Universitat Autònoma de Barcelona

escola d'enginyeria

5126-1: Disseny automàtic d'una xarxa neuronal artificial per a l'Unreal Tournament

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica

Realitzat per:
Joan Marc Llargués Asensio
Dirigit per:
Juan Peralta Donate i Antonio López Peña
Bellaterra, 4 de Juliol de 2013

Universitat Autònoma de Barcelona

escola d'enginyeria

Els sotasignats, **Juan Peralta Donate i Antonio López Peña**
Professors de l'Escola Tècnica Superior d'Enginyeria de la UAB

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en **Joan Marc Llargués Asensio**.

I per tal que consti firmen la present.

Signat: **Juan Peralta Donate i Antonio López Peña**
Bellaterra, 4 de Juliol de 2013

Agraïments

Voldria agrair als tutors del projecte de final de carrera, Juan Peralta i Antonio López per la seva paciència, els seus consells, l'ajuda rebuda i el bon ambient de treball que han aconseguit generar.

També voldria donar un fort agraïment a la meva família i als meus amics pels ànims i per la infinita paciència que han tingut al llarg del desenvolupament del projecte.

Índex de continguts

Índex d'imatges

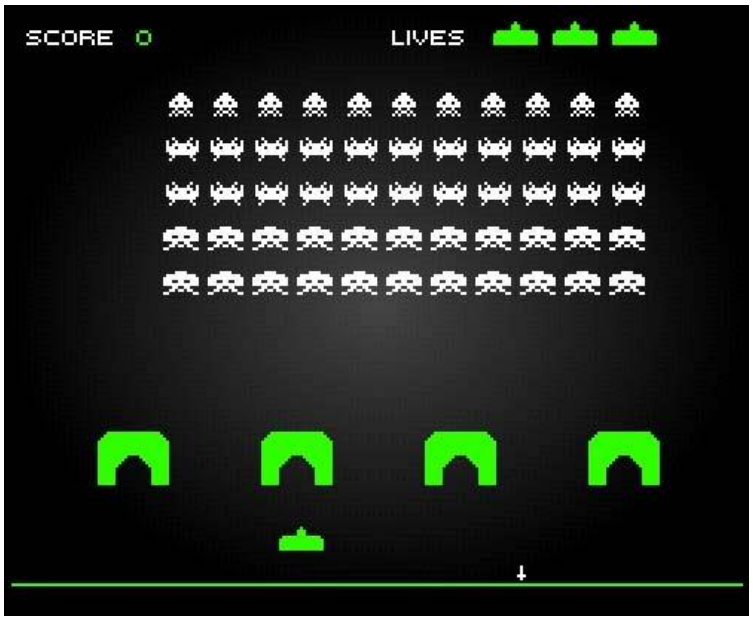
Índex de figures

Índex de taules

Índex de codi

1. Introducció

El món dels videojocs és un món que cada dia esta més present en les nostres vides. Començant amb les primeres videoconsoles fa més de 50 anys, els videojocs han anat evolucionant, com es pot veure a les imatges 1 i 2, de forma exponencial en cada nova iteració d'aquestes. S'han millorat els motors gràfics, de físiques, il·luminació,...



Imatge 1: videojoc Space Invaders (1978)

La intel·ligència artificial aplicada al món dels videojocs sempre s'ha basat en algorismes que intenten contemplar totes les opcions possibles en les que un jugador es pot trobar, ja sigui per algorismes de decisió[1], pathfinders[1], etcètera. Això planteja un seguit de qüestions importants com ara, volem que el jugador s'ho passi bé i gaudeixi del joc? Si és així, el nivell de dificultat ha de ser assequible però alhora creïble, per a que sigui creïble, la intel·ligència artificial ha de ser prou bona com per fer pensar al jugador quin serà el seu proper moviment basat en accions que pugui realitzar en aquella situació, i per tant això implica fer que el joc passi a ser difícil per a l'usuari ja que implica crear una intel·ligència realista que s'enfrontarà a un jugador sense experiència.

[1] Algorisme que permet trobar un camí entre dos punts que estiguin connectats.

Imatge 2: videojoc Battlefield 4 (2013)

És cert que durant tot aquest temps s'han anat perfeccionant les tècniques d'intel·ligència artificial però sempre s'han millorat amb la idea de fer passar una bona estona al jugador i no pas de plantejar-li realment un repte. Per aquest motiu que s'han de buscar noves maneres d'implementar i satisfer les necessitats a les que els jugadors d'avui dia es troben sense renunciar al realisme.

1.1 Objectius

El projecte consisteix en una primera presa de contacte amb el món de la intel·ligència artificial aplicada als videojocs. Es dissenya un cervell artificial mitjançant xarxes neuronals per donar vida als bots[2] del videojoc Unreal Tournament 2004[2] per ordinador.

Les proves del projecte es duran a terme en un servidor dedicat allotjat a la Universitat Autònoma de Barcelona. Els objectius s'han dividit en dos grups: principals i secundaris. Els objectius principals són:

- Estudi i comprensió de les xarxes neuronals artificials[3] [4]. Entendre com funcionen les xarxes neuronals i com en podem obtenir el millor resultat sense haver de sacrificar rendiment.

[2] Personatge no controlable pel jugador i que només pot ser controlar pel programa.

- Aplicar tècniques d'intel·ligència artificial[5]. Hi ha un conjunt de decisions que la xarxa neural no pot decidir per ella mateixa i s'han implementat un conjunt d'algorismes per ajudar-la en aquestes tasques.
- Aprenentatge amb el Pogamut[6]. Aprendre tot el funcionament del Pogamut, com es comunica amb el joc, quines classes té i quines són les mes importants, quins mètodes són els més útils i quins ens faran falta al llarg del projecte.
- Dur a terme un sistema d'intel·ligència artificial. El resultat de tots els objectius anteriors és crear una intel·ligència artificial que es pugui fer passar per una persona en la manera de jugar.

D'altra banda, el desenvolupament d'aquest projecte pretén cobrir els següents objectius secundaris:

- Participar al campionat de Saragossa. Es realitzarà un campionat a Saragossa al maig de 2014 on participaran diversos projectes relacionats amb la intel·ligència artificial i el grau d'humanitat que es pot aconseguir aplicant diversos mètodes.
- Comportament humà aplicat a casos reals. Comprovar quines accions realitzen els bots segons la situació en la que es trobin per determinar si el que estan aprenent els bots estan bé o no.
- Entrenament amb gent real. Un cop els bots hagin entrenat entre ells, s'entrenaran amb gent real, ja sigui amb o sense experiència en aquests tipus de jocs, per tal de que la xarxa s'enfronti a un altre tipus de jugadors.

1.2 Organització de la memòria

Finalitzada la introducció, la memòria es desenvolupa en els capítols següents segons es descriu a continuació.

En el proper capítol, el segon, es realitza un estudi de viabilitat del projecte per comprovar si hi ha limitacions a l'hora de poder-lo realitzar i, en cas d'haver-n'hi, quines vies alternatives podríem fer servir.

Al tercer capítol veurem què són les xarxes neuronals artificials, com funcionen i com les fem servir en el nostre sistema.

Els següents dos capítols, quart i cinquè, s'analitza la metodologia seguida a l'hora d'estructurar el projecte i la programació d'aquest.

Posteriorment, al sisè capítol, es presentaran els resultats obtinguts de l'experimentació.

Per acabar, els capítols setè i vuitè presenten, respectivament, les conclusions del projecte i quines millores i ampliacions s'hi podrien aplicar per millorar la funcionalitat d'aquest.

2. Estat de l'art

La diferència entre aquest projecte i d'altres que fan ús de xarxes neuronals resideix en el factor humà. Avui dia la indústria del videojoc no fa servir xarxes neuronals per treballar amb la intel·ligència artificial ja que els resultats obtinguts sempre han sigut massa perfectes i l'interès desapareix si el jugador s'ha d'enfrontar a algú a qui no pot guanyar, però tot i així hi ha projectes alternatius basats en xarxes neuronals artificials que un cop aplicades als videojocs, donen uns resultats bastant satisfactoris.

L'objectiu que es pretén aconseguir en aquest projecte és que la xarxa neuronal jugui com si fos un ésser humà, es a dir, cometi errors però sempre actuant d'acord amb el que pot fer en tot moment i sense les restriccions de dificultat basades en el nivell de la IA.

A continuació es veurà com funcionen les xarxes neuronals així com els avantatges que presenten a l'hora de tractar amb elles, quins algorismes genètics[7] hi ha a l'actualitat, exemples de projectes basats en xarxes neuronals aplicats al món dels videojocs i l'estudi de viabilitat del projecte.

2.1 Xarxes neuronals artificials

Les xarxes de neurones artificials[4] (denominades habitualment com RNA o en anglès com "ANN") són un paradigma d'aprenentatge i processament automàtic inspirat en la forma en què funciona el sistema nerviós dels animals. Es tracta d'un sistema d'interconnexió de neurones en una xarxa que col·labora per produir un estímul de sortida.

Les xarxes neuronals consisteixen en fer una simulació de les propietats observades en els sistemes neuronals biològics a través de models matemàtics recreats mitjançant mecanismes artificials (com un circuit integrat, un ordinador o un conjunt de vàlvules). L'objectiu és aconseguir que les màquines donin respostes similars a les que és capaç de donar el cervell humà que es caracteritzen per la seva generalització i la seva robustesa.

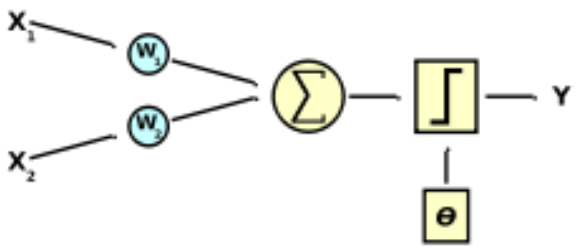


Figura 1: neurona amb dues entrades

Una xarxa neuronal es compon d'unitats anomenades neurones (veure figura 1). Cada neurona rep una sèrie d'entrades a través d'interconnexions i emet una sortida. Aquesta sortida ve donada per tres funcions:

- Una funció de propagació (també coneguda com funció d'excitació), que generalment consisteix en el sumatori de cada entrada multiplicada pel pes de la seva interconnexió (valor net). Si el pes és positiu, la connexió s'anomena *excitadora* , si és negatiu, s'anomena *inhibitòria*.
- Una funció d'activació, que modifica l'anterior. Pot no existir, en aquest cas, la sortida és la mateixa funció de propagació.
- Una funció de transferència, que s'aplica al valor retornat per la funció d'activació. S'utilitza per a limitar la sortida de la neurona i generalment ve donada per la interpretació que vulguem donar-li a aquestes sortides. Algunes de les més utilitzades són la funció sigmoide (per obtenir valors en l'interval [0,1]) i la tangent hiperbòlica (per obtenir valors en l'interval [-1,1]).

2.1.1 AVANTATGES

Les xarxes neuronals artificials tenen molts avantatges pel fet que estan basades en l'estructura del sistema nerviós, principalment el cervell:

- **Aprenentatge :** Les xarxa neuronals tenen l'habilitat d'aprendre mitjançant una etapa que s'anomena *etapa d'aprenentatge* . Aquesta consisteix en proporcionar a la xarxa neuronal dades com a entrada mentre se li indica quina és la sortida (resposta) esperada.
- **Auto organització :** Una xarxa neuronal crea la seva pròpia representació de la informació al seu interior, descarregant l'usuari d'això.

- **Tolerància a fallades :** Atès que una xarxa neuronal emmagatzema la informació de forma redundant, aquesta pot seguir responent de manera acceptable, fins i tot, si es fa malbé parcialment.
- **Flexibilitat :** Una xarxa neuronal pot gestionar canvis no importants en la informació d'entrada, com senyals amb soroll o altres canvis en l'entrada (ex. si la informació d'entrada és la imatge d'un objecte, la resposta corresponent no pateix canvis si la imatge canvia una mica la seva brillantor o l'objecte canvia lleugerament.
- **Temps real :** L'estructura d'una xarxa neuronal és paral·lela, per la qual cosa si això és implementat en ordinadors o en dispositius electrònics especials, es poden obtenir respostes en temps real.

2.1.2 INCONVENIENTS

Tot i els avantatges que presenten les xarxes neuronals a l'hora de solucionar certs problemes, també hi ha certs inconvenients que cal tenir en compte.

La capacitat de les xarxes neuronals resideix en la seva habilitat per processar informació en paral·lel i tractar conjunts molt grans d'informació. Malauradament, dels ordinadors d'avui dia no s'aprofita gaire bé la capacitat de realitzar càlculs de forma paral·lela, és per això que moltes vegades es deixa fora de les solucions viables l'anàlisi amb xarxes neuronals pel temps que comportaria obtenir una bona solució.

El fet que una xarxa neuronal trigui més a resoldre un problema no depèn exclusivament del maquinari on s'està executant, també depèn d'altres factors com ara el nombre de patrons a identificar o bé si requereix d'una major flexibilitat o capacitat d'adaptació per a reconèixer patrons que siguin molt semblants. Com més semblants siguin els patrons, més temps s'haurà d'invertir en entrenar a la xarxa neuronal per a que els valors dels pesos convergeixin i representin el que es vol ensenyar.

La complexitat d'aprenentatge per a grans tasques augmenta a mida que es necessita que la xarxa neuronal aprengui coses noves.

La xarxa neuronal per sí mateixa proporciona uns valors de sortida que ella mateixa no pot interpretar, és per això que es requereix d'un programador i de l'aplicació en sí per tal de trobar un significat a la sortida que rebuda.

Malgrat els inconvenients que presenta la xarxa neuronal, s'ha optat per treballar amb elles ja que compleixen el propòsit principal del projecte, crear una entitat que aprengui per ella mateixa a jugar i a cometre errors humans.

2.2 Algorismes genètics

És gràcies a en John Henry Holland que, durant els anys 1970, va iniciar una de les línies més interessants en el camp de la intel·ligència artificial que ara tenim els algorismes genètics. Els algorismes genètics s'anomenen així ja que s'inspiren en la evolució biològica i la seva base genètic-molecular.

Aquests algorismes fan evolucionar una població d'individus que són sotmesos a accions aleatòries, com la mutació o la recombinació genètica, semblants a les que actuen a l'evolució biològica. A aquests tipus d'algorismes també s'hi pot aplicar la selecció natural en base a algun criteri, establert amb anterioritat, el qual decideix quin són els millors individus per tal de fer-los sobreviure i quins són els menys aptes per tal de descartar-los.

Els algorismes genètics són, bàsicament, mètodes de cerca dirigida basats en probabilitat, es a dir, envers una condició molt dèbil es pot demostrar que els algorismes convergeixen a trobar el valor òptim, per tant, al augmentar el nombre de poblacions generades pels algorismes genètics la probabilitat de trobar la millor població per al cas a tractar tendeix a 1.

2.2.1 FUNCIONAMENT D'UN ALGORISME GENÈTIC

Un algorisme genètic[8][9] es presenta, essencialment, seguint un conjunt de passos ben diferenciats:

- **Inicialització:** es genera aleatòriament una població inicial que esta constituïda per un conjunt de cromosomes que representen ser les possibles solucions al problema.
- **Avaluació:** a cadascun dels cromosomes de la població se li aplicarà una funció per saber com de bona és la solució a la que s'ha arribat.

- **Selecció natural:** un cop es comprova l'aptitud de cada cromosoma, es seleccionen els cromosomes que seran creuats per a generar la propera generació. Com més apte és un cromosoma, més possibilitats té de ser seleccionat.
- **Recombinació genètica:** representa la reproducció sexual entre dos cromosomes, això genera dos cromosomes descendents on s'ha combinat la informació genètica dels dos pares.
- **Mutació:** modifica aleatòriament part dels cromosomes dels individus i permet arribar a zones de l'espai de cerca que no estaven cobertes pels individus de la població actual.
- **Substitució:** es substitueix la població anterior per la nova població generada.

Un cop s'ha substituït la població es tornen a repetir tots els passos excepte el primer, la inicialització, ja que si es fes es perdria la nova població obtinguda. A la figura 2 podem observar el diagrama flux d'execució de l'algorisme genètic.

