt- egy olyan robot, ami meglehetősen sokszor képes nyerni a DrussGT ellen , kihasználva annak egy hibáját.

A stratégia egy része, hogy a robot nem lő (hiszen versenyrobotok ellen ez a célzás úgyis csak az ő energiaszintjén rontana), és nagyon lassú forgással a falnak ütközik, és végig a fal felé mozog. Ennek hatására a DrussGT valamiért összezavarodik, és a lövéseinek nagy részét nem az így játszó robot irányába adja le – így gyorsabban csökken az energiaszintje a DrussGT-nek. Érdekes, hogy az evolúció ilyen „kiskapukat” is képes feltárni.

Sajnos nem végeztem elég futtatást, ahhoz hogy kialakuljon valami komolyabb trend – nehezen tudom őket értékelni. Inkább csak emiprikus úton vettem észre az alábbakait.

Fontos megemlítenem, hogy úgy vettem észre, hogy a futtatások során előny ha „nagyobb teret” adok az evolúciónak. Nagyobb mutálási valószínűség, ám kisebb fáknál. Ugyancsak felesleges sok szimulációt végezni a kis fitnessű egyedekkel. Így – bár nagyobb szórással –de mégis hamarabb kaptam valami látható eredményt. Hosszú távon persze ez lehet, hogy épp megfordul – akár erre is lehetne egy optimalizálást végezni: mikor meddig érdemes kiértékelni egy-egy egyedet.

Összefoglalás

A félévi munkám során megterveztem egy Robocode botok GP-alapú evolúciós optimalizációját biztosító rendszert. Implementáltam a rendszert a Watchmaker framework felhasználásával. Valamelyest teszteltem az így elkészített rendszert.

Tovább fejlesztési lehetőségek:

- A célzás irányát optimalizáló fa (4.) implementálása.
- Ellenfélmodellezés.
- Több Robocode versenyszámban való részvétel.
- Öntanulás megvalósítása.
- Valamint – amit a legfontosabbnak érzek: még több tesztelés. A jövőben szeretném kibővíteni a rendszert egy jól megtervezett tesztkörnyezettel, hogy átláthatóak legyenek az eredmények, és ezeket megfelelőképp tudjam magyarázni.
    - Szeretném vizualizálni az adataimat többféleképp is.

Melléklet

A melléklet tartalmazza az önálló labor során tervezett specifikációt.

A kiemelések jelentése a következő:

- zöld: a végleges implementációba biztosan be kell kerülnie
- piros: valamilyen okból elvetettem
- rózsaszín: a végleges implementációba biztosan nem fog bekerülni (idő hiányában)

Érzékelések / Sensing

Érzékelések (/turn)	Rövidítés	Értékkészlet (mértékegység)	Komment • egy adott turn-ön belül ezeket érzékeli az ágens • sok minden ScannedRobotEventből (SRE) kérhető le
Ellenfél sebessége	V_{ELLEN}	[0,8] (px/ turn)	• SRE: getVelocity
Saját sebesség	$V_{SAJÁT}$	[0,8] (px/ turn)	
Ellenféltől való távolság	d	[36,1000] (px)	• ~0…~pálya átló • tank szélesség: 36 • pálya: 800x600 • SRE: getDistance
Ellenfél energiája	E_{ELLEN}	[0,200] (energy)	• SRE: getEnergy • ~200 fölé biztosan nem fog menni 1v1-ben
Saját energia	$E_{SAJÁT}$	[0,200] (energy)	• …szintén…
Lőttem-e?	LŐTT	{0,1} (darab)	• szükséges: elveszhet az adat • (később szintek?: a sebesség miatt) • **HEAT MIATT NEM HASZNALOM**
Ellenfél haladási iránya	I_{ELLEN}		• getHeading , **vagy a Tudásbázisból, előző hely alapján** • **ERRŐL NINCS INFO! KIKÖVETKEZTETNI LEHET**
Ellenfél pozíciója	P_{ELLEN}	x= [0,800] (px) y= [0,600] (px)	• (x,y) koordináta • Még nincs implementálva
Ellenfél iránya a Bodyhoz képest	Bear_Body ($I_{SAJÁT}$)		• Sorozatok lövéséhez. **(ellenfélhez képest!)** • **Haladasi irany – az ellenhez kepest**
Saját pozíció	$P_{SAJÁT}$	x= [0,800] (px) y= [0,600] (px)	• Faltól való távolsághoz.
Ellenfél iránya a löveghez (headinghez) képest	Bear_Gun	[-180,180] (fok) [-PI, PI] (radián)	• SRE: getBearing, getBearingRadians • **Gunirany – az ellenhez kepest** • *formulával, getHeading helyett Gunheadinggel*
Kilőtt lövedék (objektum)	Löv	N/A	• Értesít, ha célba érkezik – • Talál, vagy nem talál - A különböző sebességek miatt akár többször
Saját Heat	$H_{SAJÁT}$	[0,20] (heat)	• **TB?**
Eventek - OnHit			•