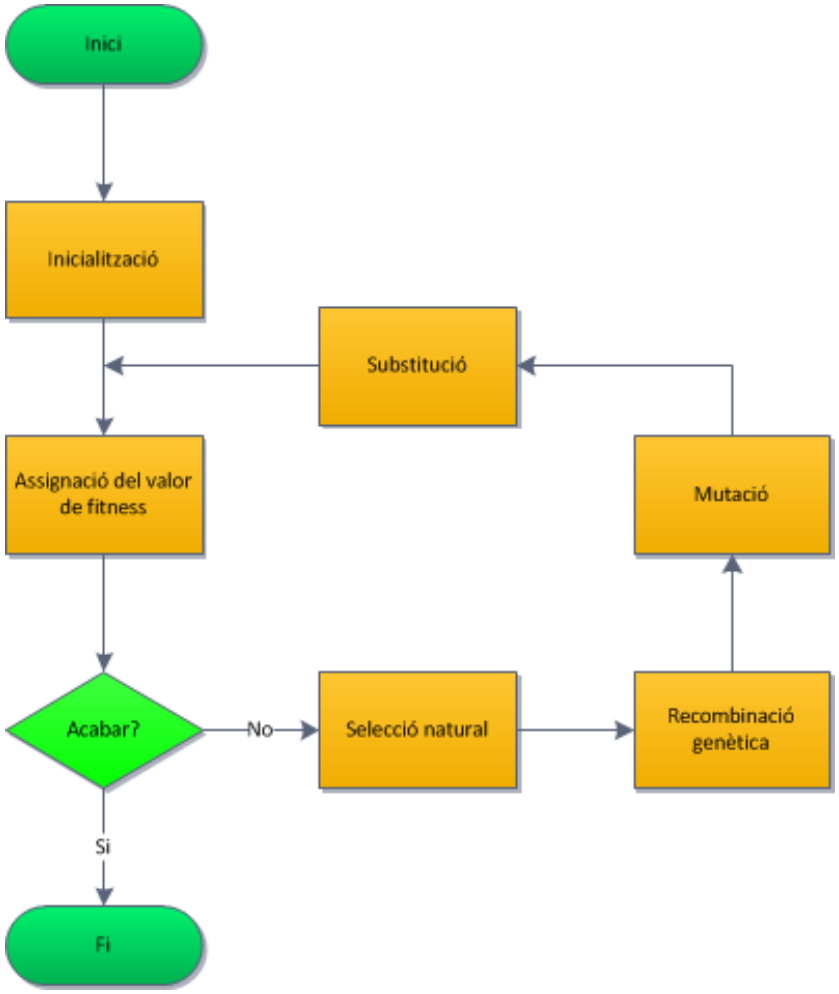


Figura 2: flux d'execució d'un algorisme genètic

2.3 Exemples aplicats de xarxes neuronals al món dels videojocs

A continuació es presenten tres exemples on les xarxes neuronals prenen el control dels personatges per desenvolupar accions que els permeten realitzar les seves tasques.

2.3.1 Super Mario Bros

Una xarxa neuronal combinada amb algorismes genètics es entrenada per superar nivells aleatoris al joc del Super Mario Bros[10] (veure imatge 3). El personatge disposa d'una matriu invisible de NxN quadres que es correspon amb el seu camp de visió, tot el que hi ha fora de la matriu no existeix. Les entrades de la xarxa són totes les accions que pot realitzar, es a dir: si pot saltar, si pot disparar, si hi ha un forat, etcètera.

Per la banda de les sortides hi ha les accions de saltar, anar a l'esquerra, anar a la dreta, disparar/córrer. De totes aquestes sortides la xarxa n'activa unes en concret i s'avalua el *fitness* del personatge al llarg del nivell. Com més lluny arriba millor valor de *fitness* té.

La competició, per la seva banda, consisteix en un nombre de mapes no vistos prèviament pels participants i plens d'enemics on el personatge, en Mario, ha d'arribar el màxim de lluny possible. Com més lluny arribi més bona puntuació obtindrà. Hi ha altres mètodes per decidir el guanyador, com ara el nombre d'enemics eliminats o les monedes que s'han recollit, però només s'apliquen en cas que hi hagi empat en distància recorreguda.

Finalment la competició obliga a que els algorismes s'apliquin en temps real i no superin els 40ms de temps de resposta.

Imatge 3: videojoc Super Mario Bros (2003) amb la xarxa neuronal com a jugador

2.3.2 NEURONAL BOT

Oponent per a la modalitat deathmatch[3] en el videojoc Quake 2[4] (veure imatge 4) que fa servir xarxes neuronals per controlar les seves accions i algorismes genètics per entrenar la xarxa[11]. El bot és totalment autònom i no te comportaments pre-programats a excepció de la selecció de a quin enemic disparar en cas d'haver-n'hi més d'un per pantalla, en aquest cas es selecciona al més proper.

Imatge 4: videojoc Quake II (1997) amb la xarxa neuronal com a jugador

2.3.3 BOTPRIZE

Sota la premissa "Poden els ordinadors jugar com els humans?" es presenta la competició del Botprize[12]. Aquesta competició es basa en un test de Turing on els concursants presenten bots que són controlats per un ordinador i, juntament amb jugadors de veritat que fan la funció de jutges, es troben en mig d'una partida deathmatch mentre els jutges intenten esbrinar quins dels seus oponents són bots i quins són humans.

Per avaluar la humanitat d'un bot, els jutges etiqueten amb el botó esquerre o botó dret segons si es tracta d'un bot o d'un jugador respectivament. Si el jutge s'equivoca al etiquetar, aquest mor, en canvi si encerta qui mor és el personatge a qui s'ha etiquetat. Al acabar la ronda es determina si un personatge és humà o bot segons el nombre de votacions

[3] Modalitat de joc on els jugadors no tenen equips i lluiten els uns contra els altres.
[4] Videojoc semblant a la saga de l'Unreal Tournament.

que ha rebut i tenint en compte quantes vegades s'ha encertat al fer la votació i quantes s'han fallat.

De tots els participants cal destacar en Raúl Arrabales. Després d'anys d'estudi en termes de consciencia artificial i psicologia, va desenvolupar una nova tecnologia anomenada cera-cranium[13][14], basada en la integració de diferents components cognitius, que li va permetre guanyar el concurs del botprize l'any 2010 amb el Conscious-Robots, amb els resultats que es mostren a la taula 1.

Nom del bot	Humanitat %
Conscious-Robots	31.8182 %
UT^2	27.2727 %
ICE-2010	23.3333 %
Discordia	17.7778 %
w00t	9.3023 %

Taula 1: resultats de la competició del Botprize 2010

Tot i els bons resultats obtinguts durant la competició del 2010, a la competició que es va celebrar a l'any 2012, com es pot apreciar a la taula 2, va ser on es va assolir el màxim grau d'humanitat aconseguit fins ara, un 52.2%.

Nom del bot	Humanitat %
MirrorBot	52.2 %
UT^2	51.9 %
ICE-CIG2012	36.0 %
NeuroBot	26.1 %
GladiatorBot	21.7 %
AmisBot	16.0 %

Taula 2: resultats de la competició del Botprize 2012

2.4 Estudi de viabilitat

En aquest apartat es tracten el conjunt de tecnologies requerides per al desenvolupament del projecte i és comprova si són viables o bé s'han de buscar propostes alternatives.

2.4.1 NETBEANS

És un entorn de desenvolupament integrat lliure, fet principalment per al llenguatge Java. És el programari sobre el qual es programarà el software que interactuarà amb la xarxa neuronal i amb el joc.

Tot i que des de que es va començar el projecte han aparegut noves versions del Netbeans[16] i d'altres programes que s'explicaran a continuació, la versió que s'ha fet servir en aquest cas és la 7.1.2 ja que en el moment de realitzar la instal·lació es recomanava fer servir aquesta versió i no d'altres per possibles incompatibilitats i bugs[5].

2.4.2 UNREAL TOURNAMENT 2004

Unreal Tournament 2004 o també conegut com UT2004 (veure imatge 5) és un videojoc d'acció en primera persona, principalment orientat a l'experiència multijugador on l'objectiu depèn del mode de joc escollit, però independentment del mode seleccionat, sempre s'ha de fer ús de violència virtual.

S'ha escollit aquest videojoc ja que disposa d'una gran facilitat per afegir mods[6] i juntament amb el Pogamut es pot controlar el funcionament dels bots.

[5] Errors de programació que desemboquen en comportaments no previstos.
[6] Modificacions afegides a un videojoc per donar-li funcionalitats que no van ser contemplades de cara a la publicació del producte final.

Imatge 5: videojoc Unreal Tournament 2004

2.4.3 PERCEPTRÓ MULTICAPA

Com es mostra a la figura 3, el perceptró multicapa [16] [17]es tracta d'una xarxa neuronal artificial formada per múltiples capes, això permet resoldre problemes que no són linealment separables. El perceptró multicapa pot ser totalment o localment connectat, es a dir, en el primer cas cada sortida d'una neurona de la capa "i" és entrada de totes les neurones de la capa "i+1". Per contra, en el segon cas, cada neurona de la capa "i" és una entrada d'una sèrie de neurones de la capa "i+1". Aquesta sèrie de neurones s'anomena regió.

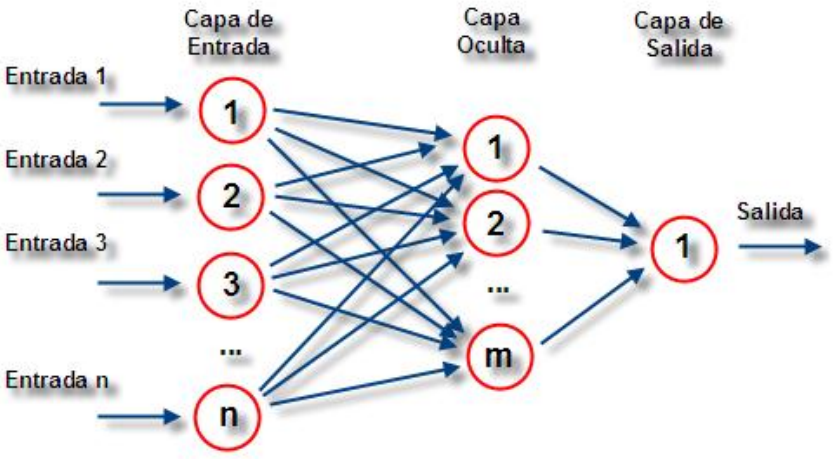


Figura 3: esquema d'un perceptró multicapa

Les capes es poden classificar en tres tipus diferents:

- Capa d'entrada. Aquesta capa esta formada per totes aquelles neurones que introdueixen els patrons d'entrada a la xarxa. Les neurones es dediquen a comunicar-se amb les de la capa oculta següent sense produir cap tipus de processat.
- Capa oculta. Formada per totes aquelles neurones les entrades de les quals provenen de capes anteriors i on les seves sortides van a capes posteriors. Aquestes neurones són les que realitzen el processament de les dades d'entrada. Si només hi ha una capa oculta, aquesta processa les entrades y envia el resultat a les sortides, si per el contrari hi ha més d'una capa oculta, cada una processa les dades que li entren i les envia a la següent capa per a que siguin processades. Això es repeteix fins que la capa a la que s'ha d'enviar la informació processada es la capa de sortida.
- Capa de sortida. Neurones les quals tenen el valor de sortida que es correspon amb els valors d'entrada un cop han sigut processats.

2.4.4 POGAMUT

El Pogamut és un middleware[7] que permet controlar agents virtuals[8] en múltiples entorns proporcionats per motors de joc. La API[9] que proporciona el Pogamut permet fer aparèixer i controlar agents virtuals i proporciona una GUI[10] en forma de plugin pel Netbeans que simplifica la feina a l'hora d'arreglar els errors.

L'objectiu principal és el de simplificar l'apartat de creació de l'agent. Moltes de les accions relacionades amb el medi, encara que siguin complicades de realitzar com ara el pathfinding o la recopilació d'informació dels agents en memòria poden ser executades amb una o dues comandes. Això permet que l'usuari centri els seus esforços en les parts interessants a desenvolupar.

[7] Software que fa de pont i comunica dos softwares diferents.
[8] Bots
[9] Interfície de programació per aplicacions. Conjunt de funcions i procediments que ofereix una biblioteca per a ser utilitzada per un altre software com a capa d'abstracció.
[10] Conjunt d'imatges i objectes gràfics per representar informació i accions disponibles en una interfície.

2.4.5 CONCLUSIONS DE VIABILITAT

Un cop realitzat l'estudi de viabilitat del projecte es pot decidir la seva viabilitat. Si es decideix que no és viable s'hauran de buscar alternatives i realitzar un nou estudi des de l'inici, d'altra banda si és viable es durà a terme.

Per una banda s'ha realitzat un estudi del programari que es necessitarà per dur a terme el projecte. Dins de l'àmplia gamma de programari destinat a crear aplicacions en java, al final es va optar a fer servir l'aplicació Netbeans ja que era una condició indispensable per fer servir el Pogamut i poder comunicar-se amb el videojoc. Tots els programes disposen de llicència gratuïta a excepció de l'Unreal Tournament 2004 que requereix adquirir-ne una.

També s'ha analitzat quin tipus d'especificacions de hardware farien falta per realitzar les simulacions correctament. Després de realitzar algunes proves es va arribar a la conclusió que, qualsevol ordinador de sobretaula amb una bona capacitat de càlcul és suficient per dur a terme bones simulacions.

D'altra banda s'ha realitzat un estudi dels costos que suposa el desenvolupament del projecte i es conclou que són assumibles donat que els costos materials son mínims i el cost més elevat és el cost per persona/hora en el desenvolupament del projecte.

La planificació que s'ha realitzat dóna un calendari raonable encara que el marge per a possibles endarreriments és força ajustat.

Finalment i tenint en compte tots els estudis realitzat es decideix que el projecte és viable i es procedeix a la seva realització.

3. Anàlisis

Durant el desenvolupament del projecte s'han dut a terme tres anàlisis molt importants: el diagrama de flux d'execució del programa, la distribució del sistema de sensors i la distribució dels fitxers generats pel propi programa.

3.1 Diagrama de flux

El sistema de control del bot es basa en la crida successiva d'un conjunt de mètodes per tal d'inicialitzar el bot, sensors i objectes i posteriorment en la repetició del mòdul de lògica, encarregat de les accions del bot (veure figura 4). A continuació es detalla que fa cada mètode i quines característiques s'han implementat en el projecte.

- **Preparar bot:** es crida abans que el bot es connecti a l'entorn però després de que l'objecte UT2004Bot[11] sigui construït. En aquest mètode afegim els listeners[12] que necessitarem al llarg de l'execució.
- **Inicialitzar:** inicialitza les propietats inicials del bot tals com el bot, la posició inicial, etcètera. Quan es crida el mètode inicialitzem diversos objectes per gestionar el moviment del bot, la selecció d'enemic, les accions, nom i skin del bot, etcètera.
- **Inicialitzar bot:** mètode cridat quan el bot rep el missatge inited [13]per part del servidor. Aquest missatge implica que el mètode anterior s'ha realitzat satisfactòriament, el handshake[14] entre el bot i el servidor s'ha acabat i que el bot ja està preparat per rebre ordres.
- **Aparició del bot:** mètode que es crida quan el bot entra per primera vegada a la partida. Això implica que el bot ja té representació gràfica i es visible pels jugadors i un objecte de tipus self on es guarda informació sobre la posició del bot i l'estat actual es rebut de l'entorn. Durant aquest procés, quan el bot entra en joc, s'aprofita per enviar un missatge a tots els jugadors indicant que el bot esta viu, es guarda la informació relacionada amb el fil del procés que s'encarrega d'aquell bot per tal de crear una identificació única per a cada bot a l'hora de tractar-los conjuntament i, en cas de que

[11] Classe base que permet controlar el bot.
[12] Objecte que serveix per observar quan passa un esdeveniment.
[13] Missatge enviat pel servidor per indicar que s'ha realitzat la connexió i que aquesta ha sigut acceptada.
[14] Procés entre dos entitats per establir els paràmetres per al canal de comunicació.

l'execució llançada estigui en mode generació, es crea una carpeta pel bot així com els fitxers per a la xarxa neuronal i es netegen fitxers que podrien contenir informació sobre execucions anteriors del bot.

- **Abans de la lògica:** la crida es realitza just després de la primera aparició del bot i just abans de començar amb la part lògica. Aquest mètode és ideal per fer les últimes preparacions dels mòduls personalitzats que s'hagin pogut implementar ja que en aquest cas el bot esta totalment inicialitzat i dins de l'entorn
- **Lògica:** mètode que es crida periòdicament amb el fil associat al bot per tal de realitzar accions.
- **Bot mort:** mètode que es crida cada vegada que un bot mor.

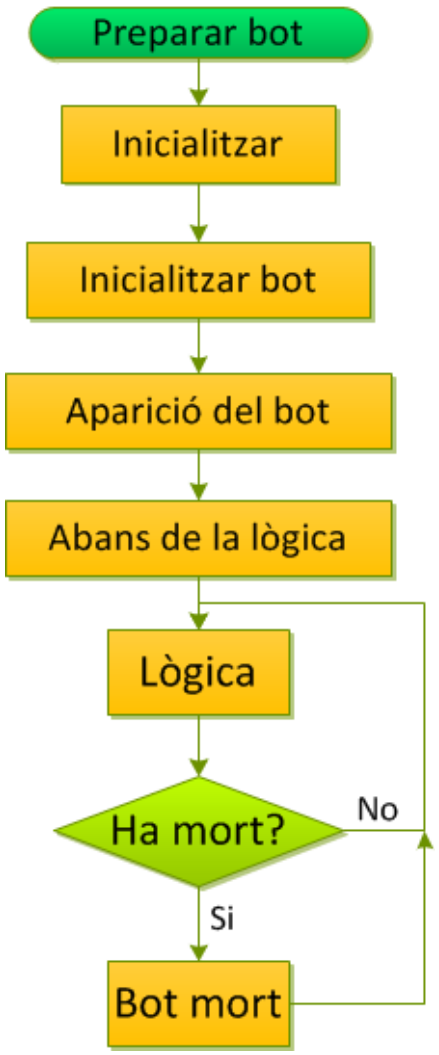


Figura 4: diagrama de flux de l'execució

3.2 Sensors de navegació

El conjunt de sensors implementats i assignats al bot serveixen per detectar col·lisions amb l'escenari. Es pot assignar qualsevol nombre de sensors al bot, però donat que cada sensor implica augmentar substancialment la càrrega de treball de la cpu, s'havia de trobar el nombre adequat de sensors per arribar a un equilibri entre rendiment i eficiència.

Primerament es van tractar els sensors que s'encarregarien de detectar xocs amb objectes que estiguessin a l'alçada de la cintura del bot ja que és des d'on surten tots els sensors. Inicialment, i com es pot comprovar a la figura 5, es van realitzar proves amb cinc sensors, un cap endavant, dos a 45º i dos més a 90º. Els resultats no eren gaire satisfactoris, el bot navegava si, però es podia donar el cas de topar-se amb un objecte que càpigues entre els sensors i que cap d'ells detectes col·lisió.

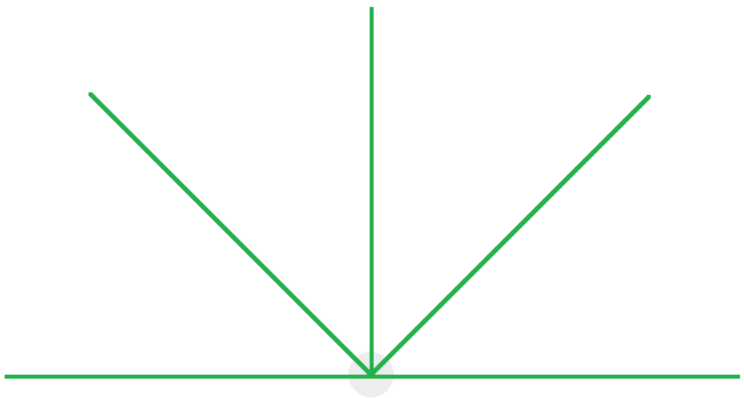


Figura 5: vista aèria del bot amb cinc sensors

Amb més experimentació es va arribar a la conclusió que, la millor distribució de sensors era la de tenir-ne un cada 30º. Amb aquesta distribució s'obtenia un bon resultat a l'hora de detecció de col·lisions ja que si un objecte en el cas anterior era detectat pel sensor de 45º, en aquest cas el mateix objecte hauria de ser detectat per dos sensors que es troben a 30º i 60º graus respectivament (veure figura 6). D'aquesta manera s'aconsegueix millorar la detecció de col·lisions, sobretot, amb objectes petits, ja que de l'altra manera si un dels sensors indicava col·lisió es podia tractar tant d'una paret, d'una caixa gran o d'una caixa petita, i no es podia saber. En canvi amb el sistema actual, segons el nombre de sensors que s'activen es pot determinar el tamany aproximat de l'objecte.

Una altra de les avantatges que presenta la distribució seleccionada dels sensors horitzontals es la d'aconseguir un moviment mes suau a l'hora de realitzar girs. En el primer cas teníem que els girs només podien ser de 45º o 90º i en aquest els girs poden ser de 30º, 60º i 90º, de manera que s'aporta més naturalitat a l'hora de realitzar els moviments.

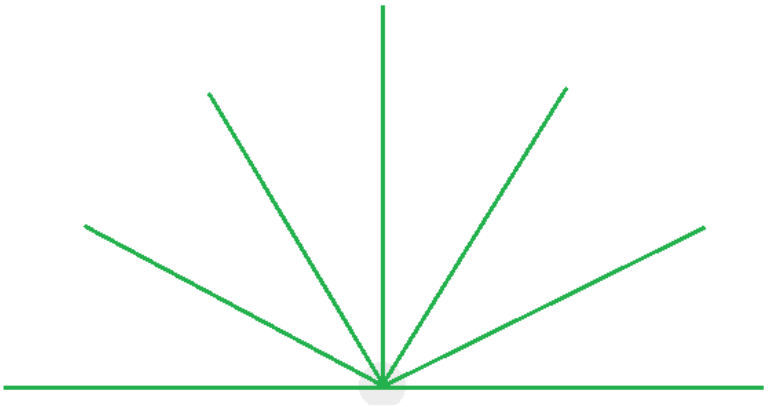


Figura 6: vista aèria del bot amb set sensors

També s'havia de tractar, com mostra la figura 7, el problema de les escales, rampes, i desnivells en sentit ascendent. Tots aquests tipus de construccions fan que el fet de només tenir un sensor apuntant cap endavant detecti col·lisió quan es trobi amb alguna d'aquestes situacions tot i que el bot pugui seguir movent-se. Per aquest motiu es va decidir afegir dos sensors més que ajudessin al sensor frontal a determinar si realment es pot o no es pot avançar. Aquests sensors, juntament amb el frontal, cobreixen tota l'alçada del bot, de manera que si dos sensors detecten col·lisió i el tercer no, vol dir que ens trobem davant d'unes escales, rampa o fins i tot una escletxa per on pot passar el bot, llavors ja es decisió de la xarxa neuronal decidir si vol seguir per aquest camí o anar per una altra banda.

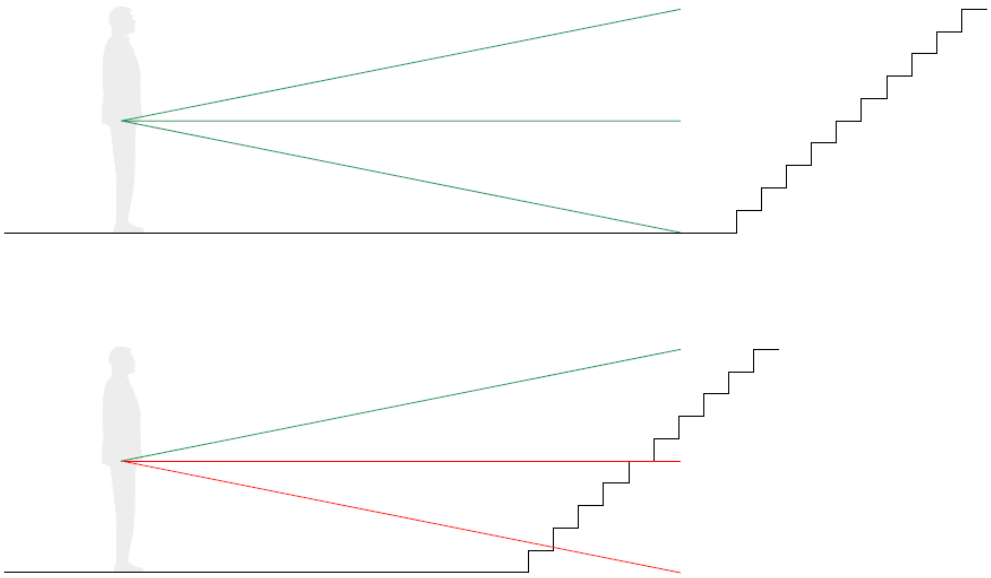


Figura 7: vista de perfil amb els sensors d'alçada

Similar al cas anterior però amb desnivells en sentit descendent, tenim el problema de que quan el bot es trobi davant d'un barranc, cap dels sensors anterior detectaria la situació i per tant el bot seguiria avançant. Per resoldre aquest problema s'han afegit dos sensors més que apunten cap endavant però la seva distància de col·lisió és inferior, aquests sensors serveixen per veure si hi ha terreny o no just davant del bot o si pot saltar sense xocar amb el sostre u obstacle situat per damunt seu. La diferència que fa particular aquests sensors és que el sensor que detecta si hi ha superfície per on caminar esta sempre detectant col·lisió.

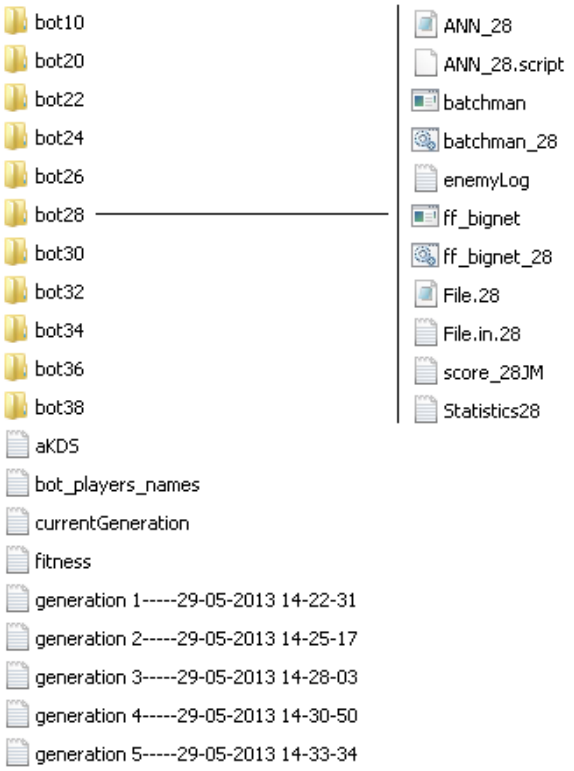
3.3 Fitxers

L'execució del programa genera una quantitat significativa de fitxers que s'han d'emmagatzemar de manera correcta ja que cada bot té els seus fitxers personals que no es poden barrejar amb els d'altres bots.

Per gestionar els fitxers s'ha optat per tenir una carpeta arrel fixa on aniran totes les carpetes i fitxers (veure imatge 6). Cada bot disposa d'una carpeta on es guardarà tota la seva informació: puntuació, estadístiques, enemics i cromosomes. El nom d'aquestes carpetes s'assigna en funció de la identificació de cada bot ja que és única, així doncs, la

carpeta per un bot que tingui com a identificador el valor 10, la seva carpeta s'anomenarà "bot 10", i així successivament per a tots els bots.

Dins de la carpeta arrel, apart d'haver-hi les carpetes dels bots, hi ha fitxers addicionals. Aquests fitxers contenen una copia de tota la població de cromosomes de la generació actuals abans de que se'ls hi apliquin els algorismes genètics. També hi ha fitxers que s'encarreguen d'emmagatzemar el *fitness* de la població actual i dos fitxers de control, un que emmagatzema el valor de generació en el que es troba l'execució i un altre on es guarden el nom que tenen els bots durant la partida.



Imatge 6: distribució de carpetes I fitxers dins de la carpeta arrel

4. Disseny

A l'apartat del disseny del projecte es veuen les classes del projecte, addicionalment, a l'apartat de l'annex es troba tota la documentació detallada relacionada amb el javadoc del projecte. Així mateix, dins del present apartat hi ha el diagrama de classes on s'indica com estan relacionades les unes amb les altres.

4.1 Classes

La classe *smartbot* és el nucli del projecte, és on es realitzen les crides per inicialitzar i configurar el bot, així com les crides per crear els fitxers de dades i implementar tota la lògica del bot.

Les variables que es poden apreciar a la figura 8 formen part de les variables que permeten configurar l'execució del programa, la resta que no apareixen en aquesta taula són variables per al correcte funcionament de la classe.

Pel que fa a les funcions, hi ha les corresponents al diagrama de flux de l'execució (veure figura 4) com la *prepareBot(...)* o la *logic().* La resta de funcions permeten reiniciar les variables d'entrada i sortida, *reset_in_ANN()* i *reset_out_ANN()* respectivament, obtenir un nom per al bot amb *nameBot()* o indicar amb *seeIncomingProjectile()* si s'acosta un projectil.

SmartBot
-root : string = Executions\\ -num_bots : int = 4 -input_ANN : int = 48 -hidden_ANN : int = input_ANN*2 -output_ANN : int = 46 -mode : string = Play -serverIp : string = localhost -serverPort : int = 3000
+Main() : void +prepareBot(entrada bot : UT2004Bot) : void +getInitializeCommand() : Initialize +botInitialized(entrada gameInfo : GameInfo, entrada currentConfig : ConfigChange, entrada init : InitedMessage) : void +botFirstSpawn(entrada gameInfo : GameInfo, entrada config : ConfigChange, entrada init : InitedMessage, entrada self : Self) : void +beforeFirstLogic() : void +botKilled(entrada event : BotKilled) : void +logic() : void +reset_in_ANN() : void +reset_out_ANN() : void +prepare_in_ANN() : string +ANN() : void +set_out_ANN(entrada out_ANN : double) : void +smartbot_does() : void +getItem(entrada _distance : int, entrada _object : ItemType) : Item +seeIncomingProjectile() : bool +pickProjectile() : IncomingProjectile +seeEnemies() : bool +nameBot() : string +getSkin() : string

Figura 8: classe *SmartBot* amb algunes de les variables més importants

Functions, com s'aprecia a la figura 9, és una classe genèrica on es defineixen tots els mètodes suport com ara quina arma s'ha de seleccionar, si el bot visualitza un tipus d'objecte en concret o si aquest esta dins del radi d'acció del bot, etcètera.

Algunes d'aquestes funcions es podrien haver afegit a la classe *smartBot* però es va optar per afegir-les a una classe genèrica per tal de no saturar i mantenir el més ordenat possible la classe principal.

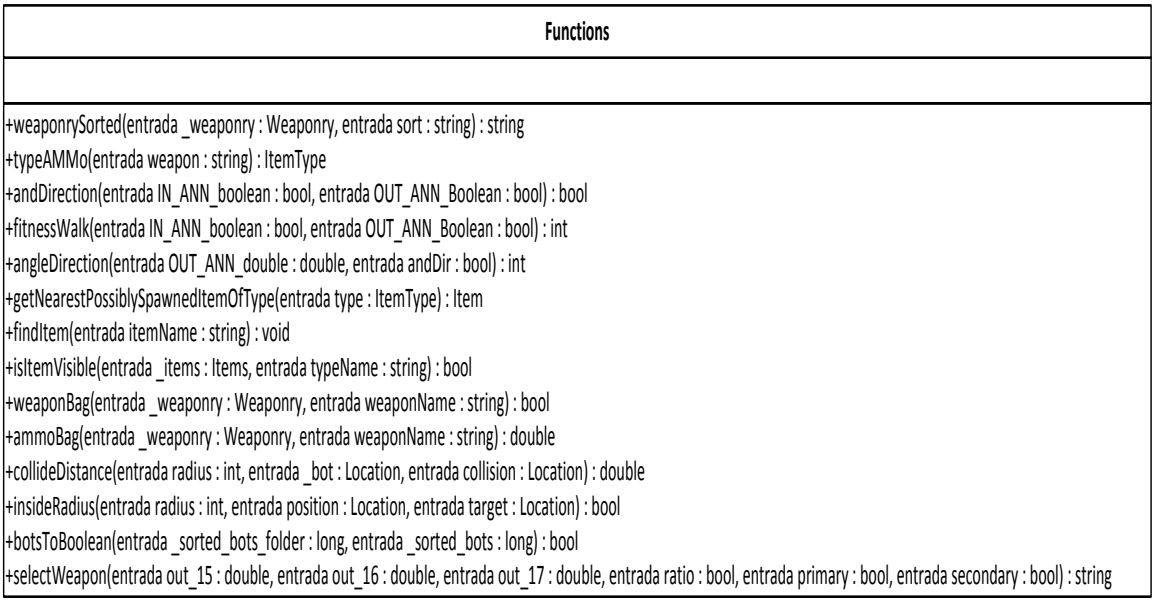


Figura 9: classe *Functions*

El *fileManager*(veure figura 10) s'encarrega de gestionar totes les sol·licituds relacionades amb els fitxers, ja sigui llegir, escriure, eliminar o obtenir informació d'algun d'ells.

Inicialment es va plantejar deixar les funcions encarregades de gestionar el fitxers a la classe principal però a mida que el projecte avançava i s'havien de crear més funcions, es va decidir crear una classe especialitzada només a tractar amb els fitxers.

Les funcions més importants d'aquesta classe són les de crear les carpetes de tots els bots, *create_folder(...)*, i crear-hi els fitxers necessaris per començar l'execució, *prepareThread(...)*. Sense funcions com *Init_ANN_creator(...), Init_ANN_Parser(...)* i *new_generations_file(...),* no es podria dur a terme l'execució de la xarxa neuronal i els algorismes evolutius ja que el primer crea el fitxer amb els pesos de la xarxa, el segon els llegeix i els guarda en un vector i el tercer s'encarrega d'aplicar la selecció natural, creuament i mutació a les xarxes per crear noves poblacions i substituir les velles.

La resta de funcions serveixen per automatitzar la neteja de fitxers quan es vol començar a entrenar una població nova i per guardar dades de cada un dels individus com ara la seva puntuació, baixes, morts i suïcidis entre altres.