Tudásbázis / Belief Base

Hiedelmek	Szükséges érzékelések és hiedelmek	Értékkészlet t ϵ [0…Inf.] deltat ϵ Z+	Komment „**-t**” turnre visszamenőleg „**deltat**” időközzel – akár több ilyen idősor más lépésközzel
LŐTT_DB_SAJÁT Saját lövések száma „**-t**” turnre	LŐTT	[0,8] (darab)	• Sorozatlövésekhez • Értékkészlet: tapasztalati úton becsülve.
ELTALÁLTAM_DB Találataim száma „**-t**” turnre	Löv **Event(uj)-ezt lehet igy?**	{0,1,2,3} (darab)	• 0 és 1 lesz az idő 99.99%-ban. **(2 érték elég?)** • Fitnesshez
ELTALÁLT_DB Ellenfél találatainak száma „**-t**” turnre	$E_{SAJÁT}$	{0,1,2,3} (darab)	• 0 és 1 lesz az idő 99.99%-ban. **(2 érték elég?)** • Fitnesshez
α célzást segítő paraméter	$l_{ELLEN(norm)}$ V_{ELLEN} Bear d - helyett (A_{ELLEN})	[0,0.2185] (fok/ turn)	• Becsült szögtartomány: megadja, p turn alatt mennyit haladna az ellenfél egyenes vonalú egyenletes mozgást feltételezve • Értékkészlet: alfa kicsi (csak +, vagy 0), **ki kell tapasztalni** max: legközelebb (36px)-re van az ellenfél, maximális, ránk merőleges sebességgel. $tang^{-1}(\alpha) = (8px/tick)/(36px)$ • **(lövedék típust/sebességet belevenni?)** • **l_{ELLEN}-t normalizálni kell: Bearingre vett skalárszorzata, vagy az ellenfél és a saját pozíciót használjuk becslésre**
v/d általánosabb mint az előző	V_{ELLEN} d (A_{ELLEN})	[0, 0.25] (1/turn)	• Ez is célzást segítő paraméter • Értékkészlet: 0:, ha v = 0; 0.25 = ha legközelebb van és leggyorsabban megy: (8px/tick) /(36px)
$V_{ELLEN-ATLAG}$ Az ellenfél átlagsebessége „**-t**” turnre	V_{ELLEN}	[0,8] (px/turn)	• „e” értékből „i” időközönként • **predikcióhoz? (alfa?)** • Értékkészlet: a min/max sebesség miatt.
A_{ELLEN} az ellenfél gyorsulása	V_{ELLEN}	[-2,1] ($px/turn^2$)	• (utolsó időpillanat) • Értékkészlet: a min/max gyorsulás miatt.
$A_{ÁTLAG}$ átlaggyorsulás (ellenfélé) „**-t**” turnre	V_{ELLEN}	[-2,1] ($px/turn^2$)	• **mintázat…** oszcillálás! „e” értékből „i” időközönként számítva - lásd: átlagpozíció(k) • (mint A_{ELLEN}, több időpillanat)
$V_{SAJÁT-ÁTLAG}$ Saját átlagsebesség „**-t**” turnre	$V_{SAJÁT}$	[0,8] (px/turn)	• Lehet egy optimális sebesség, amivel még jól lehet manőverezni (fordulás, elég kiszámíthatatlan: sok út/idő), de gyorsítani, lassítani is tud – véletlen módon – kiszámíthatatlan lehet • **mintázat…** oszcillálás! „e” értékből „i” időközönként számítva - lásd: átlagpozíció(k)
$P_{ELLEN-ATLAG[]}$ (átlagpozíció(k)	P_{ELLEN}	x= [0,800](px)	• **mintázat** (pl. oszcillálás) felismeréséhez,meg adhatunk paraméterként(q) egy számot, hogy várhatóan hány állapot között