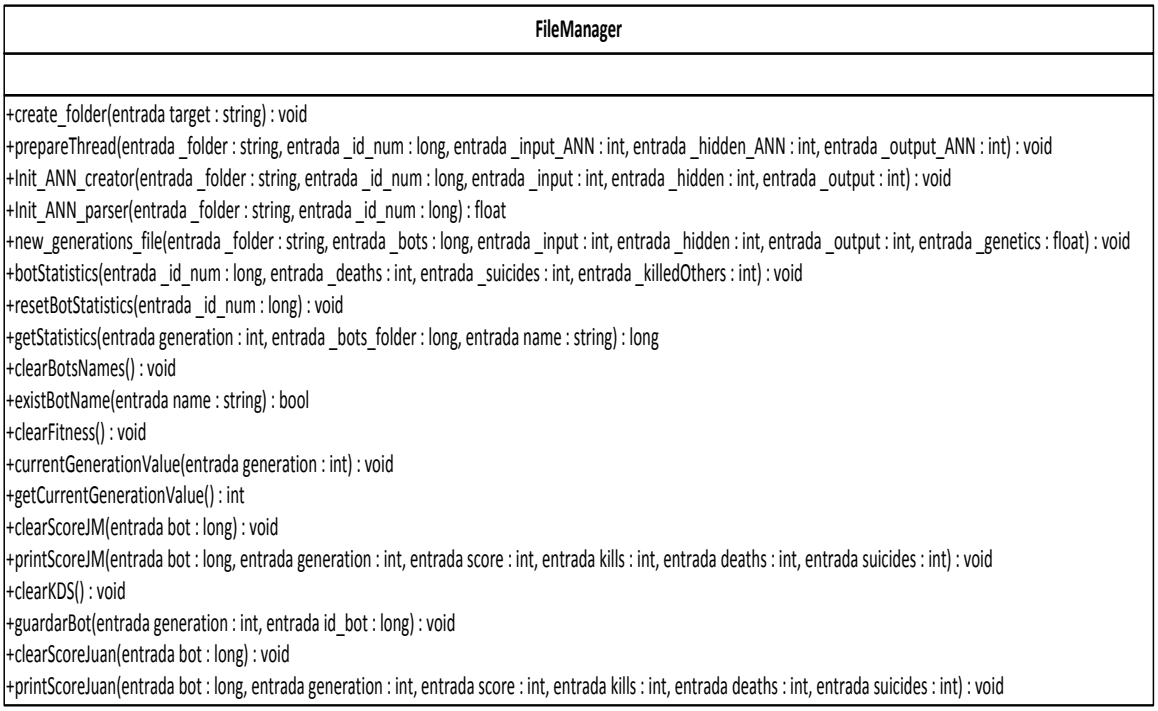


Figura 10: classe *FileManager*

Movement, mostrat a la figura 11, disposa de totes les funcions de moviment que el bot pot realitzar, saltar, caminar, evadir, etcètera.

Els moviments que s'han implementat es corresponen amb moviments senzills que el bot podria realitzar en qualsevol modalitat. Com a moviment complex es podria implementar una funció que permetés al bot navegar per tots els nodes del mapa però no tindria sentit donada la naturalesa del projecte.

Movement
-move : advancedLocomotion()
+jump() : void +doubleJump() : void +move() : void +stopMove() : void +turnHoritzontal(entrada angle : int) : void +strafeLeft(entrada distance : double) : void +strafeRight(entrada distance : double) : void +dodgeLeft(entrada botPosition : ILocated, entrada inFrontOfTheBot : ILocated) : void +dodgeRight(entrada botPosition : ILocated, entrada inFrontOfTheBot : ILocated) : void +dodgeBack(entrada botPosition : ILocated, entrada inFrontOfTheBot : ILocated) : void +dodge(entrada botPosition : ILocated, entrada inFrontOfTheBot : ILocated) : void

Figura 11: classe *Movement*

La classe *Actions* (veure figura 12) permet al bot disparar amb l'atac primari o secundari de l'arma que dugui equipada en el moment de disparar. Es poden realitzar més accions per part del bot com ara activar interruptors, entrar a un vehicle o recollir una bandera, però no s'han implementat perquè el mode de joc al que va destinat el bot no n'hi apareixen.

Les variables de les que consta la classe s'encarreguen de guardar qui és l'enemic a disparar a la variable *enemy,* les preferències d'ús d'armes amb weaponPrefs, totes les armes del bot i munició de cada una bot amb weaponry o, fins i tot, la precisió del bot a l'hora de disparar.

Actions
-enemy : Player = null -accuracy : double = 100 -shoot : ImprovedShooting -weaponPrefs : WeaponPrefs -weaponry : Weaponry
+resetEnemy() : void +setAccuracy(entrada accuracy : int) : void +shootPrimary(entrada _player : Player, entrada _weaponry : Weaponry, entrada _weaponName : string, entrada _incoming : bool, entrada _projectile : IncomingProjectile, entrada _info : AgentInfo) : void +shootSecondary(entrada _player : Player) : void

Figura 12: classe Actions

AGGen (veure figura 13) s'encarrega de totes les parts de l'algorisme genètic: elitisme, creuament i mutació.

La funció *genetics()* s'encarrega d'aplicar a un vector que conté tots els cromosomes de tots els bots, elitisme, guardant en un vector nou els millors individus. A continuació, amb el creuament, es generen fills entre tots els bots, incloent als bots seleccionats a la primera

fase. Finalment es crida a la funció *mutacio()* que modifica alguns dels cromosomes dels bots obtinguts per creuament.

AGGen
+generate_genetic_file(entrada _folder : string, entrada _generation : long, entrada _bots : bool) : string +genetics(entrada file : string, entrada _bots_folder : long, entrada _boolean_bots : bool) : float +mutacio(entrada chromosome : float) : float

Figura 13: classe AGGen

Les classes *EnemyInfo* i *EnemySelection* (figures 14 i 15 respectivament) estan relacionades l'una amb l'altra. La primera guarda la informació d'un enemic com ara el nom, morts, temps des de la última mort o quina prioritat té a l'hora de decidir atacar-lo en cas de trobar-se amb més enemics, d'altra banda, la classe *enemySelection* s'encarrega de seleccionar a quin de tots els enemics es disparará.

EnemySelection
-enemyList : EnemyInfo -enemySize : int = 0 -enemy : Player
+printLog(entrada _bot_num : long) : void +updateInfo(entrada _name : string, entrada _time : long) : void +sortList() : void +isInList(entrada _name : string) : int +setEnemy(entrada _enemy : Player) : void +resetEnemy() : void +getEnemy() : Player +chooseEnemy(entrada _players : Players) : Player

Figura 14: classe *EnemySelection*

EnemyInfo
-name : string -deaths : int -time : long -priority : float
+setName(entrada _name : string) : void +addDeath() : void +setTime(entrada _time : long) : void +morePriority(entrada _p : float) : void +lessPriority() : void +getName() : string +getDeaths() : int +getTime() : double +getPriority() : float

Figura 15: classe *EnemyInfo*

4.2 Diagrama de classes

La relació de totes les classes amb la classe principal (veure figura 16), *SmartBot*, és de dependència ja que la creació dels objectes de les classes *Functions*, *Movement*, *AGGen*, *Actions*, *FileManager* i *EnemySelection* estan condicionats per la instanciació provinent de l'objecte *SmartBot*. D'altra banda la classe *EnemyInfo* és una composició de la classe *EnemySelection*, això implica que la vida de l'objecte que es creí depèn del temps de vida d'*EnemySelection* i la relació entre ells es que un objecte *EnemySelection* pot tenir la informació de *n* diferents enemics.

La distribució de les classes s'ha realitzat agrupant totes les funcions que poguessin tenir relació, és a dir, dins de la classe *FileManager* només hi trobarem funcions que permetin llegir o escriure fitxers. A la *AGGen* totes les que estiguin relacionades amb els algorismes genètics, a moviment totes les que permeten que el bot es mogui, i així successivament.

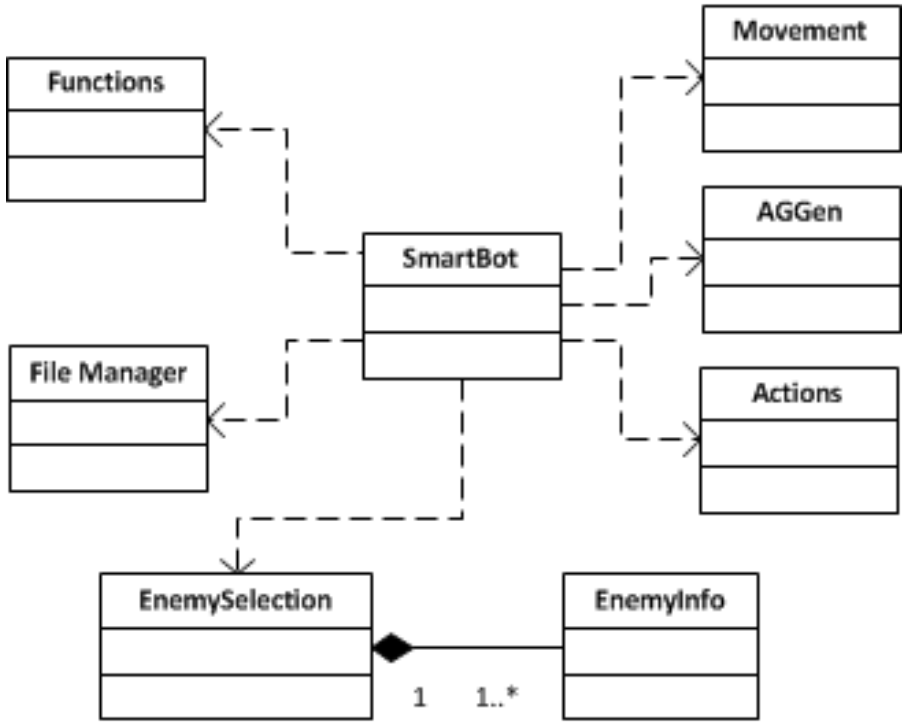


Figura 16: diagrama de classes simplificat

5. Implementació

La programació d'aquest projecte es divideix en tres parts:

- Programació de les funcions d'acció, moviment i sensors del bot.
- Inserció de la xarxa neuronal al codi del bot i configuració del sistema.
- Programació de l'estructura de carpetes i fitxers.

La programació íntegra del projecte s'ha realitzat amb java ja que és amb el llenguatge amb el qual el Pogamut adapta la informació que envia i rep de l'entorn de joc.

La programació de les funcions d'acció, control i sensors s'han programat seguint els manuals d'iniciació del Pogamut així com el javadoc del que disposa el sistema i dels seus fòrums. El Pogamut permet tractar i realitzar un gran nombre d'accions, ja sigui canviar el skin[15] del personatge o conduir vehicles segons la modalitat en la que ens trobem o disparar a objectes que volin. De totes les modalitats i comandes que ofereix el sistema, s'han descartat quasi totes les que no pertanyen al mode deathmatch ja que el projecte es centra en aquesta modalitat. Tot i així hi ha un cert nombre de comandes i mètodes que són generals per a totes les modalitats i s'han tingut en compte a l'hora de fer la tria.

Pel que fa a la inserció de la xarxa neuronal es va comprovar quina era la manera més còmoda d'executar-la des de l'entorn de desenvolupament i es va optar per que l'aplicació crees fitxers batch[16] amb la configuració de l'execució de la xarxa i els cridés sempre que fessin falta.

Finalment, després d'haver comprovat el gran nombre de fitxers que generava l'execució del programa, es va passar a la gestió de com seria l'estructura de carpetes i fitxers per tal de tenir tota la informació dels diferents bots ben controlada.

5.1 Interacció del bot amb l'entorn

Un bot pot interactuar amb l'entorn de moltes maneres diferents, ja sigui realitzant accions que el modifiquin com ara disparar fins a fer explotar bidons de gasolina, i impedir el pas als enemics o córrer per un carrer i amagar-se darrera d'una caixa.

[15] Aparença del personatge.
[16] Script per executar comandes des de la consola de windows.

A continuació veurem quin tipus d'interaccions s'han tingut en compte a l'hora de donar-li funcionalitat al bot.

5.1.1 SENSORS

Els sensors de proximitat implementats per a que el bot pugui navegar per l'escenari fan servir la tècnica anomenada raycasting. Aquesta tècnica crea uns rajos que surten de la cintura del bot amb sentit, direcció i longitud i quan detecten alguna col·lisió canvien el color a vermell, si no detecten res mantenen el seu color verd.

Cada raig que es llança des del bot segueix una línia recta fins que algun objecte, ja sigui una paret o una caixa, interromp la seva trajectòria sempre i quant estigui dins de la distància de recorregut del raig. Aquesta tècnica requereix de molt càlcul computacional ja que s'ha de realitzar una comprovació per cada raig cada vegada que es refresca la imatge, per tant s'ha d'escollir una longitud de detecció de col·lisions i un nombre de rajos que siguin els adients

Per la creació dels rajos i inicialització dels rajos s'han de definir prèviament els paràmetres que influiran en la detecció: el *rayLength*, *fastTrace*, *floorCorrection* i el *traceActor* (veure codi 1). El primer permet definir la longitud del raig i pren com a valor un nombre enter, per la seva banda, la resta de paràmetres prenen valors booleans.

El primer, *fastTrace*, indica si es vol fer servir la funció de detecció *fastTrace* o *Trace*. La funció *fastTrace* és més ràpida que la funció *trace* però aporta menys informació. L'avantatge de fer servir la funció *fastTrace* és que la informació que retorna és si es xoca amb algun objecte o no, per contra no es pot obtenir informació de la distància de la col·lisió.

El següent paràmetre, *floorCorrection*, indica si s'ha de corregir la direcció dels rajos segons la normal del bot.

Finalment, el paràmetre *traceActor* permet detectar si el bot esta xocant amb un altre personatge, si no s'activa la opció, els rajos només s'encarreguen de detectar la geometria de l'entorn.

```
final int rayLength = (int) (UnrealUtils.CHARACTER_COLLISION_RADIUS * radio);
boolean fastTrace = true;
boolean floorCorrection = false;
boolean traceActor = false;
```

Codi 1: configuració del raycast

Un cop estan definits els paràmetres de configuració dels rajos, s'han de crear, se'ls hi ha d'assignar un nom, una direcció i aplicar la configuració (veure codi 2).

```
raycasting.createRay(FRONT,   new Vector3d(1, 0, 0), rayLength, fastTrace,
      floorCorrection, traceActor);
raycasting.createRay(LEFT30,  new Vector3d(0.5, -0.866, 0), rayLength, fastTrace,
      floorCorrection, traceActor);
raycasting.createRay(LEFT60,  new Vector3d(0.866, -0.5, 0), rayLength, fastTrace,
      floorCorrection, traceActor);
raycasting.createRay(RIGHT30, new Vector3d(0.5, 0.866, 0), rayLength, fastTrace,
      floorCorrection, traceActor);
raycas
      ...
```

Codi 2: creació dels rajos amb les seves característiques

Registrem el listener que cridarem un cop els rajos hagin sigut inicialitzats (veure codi 3).

```
raycasting.getAllRaysInitialized().addListener(new FlagListener<Boolean>()
{
      @Override
      public void flagChanged(Boolean changedValue)
      {
            left = raycasting.getRay(LEFT90);
            left30 = raycasting.getRay(LEFT30);
            left60 = raycasting.getRay(LEFT60);
            right = raycasting.getRay(RIGHT90);
            right30 = raycasting.getRay(RIGHT30);
             ...
      }
});
```

Codi 3: listener per al raycast

A continuació, si no s'han d'afegir més rajos, s'informa a la instancia del gestor de raycasting que ja no se n'afegiran més (veure codi 4).

```
raycasting.endRayInitSequence();
```

Codi 4: finalització de la creació de raycast

Finalment s'han d'activar els rajos, per fer això s'ha d'indicar si es que es mostrar els rajos durant la partida, paràmetre *drawRaycasting*, i si es vol que la computació dels rajos sigui continua amb el mètode *autoTrace* (veure el codi 5 i la imatge 7).

```
getAct().act(new
       Configuration().setDrawTraceLines(drawRaycasting).setAutoTrace(true));
```

Codi 5: activació del raycast

Imatge 7: bot amb tots els rajos actius, alguns d'ells detectant col·lisió

5.1.2 ACCIONS

Les funcions d'acció que pot realitzar el bot són moltes: accionar interruptors, disparar o pujar a un vehicle però per a la modalitat de joc emprada en el projecte s'han deixat les bàsiques que són les d'atacar als enemics.

La classe que conté totes les accions del bot, anomenada *Actions*, té un constructor que rep com a paràmetres un objecte que permet al bot disparar i un altre objecte que conté la configuració de les armes del bot. Tot i que al final no s'ha descartat el codi, aquesta classe permetia configurar les armes de manera automàtica per tal de que el bot seleccionés una arma segons la distància a la que es trobava l'enemic a disparar.

Els mètodes importants de la classe són els de *shootPrimary* i *shootSecondary*, aquests mètodes permeten al bot disparar amb l'atac primari o secundari de l'arma que portin equipada. Tot i que els dos mètodes en essència fan el mateix, el primer és més complex si el comparem amb el segon.

La acció de disparar amb l'atac primari (veure codi 6) de l'arma equipada permet al bot seleccionar amb quina arma vol disparar segons les necessitats del moment, per exemple, si el bot vol disparar amb l'arma que tingui més munició de totes les que disposa, es crida a una funció indicant que vol una arma nova i la vol segons el seu ratio, d'aquesta manera la funció retorna el nom de l'arma amb més munició i se li passa a la funció de shootPrimary. A més a més d'indicar l'arma amb la que dispararà el bot, també se li passa l'enemic a qui disparar, informació sobre les armes de les que disposa el bot, dos paràmetres més que indiquen si hi ha algun projectil a l'aire i informació sobre el mateix bot.

```
public void shootPrimary(Player enemy, Weaponry _weaponry, String _weaponName,
      boolean _incoming, IncomingProjectile _projectile, AgentInfo _info)
```

Codi 6: paràmetres de la funció shootPrimary

Un cop la funció té tots els paràmetres, equipa al bot amb l'arma seleccionada (veure codi 7).

```
if(!(_weaponry.getCurrentWeapon().getType().toString()).equals(_weaponName))
{
	if(_weaponName.equals("XWeapons.ShieldGunPickup"))
		_weaponry.changeWeapon(ItemType.SHIELD_GUN);
	if(_weaponName.equals("XWeapons.AssaultRiflePickup"))
		_weaponry.changeWeapon(ItemType.ASSAULT_RIFLE);
	if(_weaponName.equals("XWeapons.BioRiflePickup"))
		_weaponry.changeWeapon(ItemType.FLAK_CANNON);
	if(_weaponName.equals("XWeapons.RocketLauncherPickup"))
		_weaponry.changeWeapon(ItemType.ROCKET_LAUNCHER);
	...
}
```

Codi 7: canviar l'arma equipada per l'arma escollida

Seguidament comprova si hi ha algú a qui disparar (veure codi 8), si no pot disparar a ningú es surt de la funció, pel contrari, si hi ha algú, es comprovarà primer que no hi hagi projectils que vagin dirigits cap al bot, si n'hi ha, el bot disparà als projectils per evitar ser danyat i després dispararà a l'enemic i si no n'hi ha disparà directament a l'enemic.

```
if(enemy != null)
{
	if(_weaponName.equals(ItemType.SHOCK_RIFLE.getName()))
	{
		if (_incoming)
		{
			shoot.shoot(_projectile.getId());
			return;
		}
		shoot.shootSecondary(_info.getLocation().add(_info.getRotation().toLoca
	tion().setZ(0.40).scale(100)));
	}
	else
	{
		if (_incoming)
			shoot.shoot(_projectile.getId());
		else
			shoot.shoot(enemy.getLocation());
```

```
                return;
        }
        shoot.shoot(enemy);
}
else
{
   shoot.stopShooting();
}
```

Codi 8: disparar a un projectil i a un enemic

Pel contrari, la funció del shootSecondary només contempla a qui es vol disparar (veure codi 9), sense tenir en compte l'arma que es vol equipar. Això s'ha fet així ja que s'ha comprovat que els jugadors a l'hora de jugar, trien les armes segons l'atac primari i no pas l'atac secundari. En el cas del bot, la funció rep com a paràmetre l'enemic a disparar.

```
public void shootSecondary(Player enemy)
{
        if(enemy != null)
                shoot.shootSecondary(enemy);
        else
                shoot.stopShooting();
}
```

Codi 9: atac secundari

5.1.3 MOVIMENT

La classe encarregada del moviment, anomenada Movement, li permet caminar, saltar, ajupir-se, etcètera. Tot tipus d'accions que estiguin relacionades amb el desplaçament del bot per l'entorn i per evitar obstacles que es pugui trobar.

Per exemple, si el bot volgués saltar (veure codi 10) es cridaria a la funció corresponent, en aquest cas saltar.

```
public void Jump()
{
        move.jump();
}
```

Codi 10: saltar

Tot i que aquesta funció és senzilla, n'hi ha d'altres que són una mica més complicades, com és el cas de la funció dodgeLeft (veure codi 11). Aquesta funció permet esquivar a un enemic de manera ràpida saltant cap a l'esquerra. Per tant, per poder realitzar això necessita la posició actual del bot i la posició de l'objectiu a esquivar.

```
public void dodgeLeft(ILocated botPosition, ILocated inFrontOfTheBot)
{
      move.dodgeLeft(botPosition, inFrontOfTheBot);
}
```

Codi 11: evasion a l'esquerra

5.2 Xarxa neuronal

La xarxa neuronal és la part més important i complexa del sistema, sense ella els bots només serien un conjunt de polígons inanimats.

Abans de poder posar la xarxa neuronal en marxa, s'han de crear i configurar els fitxers que farà servir per llegir i guardar la informació que processarà. Els fitxers que necessitarà la xarxa neural per poder treballar són els següents: File.in.#bot, ANN_#bot.net, ANN_#bot.script i File.#bot.out.

El primer, File.in.#bot conté la distribució bàsica de l'estructura de la xarxa neuronal, es a dir, nombre d'entrades i de sortides, i totes les dades d'entrada codificades en valors compresos entre 0 i 1, amb aquesta informació treballarà la xarxa a l'hora d'executar-se.

El següent fitxer, el ANN_#bot.net conté el nombre total de nodes, de connexions entre tots els nodes, el tipus d'aprenentatge que en el nostre cas és el backpropagation[17], una taula amb valors inicialitzats per defecte i, finalment, la taula de connexions.

La taula de connexions és on es guarden tots els pesos entre els nodes de la xarxa. La taula esta distribuïda de manera que el primer valor que surt, el target, és el node al qual es connectaran tots els nodes de l'apartat source amb el pes de les seves connexions. Els nodes que surten d'aquesta taula són tots els de la capa oculta, on es connecten els de la capa d'entrada, i els de la capa de sortida, on es connecten els nodes de la capa oculta.

[17] Algorisme d'aprenentatge supervisat que s'usa per entrenar xarxes neuronals artificials

El ANN_#bot.script conté informació sobre com s'ha d'executar la xarxa, quins fitxers seran els d'entrada i sortida, a on es guardaran els pesos de la xarxa, la funció d'aprenentatge, etcètera.

Finalment, el fitxer File.#bot.out guarda les sortides de la xarxa neuronal. Aquest fitxer té el mateix format que el d'entrada de dades a la xarxa neuronal, però aquest té una part addicional on es veuen els valors de sortida de la xarxa, tots compresos entre 0 i 1.

Un cop estan els fitxers creats, s'ha de crear i configurar el fitxer batch que permetrà l'execució de la xarxa, aquest fitxer s'anomena batchman_#bot. Aquest fitxer el que fa es cridar a l'executable encarregat de posar en marxa la xarxa neuronal quan li passem com a paràmetres –f, que indica quin script executarem, i el nom del fitxer corresponent al script del bot que volem executar. Un cop s'han creat tots els fitxers i estan ja preparats per executar, es passa a la part lògica del programa, allà és on es crida l'execució de la xarxa neuronal.

Dins l'execució de la part lògica, tal i com es mostra a la figura 17, es comprova si ha passat el temps mínim per executar la xarxa neuronal, aquest s'estableix a la configuració de l'execució. El temps d'execució d'accions amb el que s'ha treballat ha sigut de 500ms, això vol dir que els bots realitzen dues accions per segon i dóna la sensació de que juguen calmats. Si el temps és inferior, els bots volen realitzar moltes accions per segon i al final acaben per no fer-ne cap ja que no tenen temps per dur-les a terme i si el temps és massa gran es corre el risc de que realitzin accions que no es corresponguin amb la situació en la que es troben.

Un cop s'obtenen totes les variables d'entrada corresponents als sensors, estat del bot i equipament, es passen a la xarxa neuronal com a entrades. La xarxa s'executa en base a les entrades i retorna un conjunt de valors de sortida que es corresponen amb les accions a realitzar pel bot.

El bot recorre una a una totes les accions que pot realitzar i comprova si la variable de sortida de la xarxa neuronal associada a aquella acció li permet realitzar-la o no. Si li permet realitzar l'acció, l'executa i continua comprovant valors de sortida.

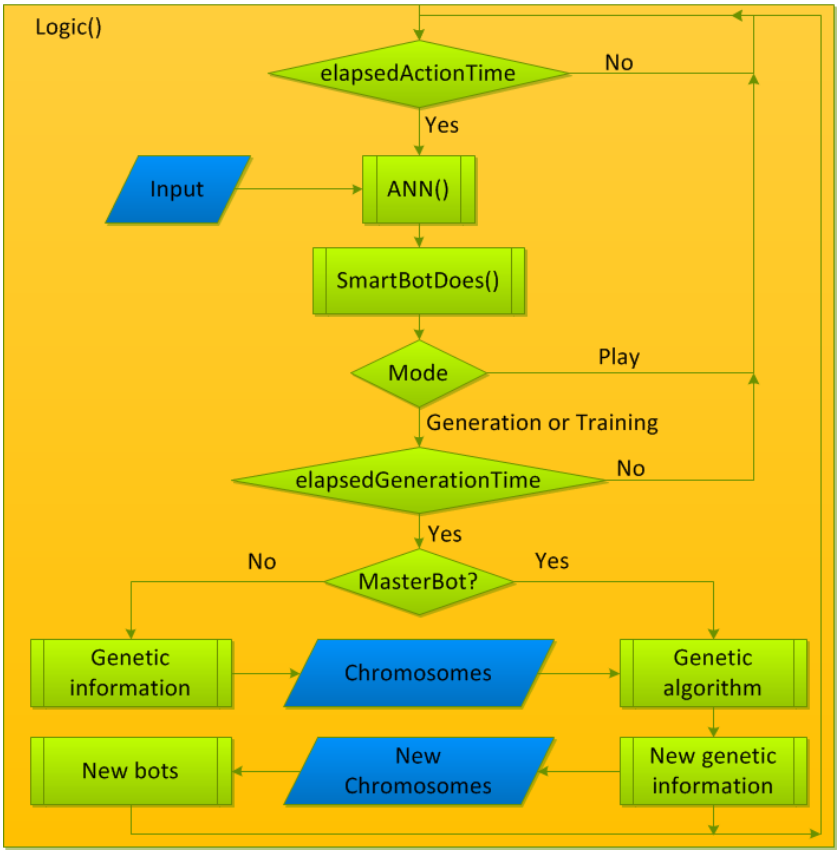


Figura 17: diagrama de flux del bloc lògic

Si el bot es troba en mode *Play*, es torna al punt de comprovar si ha passat prou temps per executar una acció, en canvi si es troba en els modes de *Generation* o *Training* es comprova si ha passat prou temps per aplicar els algorismes genètics. Si no es poden aplicar els algorismes genètics es torna al principi del bloc lògic per a comprovar si podem tornar a executar accions. En cas contrari vol dir que els podem aplicar i llavors s'ha de comprovar de quin tipus de bot es tracta per saber què ha de fer.

De bots n'hi ha de dos tipus durant l'execució: bots que s'entrenen i el masterbot. El masterbot, per la seva banda, no ataca ni es mou, es un bot que es queda quiet a l'escenari i no interactua amb ningú. La seva finalitat és la d'actuar com a àrbitre i d'encarregar-se d'aturar els bots passat un cert temps, gestionar els cromosomes de cadascun d'ells i tornar-los a posar en marxa. D'altra banda, els bots entrenats són els que es mouen per tot l'entorn mentre lluiten els uns contra els altres.

Si es tracta de bots que s'estaven entrenant, aquests guarden la seva informació genètica en fitxers així com la seva puntuació. La puntuació d'un bot es calcula segons el nombre de morts que ha fet, les vegades que l'han mort i quantes vegades s'ha suïcidat.

En el cas del masterbot, aquest s'espera a que tots els bots que s'entrenaven hagin acabat de guardar la seva informació en fitxers per aturar-los. Un cop els bots s'han aturat, s'ordenen de millor a pitjor segons la funció de *fitness* establerta. La funció de *fitness* determina quin dels bots que s'estaven entrenant és el millor. Per saber quins bots són els millors, s'aplica la següent fórmula:

Fitness = enemics morts / (morts + suïcidis)

Com més vegades un bot mati a altres bots i menys vegades es mori, millor *fitness* obtindrà i per tant millor serà, en canvi, si no mata gaire i es mor moltes vegades, ja sigui perquè es suïcida o perquè el maten, tindrà un *fitness* més baix. D'aquesta manera obtenim una llista de bots ordenada de millor a pitjor.

Seguidament el masterbot llegeix, segons l'ordre obtingut durant l'ordenació dels bots en funció del seu *fitness*, els fitxers que contenen la informació genètica d'aquests i els carrega a la memòria. A continuació s'apliquen, per ordre els algorismes genètics d'elitisme, creuament i mutació.

L'elitisme s'aplica als dos millors bots de l'actual generació. Aquests passen a la següent generació sense ser creuats amb ningú ni mutats.

El creuament i la mutació s'aplica a la resta de bots per igual. El primer, el creuament (veure figura 18), agafa un bot de tots els disponibles, l'etiqueta com a pare i busca un altre bot, incloent els dos seleccionats per elitisme, i els etiqueta com a mare. A continuació es genera un nombre aleatori que va entre 1 i la longitud del cromosoma menys 2 i és a partir d'aquest punt en el qual es modifica la informació genètica entre els dos progenitors. A continuació se'ls aplica la mutació. La mutació consisteix en generar un valor aleatori comprès entre 0 i 1, si aquest valor és inferior a 0.15, s'agafa el cromosoma i se li suma un valor aleatori generat entre -1 i 1.

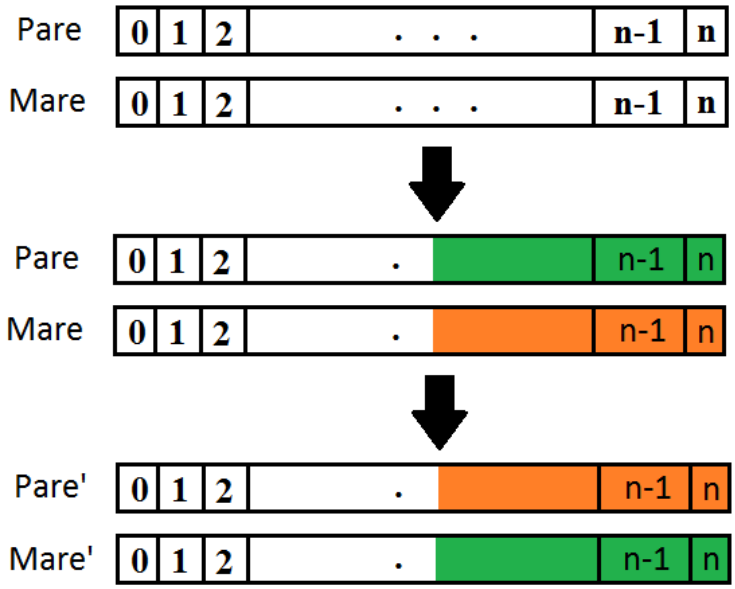


Figura 18: algorisme de creuament genètic

Finalment, quan s'han obtingut tots els nous individus, el masterbot s'encarrega d'escriure la nova informació genètica per a cada un dels bots i, posteriorment, els torna a posar en marxa per poder continuar amb l'entrenament.

5.2.1 CONFIGURACIÓ

El sistema que s'ha fet servir ha sigut el del perceptró multicapa configurat amb una sola capa oculta. Es fa servir una sola capa oculta ja que segons el teorema d'aproximació universal [18], i donat el cas de tenir suficients neurones, es pot format qualsevol assignació necessària.

A la pràctica, amb dos capes ocultes n'hi ha normalment prou per augmentar la velocitat de convergència mentre que amb més de 5 o 6 capes ocultes, ja no és necessari afegir-ne més. Una xarxa neuronal aprèn millor quan el mapeig és més senzill.

El sistema consta del següent nombre de neurones a la capa d'entrada, a la capa oculta i a la capa de sortida:

- **Capa d'entrada:** 48 nodes corresponents a tota la informació d'entrada que se li subministra a la xarxa.
- **Capa oculta:** nombre de neurones de la capa d'entrada multiplicat per dos [18] [19] [20].
- **Capa de sortida:** 46 nodes corresponents a totes les accions que pot realitzar el bot.

5.2.2 Entrades i sortides

El conjunt d'entrades de la xarxa neuronal es correspon amb tot el que el bot pot percebre de l'entorn, ja sigui si veu a un enemic o si hi ha vida al seu voltant, i de sí mateix, mentre que les sortides es corresponen amb totes les accions que el bot pot realitzar.

El sistema de gestió de la xarxa neuronal s'encarrega de posar els valors per defecte a les variables d'entrada i sortida d'aquesta. En el cas de les entrades, el reinici de les variables es realitza just al acabar de realitzar les accions seleccionades pel bot. En l'altre cas, les variables de sortida, el seu reinici es realitza després de reiniciar les variables d'entrada.

A continuació s'identifiquen i expliquen una a una tot el conjunt d'entrades a la xarxa neuronal i posteriorment es realitza el mateix amb les sortides d'aquesta (veure figura 19).