q, hibafüggvény) „**-t**” turnre		y= [0,600](px)	váltogat az ellenfél, és ennyi koordinátával próbálja leginkább leírni az „átlagpozíciók” által meghatározott pályát. – négyzetes hiba?
Δd „**-1**” turnre	d	[-16,16] (px)	• közeledés (-), vagy távolodás (+) • utolsó 2 érték alapján • EZ NEM AZ ELLENFL KÖZELEDÉSE, LEHET HOGY CSAK MI MOZGUNK, DE EZZEL EGYÜTT HASZNOS. • Lehetne az eredetinek megfelelő is
Δl_{ELLEN} „**-1**” turnre	l_{ELLEN} (igazibol Pellen kell)	[-10, 10] (fok/turn)	• Δle a haladási irány változtatása, tekinthetjük „szög**sebesség**nek” is. Értékkészlet: 0 sebesség mellett, (+) óra járása, (-) órával ellentétes.
Δle-ÁTLAG q „**-t**” turnre	l_{ELLEN} (igazibol Pellen kell)	[-10, 10] (fok/turn)	• **mintázat…** oszcillálás! q - lásd: „átlagpozíció(k)”
ΔE „**-1**” turnre	E	[0,200] (energia)	• miért lehet hasznos? pl.: ha sok E-t veszít az ellenfél -> közvetetten többet lő (mert bízik abban, hogy jobban céloz) • **(vagy érdemes lehet külön „TALÁLAT” változót csinálni belőle, ami 0/1 !** Értékkészlet: maximum a max életét veszítheti el. (Ez praktikusan nagy túlzás.)
x_tav	Psajat	[16,400] (px)	• a kisebb távolság a bal és jobb oldali falaktól
y_tav	Psajat	[16,300] (px)	• a kisebb távolság a fenti és lenti falaktól • ellenfelnel is lehetne ugyanilyen
α2 „**db**” lövésre visszamenőleg	LŐTT Bear „alfa”	[0,180/1000] (px/fok)	• mennyire lőttünk elé/mögé (fok/távolságban) lőtt a becsülthöz képest. Értékkészlet: max: 180 fokkal „mellé”, max távolságnál. Mivel már mindig rá lövünk
„**epszilon**” „**db**” lövésre visszamenőleg	LŐTT Bear Pe		• mennyire elé/mögé (nem is fokban, nem is távolságban) lőtt a jelenlegi helyéhez képest. • On-Head Targeting van
heat	Heat	[0,20]	• érdemes lehet tartalékolni, vagy több kicsit lőni egy nagy helyett, stb.
Esajat			•
Eellen			•

Döntéshozás, következtetés

4 fában

A tudásbázisban lévő értékkészletek alapján lehet meg határozni, hogy milyen jellegű belső csomópontokat szeretnénk belőlük összerakni. (>,<, …. ???)

Paraméter megszorítások – a keresés meggyorsítására, tervezői időben.

Döntéshozó függvény - mindegyik bemenete egy Hiedelem és egy szám	Szükséges tudás	típus	Paraméter megszorítások
LŐTT_SZÁM_SAJÁT (<=, >)	**LŐTT_DB_SAJÁT**	int	[0,8] (darab)
TALÁLT_DB (<=, >) p	**ELTALÁLTAM_DB**	int	{0,1,2,3} (darab)
ELTALÁLT_DB (<=, >) p	ELTALÁLT_DB	int	{0,1,2,3} (darab)
alfa_ele_celzas (<=, >)	**α**	double	[0,0.2185] (fok/ turn)
ele_celzas(<=, >)	**v/d**	double	[0, 0.5] 8/16? (1/turn)
sebesseg_ellen_atlag(<=, >)	**$V_{ELLEN-ÁTLAG}$**	double	[0,8] (px/turn)
gyorsulas_ellen(<=, >)	**A_{ELLEN}**	double	[-2,1] (px/turn^2)
gyorsulas_ellen_atlag(<=, >)	**$A_{ÁTLAG}$**	double	[-2,1] (px/turn^2)
sebesseg_sajat_atlag(<=, >)	**$V_{SAJÁT-ÁTLAG}$**	double	[0,8] (px/turn)
pozicio_ellen_atlag(<=, >)	**$P_{ELLEN-ATLAG[]}$**	double, double	x= [0,800](px) y= [0,600](px)
kozeledes(=közeledik?)	**Δd**	0,1	[-8,8] (px)
irany_valtozas_ellen(<=, >)	**Δe**	double	[-10, 10] (fok/turn)
irany_valtozasatlag_ellen(<=, >)	**$Δe_{ÁTLAG}$**	double	[-10, 10] (fok/turn)
energia_valtozas_ellen(találat: {0,1})	ΔE – v. elég a találat?	double	[0,200] (energia)
x_tavolsag(<=, >)	**x_tav**	double	[16,400] (px)
y_tavolsag (<=, >)	**y_tav**	double	[16,300] (px)
beta_ele_celzas (<=, >)	**alfa2**	double	0,180/1000] (px/fok)
epsilon(<=, >)	**epsilon**	double	??

heat	heat	double	[0,20]
Esajat	Esajat	double	[0,120] – 120 prakt
Eellen	Eellen	double	[0,120] – 120 prakt
SebessegEllen	Vellen		
SebessegSajat	Vsajat		

Beavatkozás, cselekvés

4 független módon.

http://robowiki.net/wiki/Robocode/Game_Physics

Cselekvés	Értékkészlet	Komment
gunTurn	+-[0,20]	szög/ turn
bodyTurn	+-[0,4-10]	szög/ turn változó (**sebességfüggő)**
fire	{0,1,2,3}	
accelerate/decelerate	{-2,-1,-0.5,0,0.3,0.6}	