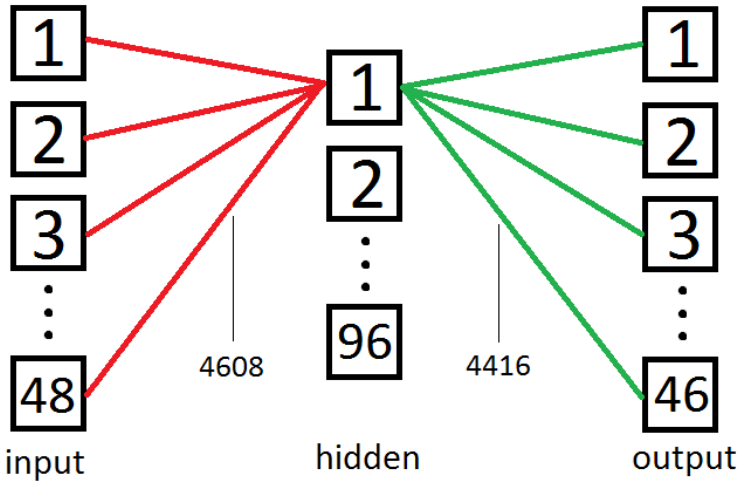


Figura 19: esquema de la xarxa

5.2.3 Entrades

En general, totes les entrades prenen com a valors 0 o 1. 0 equival a que l'entrada que li arriba no ha detectat res i 1 que si ha detectat. En els casos que els valors de les entrades comprenguin valors diferents a 0 i 1 es comenten en la mateixa entrada.

- **in_ANN_0:** in_damage. Indica si el bot esta rebent dany.
- **in_ANN_1:** in_worldDamage. Indica si el bot esta rebent dany per culpa de l'escenari, ja sigui per bidons explosius, lava o qualsevol altre tipus d'objecte.

- **in_ANN_2:** in_bulletDamage. Indica si el bot esta rebent dany per culpa de bales disparades per enemics.
- **in_ANN_3:** in_worldKiller. Indica si la mort del bot ha sigut causada per l'escenari.
- **in_ANN_4:** in_bulletKiller. Indica si la mort del bot ha sigut causada per un enemic.
- **in_ANN_5:** in_falling. Indica si el bot està a l'aire i esta caient.
- **in_ANN_6:** sensorFront. Indica si el raig de raycasting que esta apuntant cap endavant xoca amb quelcom o no.
- **in_ANN_7:** sensorLeft. Indica si el raig de raycasting que esta apuntant 90 graus a l'esquerra xoca amb quelcom o no.
- **in_ANN_8:** sensorLeft30. Indica si el raig de raycasting que esta apuntant 30 graus a l'esquerra xoca amb quelcom o no.
- **in_ANN_9:** sensorLeft60. Indica si el raig de raycasting que esta apuntant 60 graus a l'esquerra xoca amb quelcom o no.
- **in_ANN_10:** sensorRight. Indica si el raig de raycasting que esta apuntant 90 graus a la dreta xoca amb quelcom o no.
- **in_ANN_11:** sensorRight30. Indica si el raig de raycasting que esta apuntant 30 graus a la dreta xoca amb quelcom o no.
- **in_ANN_12:** sensorRight60. Indica si el raig de raycasting que esta apuntant 60 graus a la dreta xoca amb quelcom o no.
- **in_ANN_13:** sensorUpLong. Indica si el raig de raycasting que esta apuntant cap amunt xoca amb quelcom o no.
- **in_ANN_14:** sensorDownLong. Indica si el raig de raycasting que esta apuntant cap avall xoca amb quelcom o no.
- **in_ANN_15:** sensorFrontUp30. Indica si el raig de raycasting que esta apuntant endavant amb inclinació de 30 graus xoca amb quelcom o no.
- **in_ANN_16:** sensorFrontDown30. Indica si el raig de raycasting que esta apuntant endavant amb inclinació de -30 graus xoca amb quelcom o no.

- **in_ANN_17:** info.getHealth(). Retorna la quantitat de vida del bot i es divideix per 100. El valor màxim de vida és 200 però s'ha considerat que 100 és un valor adequat ja que tot el que sobrepassa aquest valor és vida extra. Obtenim així un valor comprès entre 0 i 1.
- **in_ANN_18:** info.getArmor(). Retorna la quantitat d'escut del bot i es divideix per 150 ja que és el valor màxim que es pot tenir. Obtenim així un valor comprès entre 0 i 1.
- **in_ANN_19:** energy. Retorna la quantitat de vitalitat del bot i la divideix entre 100 per obtenir així un valor comprès entre 0 i 1.
- **in_ANN_20:** players.canSeeEnemies(). Indica si el bot veu enemics o no.
- **in_ANN_21:** isItemVisible(items, "health"). Indica si el bot veu objectes de tipus vida.
- **in_ANN_22:** isItemVisible(items, "shield"). Indica si el bot veu objectes de tipus escut.
- **in_ANN_23:** isItemVisible(items, "weapon"). Indica si el bot veu objectes de tipus arma.
- **in_ANN_24:** isItemVisible(items, "ammo"). Indica si el bot veu objectes de tipus munició.
- **in_ANN_25:** isItemVisible(items, "uDamage[18]"). Indica si el bot veu objectes de tipus augment de dany.
- **in_ANN_26:** functions.weaponBag(weaponry, "XWeapons.ShieldGunPickup"). Indica si el bot disposa de l'arma Shield gun.
- **in_ANN_27:** functions.ammoBag(weaponry, "XWeapons.ShieldGunPickup"). Retorna la quantitat de munició de l'arma Shield gun compresa entre 0 i 1 on 0 equival a no tenir munició i 1 a tenir la munició complerta.
- **in_ANN_28:** functions.weaponBag(weaponry, "XWeapons.AssaultRiflePickup"). Indica si el bot disposa de l'arma Assault rifle.
- **in_ANN_29:** functions.ammoBag(weaponry, "XWeapons.AssaultRiflePickup"). Retorna la quantitat de munició de l'arma Assault rifle compresa entre 0 i 1 on 0 equival a no tenir munició i 1 a tenir la munició complerta.
- **in_ANN_30:** functions.weaponBag(weaponry, "XWeapons.BioRiflePickup"). Indica si el bot disposa de l'arma Bio rifle.

[18] Augmenta el dany amb qualsevol arma que el bot disposi durant 30 segons

- **in_ANN_31:** functions.ammoBag(weaponry, "XWeapons.BioRiflePickup"). Retorna la quantitat de munició de l'arma Bio rifle compresa entre 0 i 1 on 0 equival a no tenir munició i 1 a tenir la munició complerta.
- **in_ANN_32:** functions.weaponBag(weaponry, ItemType.SHOCK_RIFLE.getName()). Indica si el bot disposa de l'arma Shock rifle.
- **in_ANN_33:** functions.ammoBag(weaponry, ItemType.SHOCK_RIFLE.getName()). Retorna la quantitat de munició de l'arma Shock rifle compresa entre 0 i 1 on 0 equival a no tenir munició i 1 a tenir la munició complerta.
- **in_ANN_34:** functions.weaponBag(weaponry, "XWeapons.LinkGunPickup"). Indica si el bot disposa de l'arma Link gun.
- **in_ANN_35:** functions.ammoBag(weaponry, "XWeapons.LinkGunPickup"). Retorna la quantitat de munició de l'arma Link gun compresa entre 0 i 1 on 0 equival a no tenir munició i 1 a tenir la munició complerta.
- **in_ANN_36:** functions.weaponBag(weaponry, "XWeapons.MinigunPickup"). Indica si el bot disposa de l'arma Mini gun.
- **in_ANN_37:** functions.ammoBag(weaponry, "XWeapons.MinigunPickup"). Retorna la quantitat de munició de l'arma Mini gun compresa entre 0 i 1 on 0 equival a no tenir munició i 1 a tenir la munició complerta.
- **in_ANN_38:** functions.weaponBag(weaponry, "XWeapons.FlakCannonPickup"). Indica si el bot disposa de l'arma Flak cannon.
- **in_ANN_39:** functions.ammoBag(weaponry, "XWeapons.FlakCannonPickup"). Retorna la quantitat de munició de l'arma Flak cannon compresa entre 0 i 1 on 0 equival a no tenir munició i 1 a tenir la munició complerta.
- **in_ANN_40:** functions.weaponBag(weaponry, "XWeapons.RocketLauncherPickup"). Indica si el bot disposa de l'arma Rocket launcher.
- **in_ANN_41:** functions.ammoBag(weaponry, "XWeapons.RocketLauncherPickup"). Retorna la quantitat de munició de l'arma Rocket Launcher compresa entre 0 i 1 on 0 equival a no tenir munició i 1 a tenir la munició complerta.

- **in_ANN_42:** functions.weaponBag(weaponry, "XWeapons.SniperRiflePickup"). Indica si el bot disposa de l'arma Sniper rifle.
- **in_ANN_43:** functions.ammoBag(weaponry, "XWeapons.SniperRiflePickup"). Retorna la quantitat de munició de l'arma Sniper rifle compresa entre 0 i 1 on 0 equival a no tenir munició i 1 a tenir la munició complerta.
- **in_ANN_44:** functions.weaponBag(weaponry, "XWeapons.LightningGunPickup"). Indica si el bot disposa de l'arma Lightning gun.
- **in_ANN_45:** functions.ammoBag(weaponry, "XWeapons.LightningGunPickup"). Retorna la quantitat de munició de l'arma Lightning gun compresa entre 0 i 1 on 0 equival a no tenir munició i 1 a tenir la munició complerta.
- **in_ANN_46:** functions.weaponBag(weaponry, "XWeapons.RedeemerPickup"). Indica si el bot disposa de l'arma Redeemer.
- **in_ANN_47:** functions.ammoBag(weaponry, "XWeapons.RedeemerPickup"). Retorna la quantitat de munició de l'arma Redeemer compresa entre 0 i 1 on 0 equival a no tenir munició i 1 a tenir la munició complerta.

5.2.4 SORTIDES

Per la part de les sortides, aquestes prenen valors compresos entre 0 i 1. Per decidir quines sortides s'activen i quines no, s'ha aplicat un llindar específic a cada sortida segons la prioritat que vulguem donar. Per a que una sortida s'activi, el valor d'aquesta ha de ser superior al llindar.

- **out_ANN_0:** out_move = true. El bot avança cap on estigui mirant.
- **out_ANN_1:** out_rotateL = true. Aquesta sortida ja no es fa servir.
- **out_ANN_2:** out_rotateInstigator = true. Encarar-se cap a un jugador que no esta dins del seu radi de visió. El bot té un angle de visió de 180º. Si rep dany i no hi ha ningú davant seu vol dir que el qui li dispara ha d'estar, per força, al darrera. Si la sortida s'activa el bot gira 180º i llavors té a l'enemic dins del camp de visió.
- **out_ANN_3:** out_dodgeLeft = true. Salt evasiu ràpid cap a l'esquerra i mirant cap endavant.

- **out_ANN_4:** out_dodgeRight = true. Salt evasiu ràpid cap a la dreta i mirant cap endavant.
- **out_ANN_5:** out_dodge = true. Salt evasiu ràpid cap endavant i mirant cap endavant.
- **out_ANN_6:** out_dodgeBack = true. Salt evasiu ràpid cap a darrera i mirant cap endavant.
- **out_ANN_7:** out_shootPrimary = true. Disparar amb l'atac principal de l'arma equipada.
- **out_ANN_8:** out_shootSecondary = true. Disparar amb l'atac secundari de l'arma equipada.
- **out_ANN_9:** out_jump = true. Realitzar un salt.
- **out_ANN_10:** out_doubleJump = true. Realitzar un doble salt.
- **out_ANN_11:** out_crouch = true. Ajupir-se
- out_strafeL = **out_ANN_12***50. Desplaçament a l'esquerra mantenint el focus.
- out_strafeR = **out_ANN_13***50. Desplaçament a la dreta mantenint el focus.
- **out_ANN_14:** out_needMedkit = true. El bot vol agafar un kit de salut.
- **out_ANN_15:** out_ratio = true. Seleccionar arma segons la quantitat de munició que li queda. Retorna quina de les armes disponibles pel bot disposa de més munició.
- **out_ANN_16:** out_primary = true. Seleccionar arma segons la quantitat de dany de l'atac primari. Retorna quina de les armes disponibles pel bot fa més dany amb l'atac primari.
- **out_ANN_17:** out_secondary = true. Seleccionar arma segons la quantitat de dany de l'atac secundari. Retorna quina de les armes disponibles pel bot fa més dany amb l'atac secundari.
- **out_ANN_18:** out_shield = true. El bot vol agafar escut.
- **out_ANN_19:** out_uDamage = true. El bot vol agafar la millora de dany.
- **out_ANN_20:** out_R90L = true. Girar 90º a l'esquerra.
- **out_ANN_21:** out_R60L = true. Girar 60º a l'esquerra.
- **out_ANN_22:** out_R30L = true. Girar 30º a l'esquerra.

- **out_ANN_23:** out_R30R = true. Girar 30º a la dreta.
- **out_ANN_24:** out_R60R = true. Girar 60º a la dreta.
- **out_ANN_25:** out_R90R = true. Girar 90º a la dreta.
- **out_ANN_26:** out_AssaultRifle = true. El bot vol recollir l'arma Assault rifle.
- **out_ANN_27:** out_ammoAR = true. El bot vol recollir munició per l'arma Assault rifle.
- **out_ANN_28:** out_BioRifle = true. El bot vol recollir l'arma Bio rifle.
- **out_ANN_29:** out_ammoBR = true. El bot vol recollir munició per l'arma Bio rifle.
- **out_ANN_30:** out_ShockRifle = true. El bot vol recollir l'arma Shock rifle.
- **out_ANN_31:** out_ammoSR = true. El bot vol recollir munició per l'arma Shock rifle.
- **out_ANN_32:** out_LinkGun = true. El bot vol recollir l'arma Link gun.
- **out_ANN_33:** out_ammoLG = true. El bot vol recollir munició per l'arma Link gun.
- **out_ANN_34:** out_MiniGun = true. El bot vol recollir l'arma Minigun
- **out_ANN_35:** out_ammoMG = true. El bot vol recollir munició per l'arma Minigun.
- **out_ANN_36:** out_FlackCannon = true. El bot vol recollir l'arma Flack cannon.
- **out_ANN_37:** out_ammoFC = true. El bot vol recollir munició per l'arma Flack cannon.
- **out_ANN_38:** out_RocketLauncher = true. El bot vol recollir l'arma Rocket launcher.
- **out_ANN_39:** out_ammoRL = true. El bot vol recollir munició per l'arma Rocket launcher.
- **out_ANN_40:** out_SniperRifle = true. El bot vol recollir l'arma Sniper rifle.
- **out_ANN_41:** out_ammoSRifle = true. El bot vol recollir munició per l'arma Sniper rifle.
- **out_ANN_42:** out_LightningGun = true. El bot vol recollir l'arma Lightning gun.
- **out_ANN_43:** out_ammoLightG = true. El bot vol recollir munició per l'arma Linghtning gun.
- **out_ANN_44:** out_Redeemer = true. El bot vol recollir l'arma Redeemer.
- **out_ANN_45:** out_ammoR = true. El bot vol recollir munició per l'arma Redeemer.

5.3 Carpetes i fitxers

Com s'ha comentat amb anterioritat, cada bot té la seva pròpia carpeta on es guarda tota la informació relacionada amb ell i la carpeta arrel es guarden els noms, valors de *fitness* i cromosomes de cada població de bots.

El fitxer de noms, bot_players_names, té una llista ordenada dels noms dels bots segons s'han anat assignant, això es fa per no tenir bots amb un mateix nom durant l'execució de l'experiment (veure codi 12).

```
Roomba
Rambo
Ofender
Spartacus
```

Codi 12: contingut del fitxer bot_players_names

A continuació tenim el fitxer de *fitness* on es guarda la puntuació de cada bot d'una generació en concret, això permet veure l'evolució del *fitness* al llarg de l'entrenament i seguir-lo per a cada bot (veure codi 13).

```
Generation: 1
10:1.0, 20:1.0, 22:0.0, 24:0.0
Generation: 2
10:0.5, 20:1.0, 22:0.66, 24:0.0
Generation: 3
10:1.0, 20:0.66, 22:0.5, 24:1.0
```

Codi 13: *fitness* de cada bot de l'execució durant 3 generacions

Finalment hi ha dos fitxers més, un anomenat currentGeneration que es fa servir a mode de temporal per saber quina és la generació actual, de manera que si s'ha de pausar l'execució per fer un canvi de mapa i s'ha de continuar des d'allà on estava es pot fer sense problema. L'altre fitxer és el que s'anomena:

Generation #generació----- data.txt

El nom va canviant segons la generació que es guarda abans d'aplicar els algorismes genètics i s'hi afegeix la data, d'aquesta manera es genera un fitxer nou per cada vegada nova població que s'obté.

El fitxer segueix un format molt concret per tal de tenir el màxim d'informació possible, així doncs hi ha una capçalera amb paràmetres que indiquen quants bots hi ha guardats al fitxer, el nombre de nodes a cada una de les capes i el total de connexions. Tota aquesta informació, a més a més de ser útil per a l'usuari al observar el fitxer, és útil a l'hora de llegir la informació genètica de cada bot ja que sabent el nombre de nodes de cada capa es pot saber fins a quin valor es correspon amb una connexió d'entrada a la capa oculta i quin es correspon amb la connexió a la capa de sortida.

Finalment hi ha una capçalera que indica a quin bot pertanyen tots els valors separats per comes que hi ha a continuació (veure codi 14).

```
    Bots    |   Input   |    Hidden   |   Output   |   Chromosome
------------|-----------|-------------|------------|----------------
      4     |    48     |      96     |     46     |      9024
------------|-----------|-------------|------------|----------------
-----------------------------------------------
-----------------------10----------------------
-----------------------------------------------
0.11898649, 0.304767, -0.6110643, -0.46178758, -0.6801237, -0.1945138, 0.043199...
```

Codi 14: contingut del fitxer Generation 1-----13-06-2013 22-57-28

Com s'ha comentat anteriorment, cada bot disposa d'una carpeta pròpia on guarda tots els fitxers d'execució de la xarxa neuronal i a la seva informació, aquesta es troba repartida en fitxers segons el que aquests guardin. Exemples en són els fitxers per guardar enemics, un altre per estadístiques i un últim per guardar les dades sense processar de la puntuació del bot.

El fitxer on es guarden els enemics, anomenat enemyLog, pretén ajudar al bot a l'hora de seleccionar a quin enemic disparar (veure codi 15). Cada vegada que el bot es mort per un enemic, aquest s'afegeix a la llista d'enemics, o, si ja hi existeix, s'actualitza la seva informació. La informació conté el nom de l'enemic, quantes vegades ha mort al bot, el temps des de la última vegada que va ser mort per aquest i la prioritat. Aquests dos últims valors són els que es tenen en compte a l'hora de seleccionar l'enemic a disparar en cas de trobar-se amb més d'un en el camp de visió. La prioritat augmenta en intervals de 0.5 cada vegada que un enemic mata al bot, així doncs, a l'hora de seleccionar a qui disparar es mira

quin dels enemics de la llista té la prioritat més alta i se li dispara. En cas de que el de prioritat més alta no estigui dins del camp de visió es van comprovant tots els enemics de la llista un a un i, si s'arriba al final i no n'hi ha cap d'ells, es selecciona aleatòriament un dels enemics visibles. En cas d'haver-hi empat i si uns quants enemics tenen la mateixa prioritat, es selecciona el que faci menys temps que hagi disparat al bot.

```
      #      |      Player |      Deaths |      Priority      |      Time
-------------------------------------------------------------------------------
      1      |      Bender |      4      |      2.0    |      25378
      2      |      Goku   |      2      |      1.0    |      98361
      3      |      HAL    |      1      |      0.5    |      63451
      4      |      void   |      0      |      0.0    |      0.0
      5      |      void   |      0      |      0.0    |      0.0
```

Codi 15: contingut del enemyLog

Per aplicar aquesta conducta al bot a l'hora de seleccionar l'enemic a qui disparar es va fer una petita enquesta a un conjunt de 20 persones que podien estar acostumades, o no, a jugar a videojocs i se'ls va preguntar que farien en cas de trobar-se envoltats per un conjunt d'enemics on, un d'ells, fos algú que no parava d'atacar-lo. Les possibles respostes eren:

- a) Disparar al primer enemic que es pugui.
- b) Intentar disparar a l'enemic concret.
- c) Amagar-se i esperar a veure que passa.
- d) No jugo a videojocs.

Tal i com es mostra a la figura 20, la majoria dels enquestats prefereix intentar disparar a l'enemic que més els hagi matat, seguit per la opció de disparar al primer enemic que es posi a tret i finalment, un empat entre les opcions d'amagar-se i no jugo a videojocs.

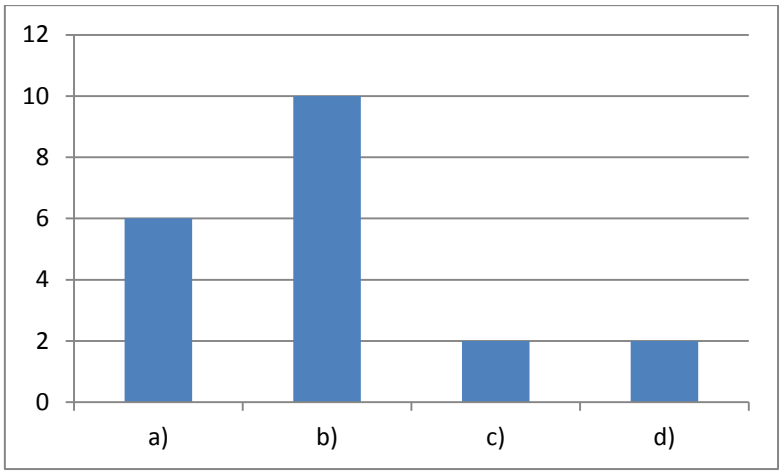


Figura 20: resultats de l'enquesta

L'últim fitxer que guarda informació del bot és el score_#bot. Aquest fitxer guarda, per a cada generació: la puntuació del bot, les morts realitzades, quantes vegades ha estat mort per un enemic i quantes vegades s'ha suïcidat, tant per caure al buit com per fer-se mal a l'hora de disparar un projectil. S'ha optat per guardar aquesta informació així, per tal de poder seguir i comprovar l'evolució del bot de manera desglossada(veure codi 16).

```
13-06-2013 22-57-08
Generation: 1
Score: 0     Kills: 0     Deaths: 1    Suicides: 0
----------
13-06-2013 22-58-08
Generation: 2
Score: 1     Kills: 1     Deaths: 1    Suicides: 0
----------
13-06-2013 22-59-51
Generation: 3
Score: 1     Kills: 1     Deaths: 0    Suicides: 1
```

Codi 16: contigut del fitxer score_10

6. Experimentació i resultats

En un projecte com aquest és important realitzar moltes proves ja que es treballa amb una tecnologia poc aplicada en el camp dels videojocs i no hi ha gaire experiència en treballs anteriors.

La metodologia seguida durant la major part del projecte ha estat la de prova i error per comprovar que no hi havia errors durant la generació de noves poblacions, ja que si al crear noves poblacions es fa amb errors. La investigació no serveix. Un cop es va comprovar que les noves poblacions generades ho feien sense errors, es va passar a la fase d'entrenar-les un temps indefinit fins que s'assolissin els objectius.

6.1 Experimentació del fitness

Com s'ha vist anteriorment, el *fitness* es fa servir per avaluar com de bo és un bot però només té en compte l'habilitat a l'hora de matar enemics i evitar ser matat i no contempla la navegació.

Per a la navegació es va implementar un sistema de penalització cada vegada que el bot xocava, de manera que al final de la generació el bot que tenia un valor de *fitness* de navegació més petit era el bot que millor navegada. Això però, presenta sèrie de de problemes:

- Terreny totalment irregular i difícil de reconèixer.
- El bot pot decidir no moure's.
- El bot pot decidir caminar en cercles.

El primer problema es presenta al comprovar que tots els escenaris són diferents i les zones de navegació del bot varien d'un a l'altre. El bot es pot desplaçar per les 3 dimensions del mapa, això implica que ha de xocar moltes vegades durant el desplaçament, ja sigui amb les parets o amb el sostre. En canvi si el mapa fos en 2 dimensions el problema es reduiria a només les parets, tot i així seguiria havent-hi el problema de la navegació.

Durant la fase d'entrenament es va comprovar que tots els bots que es movien pel mapa obtenien valors de *fitness* molt elevats ja que xocaven constantment amb les parets o, si el tenien baix, era perquè decidien quedar-se quiets al lloc i així no ser penalitzats al no xocar

amb res o bé, si la superfície era prou àmplia, donaven voltes en cercles. Vist això es va optar per buscar altres mètodes per millorar la navegació.

Un mètode que semblaria donar bons resultats és el de col·locar els bots a un costat d'un laberint[21] [22] i esperar a que aconseguissin sortir per l'altra banda. El bot que es trobés més proper a la sortida seria el que millor *fitness* obtindria. Un cop es van posar els bots dins d'un laberint preparat per un usuari, se'l va deixar entrenant una estona i es va comprovar que els bots no es movien degut a que el mapa, al ser estret, feia que tots els rajos de detecció de col·lisió estiguessin constantment avisant de col·lisió. Per evitar això es va reduir la longitud dels rajos i es va prosseguir amb l'entrenament. Els resultats que es van obtenir de l'entrenament eren bots que, o no es movien o, si ho feien, es desplaçaven uns metres i després tornaven pel camí recorregut.

Finalment es va descartar intentar trobar un mètode d'entrenament òptim per tal de fer que el bot navegués correctament pel mapa i es va optar per deixar que navegués lliurement afegint la recol·lecció d'objectes.

La recol·lecció d'objectes funciona de manera que, si un objecte es troba dins de l'espai visual del bot, aquest es troba dins del radi d'acció definit i el bot te necessitat d'aquest objecte. S'ignoren totes les sortides de la xarxa neuronal referents al moviment del bot i s'activa una funció de navegació automàtica per punts de control que fan que el bot arribi a l'objecte desitjat (veure imatge 8). Un cop el bot ha aconseguit l'objecte que volia, es tornen a tenir en compte les opcions de navegació de la xarxa neuronal.

El resultat final de la incorporació de la funció de recol·lecció d'objectes amb la de navegació lliure fan que el bot sembli que es desplaça pel mapa correctament ja que la xarxa neuronal tendeix a fer que el bot proporcionalment camini més en línia recta en comptes de donar voltes sobre sí mateix i si ha de recollir un objecte ho farà correctament ja que, mentre hi va, anirà per llocs on és segur que no xocarà amb cap element.

Imatge 8: camins entre nodes de navegació

Així doncs, un cop ens assegurem que el bot realitza correctament les accions assignades, es llança una nova execució per generar diverses poblacions amb deu individus a cada una. Un cop s'obtenen 100 generacions, es canvia el mode dels bots pel de jugar i s'observa la partida detingudament. La selecció del bot per presentar a la nostra competició ha de complir que, a més de disparar als enemics correctament, realitzi una navegació i uns moviments d'acord amb l'entorn que l'envolta. És aleshores quan s'observa durant una estona cada un dels bots i es van descartant segons la gravetat dels errors comesos, fins que al final se'n selecciona un que reuneix les condicions necessàries per participar a l'experiment.

6.2 Millores

Per comparar les millores obtingudes en conjunt per la xarxa neuronal i els algorismes genètics s'han llençat dos execucions amb 10 bots cada una; la primera ha entrenat bots al llarg de 100 generacions, fent pauses cada 6 minuts per tal de renovar les poblacions i escriure en fitxers les puntuacions obtingudes al llarg de la generació de baixes, vegades que han mort i suïcidis. La segona generació no ha sigut entrenada, s'ha obtingut una generació inicial i cada 6 minuts s'han anat guardant les puntuacions com si d'una nova generació es tractés.

Un cop s'han obtingut tots els fitxers de cada execució, s'ha calculat la mitja de baixes causades, morts i suïcidis obtinguts per les diferents generacions amb els següents resultats:

Com es pot veure a la figura 21, es mostra el valor mig de baixes causades per un bot al llarg de blocs de 10 generacions. Els que s'han entrenat han notat ràpidament una millora notable en la seva capacitat de matar, mentre que els que no s'han entrenat no han millorat i mostren una tendència a estancar-se al voltant dels 3 baixes. Un dels motius pels quals es dona aquest estancament és pel fet que, inicialment, molts dels bots no es mouen, per tant si no es volen moure no veuen enemics i per tant no causen baixes.

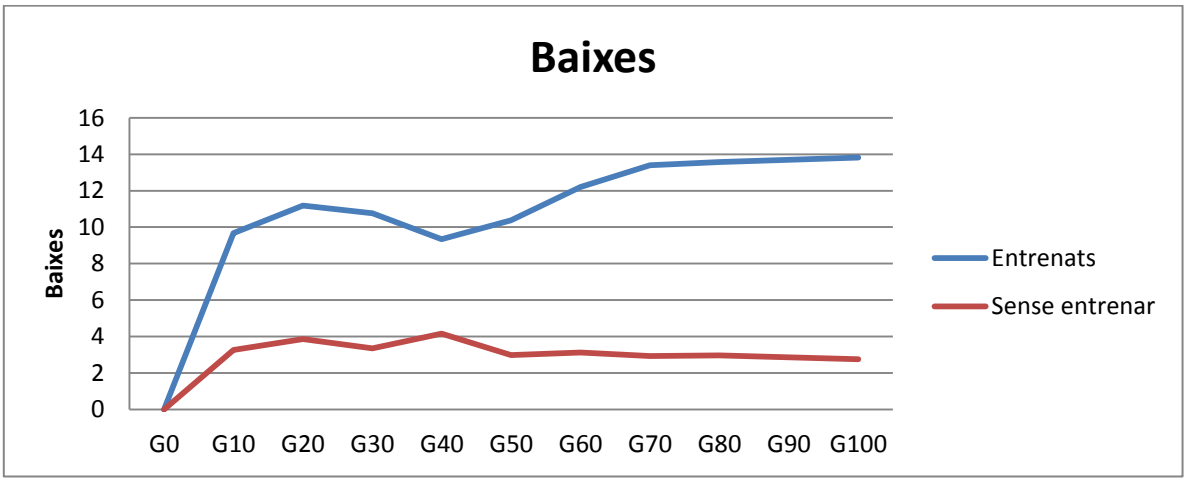


Figura 21: relació de baixes

En aquesta altra figura, la 22, es mostra la millora centrada en el nombre de baixes que ocasiona un bot entrenat, de mitja, en un minut. Mentre que un bot sense entrenament difícilment arriba a matar a un enemic cada dos minuts, un bot entrenat és capaç de matar dos enemics per minut.

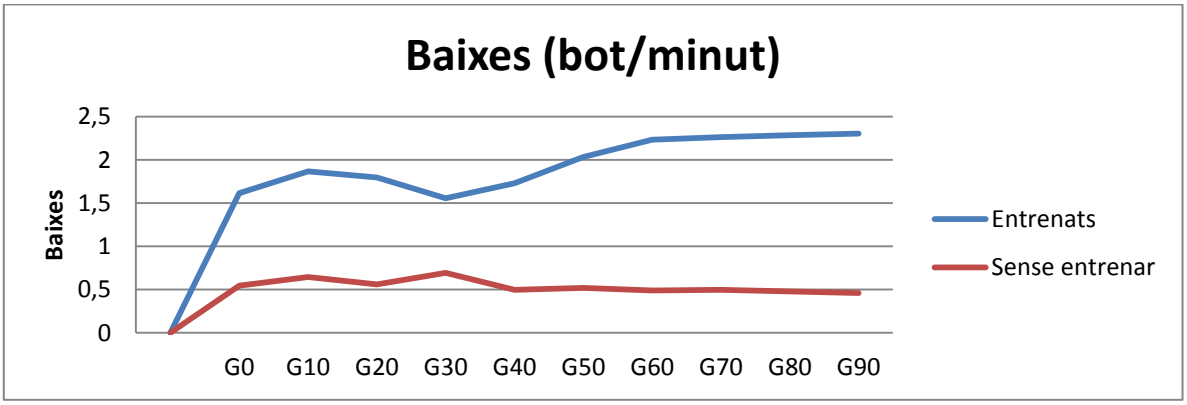


Figura 22: relació de baixes d'un bot/minut

Pel que fa al tema de les vegades que ha mort el bot, com s'aprecia a la figura 23, és normal que, sense entrenament, el nombre de morts dels bots no augmenti gaire ja que no es

mouen i per tant no interactuen entre ells, tot i això, quan comencen a entrenar i a aprendre a matar, el nombre de morts augmenta ja que va en consonància amb el nombre de baixes, com més baixes causen els bots, més morts es produeixen.

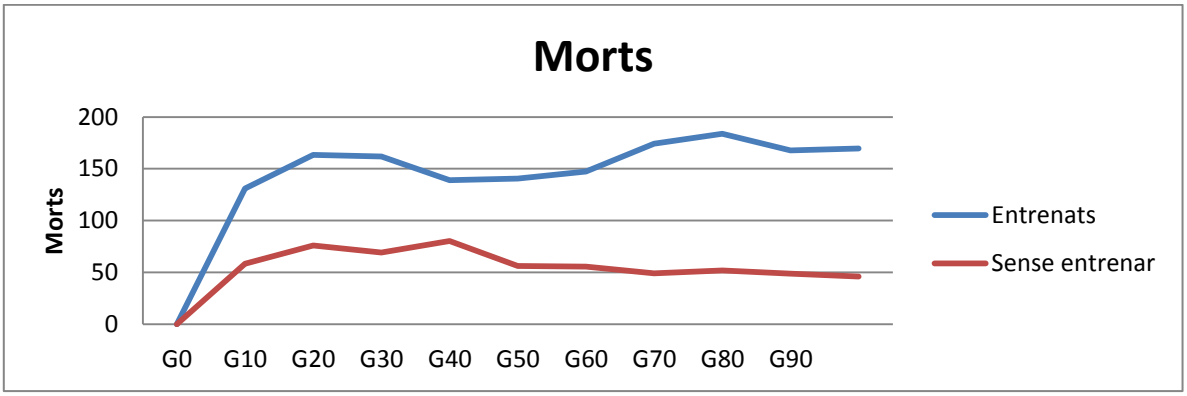


Figura 23: relació de morts

El nombre de suïcidis que mostra la figura 24 també segueix la mecànica de les gràfiques anteriors si pensem que, el fet que un bot no es mogui li proporciona seguretat a l'hora de no suïcidar-se ja que no interactua amb l'escenari i per tant no pot caure per desnivells abruptes. Quan comencen a aprendre és quan augmenta molt el nombre de suïcidis ja que la xarxa neuronal ha d'aprendre que aquesta acció no és favorable, és per això que a partir de la generació vint cada vegada es suïciden menys encara que es moguin per l'escenari.

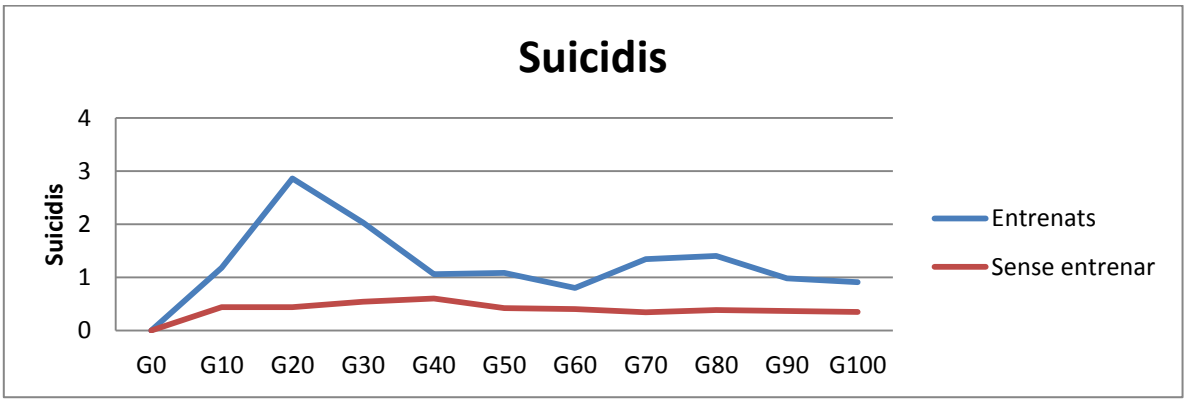


Figura 24: relació de suicidis d'un bot/minut

6.3 Resultats

Per obtenir els resultats sobre el grau d'humanitat del bot s'han realitzat 4 competicions amb altres bots i amb persones actuant com a jutges. Els bots que han participat han sigut el Conscious-Robots, creat per en Raul Arrabales, el CCBotSOAR, creat per en Raúl Arrabales i en Jorge Muñoz, i finalment, l'ADANN, que és el que correspon a aquest projecte. Mentre que els bots s'han mantingut al llarg de totes les competicions amb alguna millora aplicada, les persones que jugaven han anat canviant, això s'ha fet per tal de que la gent quan anava a votar no tingués cap experiència prèvia i així poder ser més objectiva.

A la primera competició es va demanar la participació d'un grup d'alumnes per a l'experimentació. Per a realitzar la votació, els jugadors s'havien d'equipar amb una arma concreta, la Link gun, i disparar amb el botó primari o secundari segons si creien que era un bot o un humà respectivament.

Els resultats que es van obtenir, respecte al nostre bot, es corresponien amb un grau d'humanitat del 17.3% mentre que els altres bots, el CCBotSoar i el Conscious-Robots, van obtenir un 21.2% i un 19.6% respectivament (veure taula 3). Durant l'experiment es va comprovar que l'ADANN a vegades disparava a objectius que no estaven presents en aquell moment i, per tant, semblava que disparés a l'aire o cap a les parets. Els participants comentaven que havia de ser un bot per força ja que ningú dispara a l'aire o a les parets, així que per tal d'arreglar el problema es va prescindir de la funció de seleccionar enemic segons la llista de prioritats i es va passar a seleccionar-ne un segons si es trobava dins del camp de visió del bot, independentment de si l'enemic a qui disparava el bot l'havia matat o no.

A la segona competició es va entrenar un bot nou ja que l'anterior no ens servia i es va demanar la participació d'un altre grup d'alumnes. Aquest cop el grau d'humanitat es va casi duplicar, arribant fins al 37% (veure taula 3). Tot i ser un molt bon resultat, van sorgir algunes idees que crèiem que podien ajudar a millorar el bot. Una d'elles era canviar el nombre d'execucions que realitzava un bot cada segon. Durant totes les execucions el nombre d'accions per segon s'han mantingut en 3, es a dir, cada segon es comprovaven tots els sensors i variables, la xarxa neuronal les processava i triava quines accions eren les més adequades, això comportava que a vegades el bot estigues indecís a l'hora de navegar, ja que la primera acció li podia indicar d'avançar, mentre que la segona el feia aturar i la

tercera el feia girar. Tot i que tres accions per segon són el nombre d'accions que podria arribar a realitzar un jugador experimentat, es va decidir intentar a veure quins resultats s'obtenia amb només dues.

Finalment es va augmentar la distància de visió d'objectes. Anteriorment els objectes havien d'estar a una distància de 250 unitats, distància equivalent a la longitud dels rajos de raycasting, però es va augmentar fins a arribar a ser 3 vegades la longitud d'aquests, 750 unitats.

Després d'aplicar aquestes modificacions i d'entrenar un bot nou, es va formar un tercer grup de participants per a l'experiment, en aquest cas es tractava de persones que treballaven al Centre de Visió per Computador. En aquest cas el grau d'humanitat obtingut ha estat una mica inferior, d'un 5.5% menys (veure taula 3). Tot i ser un resultat inferior, el conjunt d'accions realitzades pel bot donaven més una sensació d'harmonia i control que no pas en altres experiments. Una acció que es va comentar i que va penalitzar el bot va ser el fet de que quan se'l disparava, en comptes de girar-se i veure quina era la situació, continuava fent qualsevol altra cosa. Davant d'aquesta situació es podia implementar algun sistema per tal de que el bot, si no veia ningú al davant i a més rebia dany, fes una rotació de 180º, però implementant aquest sistema es manipulava d'alguna manera les decisions del bot i, com que el bot pot aprendre per sí mateix a realitzar aquest conjunt d'accions, ja que totes elles formen part de d'entrades i sortides, no es va fer i així s'aconseguia un dels objectius d'aquest projecte que es el de fer una IA en que intervingui el factor humà i que no fos tant perfecta.

Finalment, per a la última competició es va fer servir el mateix bot que per a la tercera competició per veure si els resultats eren semblants o variaven molt. Després de fer participar a altres professors del CVC es va obtenir una puntuació pràcticament igual a la competició anterior, amb una diferencia de 1.5% menys respecte al grau d'humanitat anterior (veure taula 3).

Bot	Competició 1	Competició 2	Competició 3	Competició 4	Mitja
ADANN	17.3410 %	37.0370 %	31.7308 %	30.2632 %	29.093 %
CCBotSOAR	21.1765 %	27.0588 %	35.8025 %	38.8158 %	30.7134 %
Conscious-Robots	19.5876 %	19.8413 %	16.8142 %	20.1389 %	19.0955 %

Taula 3: graus d'humanitat obtinguts durant les competicions

Si realitzem les mitges entre els diferents bots, podem observar com el Conscious-Robots, bot que va guanyar la competició del Botprize del 2012 amb un 32% d'humanitat ha quedat en tercera posició amb un nou grau d'humanitat del 19%, mentre que els altres dos bots, el CCBotSOAR i l'ADANN han obtingut un grau d'humanitat sensiblement del 30%.

7. Conclusions

Amb els resultats obtinguts fins ara, es pot afirmar que el treball realitzat ha complert de manera satisfactòria els objectius que s'havien fixat:

- Estudi i comprensió de les xarxes neuronals artificials.
- Aplicar tècniques d'intel·ligència artificial.
- Aprenentatge amb el Pogamut.
- Dur a terme un sistema d'intel·ligència artificial.

Tot i ja tenir una base de coneixements en xarxes neuronals artificials, s'ha pogut aprofundir una mica més a l'hora de posar en pràctica la unió d'aquestes xarxes amb la idea de crear una intel·ligència capaç de jugar per si sola a un videojoc. La xarxa havia de ser prou lleugera per executar-se entre dues i tres vegades cada segon sense que l'usuari notés que el bot seguia realitzant una acció que no tenia res a veure amb el context en el que es trobava, això ha obligat a investigar quina és la estructura i la distribució de les millors connexions per tal d'obtenir uns resultats satisfactoris.

D'altra banda també s'ha hagut d'aplicar diversos algorismes de decisió per tal d'ajudar a la xarxa a seleccionar l'enemic ja que ella per sí sola no ho pot fer. Per a que la xarxa pogués seleccionar un enemic, s'hauria de crear una entrada per a cada un de tots els possibles enemics que el bot pugui trobar-se durant la partida i això augmentaria la complexitat de la xarxa neuronal així com el càlcul del seu resultat.

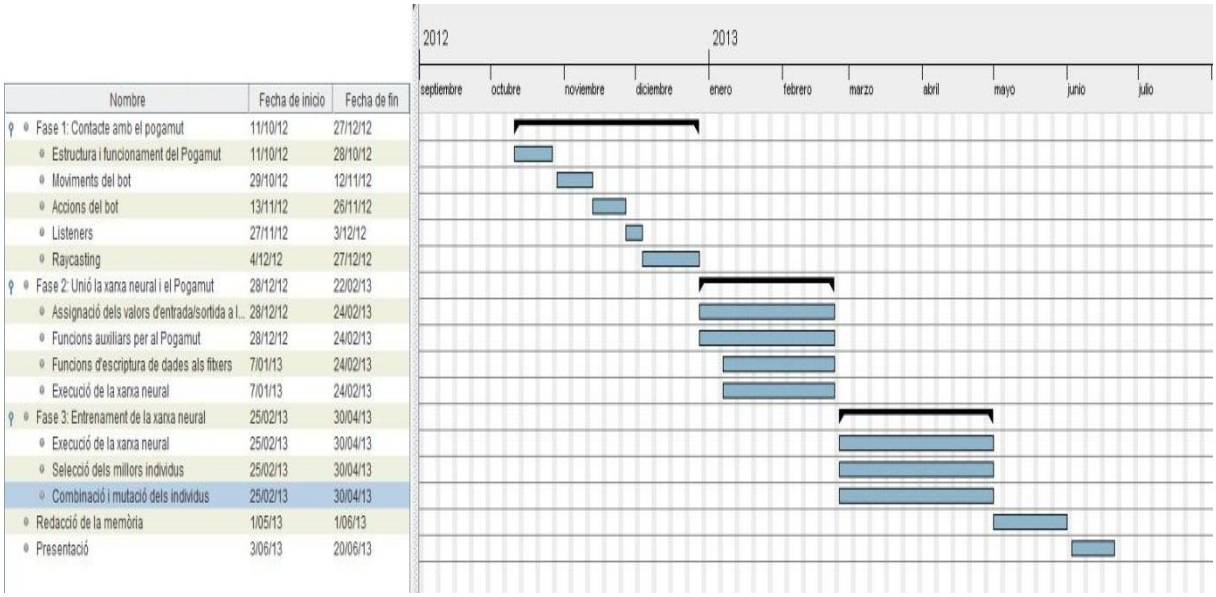
Pel que fa al Pogamut, s'ha trobat una plataforma còmode i molt ben documentada que ha permès realitzar tot tipus d'accions i obtenir tot tipus d'informació, tant del bot com de la partida en sí. A més a més, al ser la plataforma amb la qual tots els participants de la competició del Botprize han hagut d'utilitzar, li dona un bon recolzament per part de la comunitat i en garanteix la seva continuïtat. Recentment han llençat una nova versió, la 3.5.1 que, entre d'altres millores, permet la navegació del bot mitjançant la malla i s'ha habilitat la compatibilitat amb l'Unreal 3.

Finalment es pot dir que s'ha aconseguit realitzar un sistema complex d'intel·ligència artificial capaç d'obtenir en el test de Turing, un grau d'humanitat del 29%, i superant, en un

cara a cara, el bot Conscious-Robots que va participar i guanyar a la competició del Botprize de 2010.

7.1 Planificació

Inicialment el projecte consistia en tres fases, amb la previsió de finalitzar la memòria al mes de maig i fer la presentació del projecte duran el mes de juny, com mostra la figura 25:



Nombre	Fecha de inicio	Fecha de fin
Fase 1: Contacte amb el pogamut	11/10/12	27/12/12
Estructura i funcionament del Pogamut	11/10/12	28/10/12
Moviments del bot	29/10/12	12/11/12
Accions del bot	13/11/12	26/11/12
Listeners	27/11/12	3/12/12
Raycasting	4/12/12	27/12/12
Fase 2: Unió la xarxa neural i el Pogamut	28/12/12	22/02/13
Assignació dels valors d'entrada/sortida a l...	28/12/12	24/02/13
Funcions auxiliars per al Pogamut	28/12/12	24/02/13
Funcions d'escriptura de dades als fitxers	7/01/13	24/02/13
Execució de la xarxa neural	7/01/13	24/02/13
Fase 3: Entrenament de la xarxa neural	25/02/13	30/04/13
Execució de la xarxa neural	25/02/13	30/04/13
Selecció dels millors individus	25/02/13	30/04/13
Combinació i mutació dels individus	25/02/13	30/04/13
Redacció de la memòria	1/05/13	1/06/13
Presentació	3/06/13	20/06/13

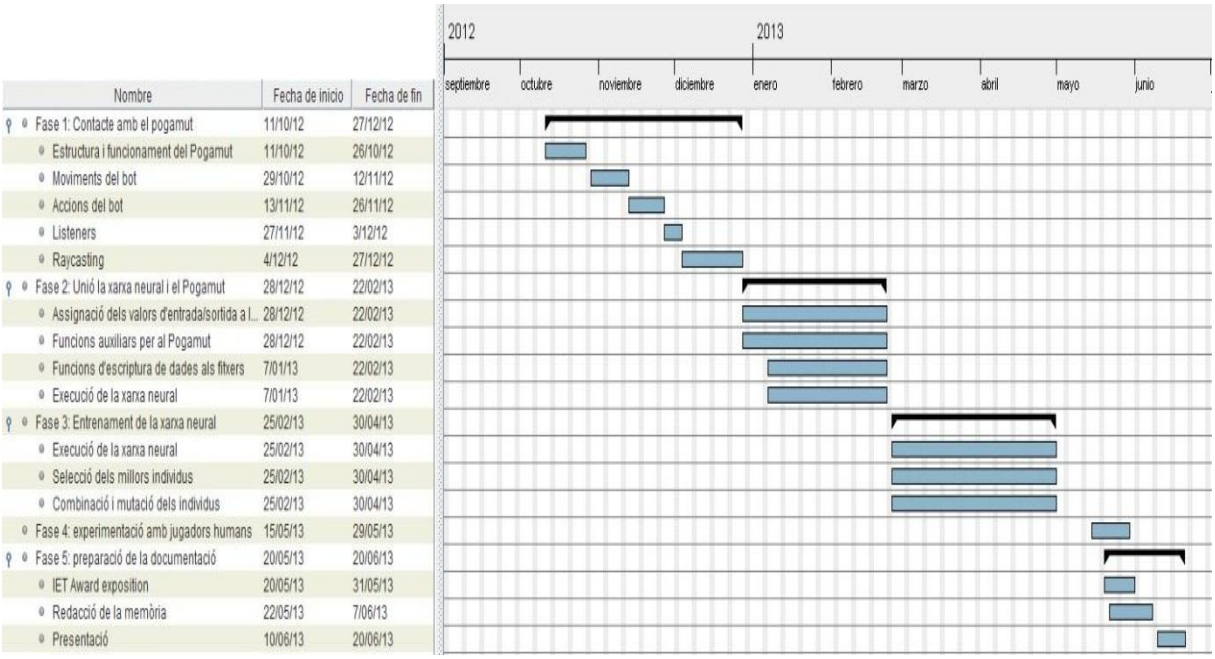
Figura 25: diagrama de Gantt amb la planificació inicial del projecte

En el curs del desenvolupament del treball s'ha vist la possibilitat d'ampliar el projecte amb una quarta i cinquena fase, modificant-se així la darrera part (veure figura 26).

La quarta fase correspon a la realització de partides entre els bots entrenats amb bots d'altres universitats i persones humanes i la cinquena fase és la de preparació i presentació del projecte al concurs proposat per la IET[19]. Finalment el projecte queda dividit en cinc fases:

[19] Institution of Engineering and Technology

- **Fase 1:** aprenentatge de l'estructura i el funcionament del Pogamut així com implementació de mètodes d'obtenció d'informació del bot i quines accions pot realitzar.
- **Fase 2:** inserció de la xarxa neuronal al codi realitzat, implementació mètodes auxiliars i escriptura de dades en fitxers i comprovació de que funciona correctament.
- **Fase 3:** entrenament de bots i ajust de paràmetres i funcions.
- **Fase 4:** experimentació amb altres bots i amb jugadors humans.
- **Fase 5:** redacció de la memòria, presentació del projecte i presentació per al concurs d'IET.



Nombre	Fecha de inicio	Fecha de fin
Fase 1: Contacte amb el pogamut	11/10/12	27/12/12
Estructura i funcionament del Pogamut	11/10/12	26/10/12
Moviments del bot	29/10/12	12/11/12
Accions del bot	13/11/12	26/11/12
Listeners	27/11/12	3/12/12
Raycasting	4/12/12	27/12/12
Fase 2: Unió la xarxa neural i el Pogamut	28/12/12	22/02/13
Assignació dels valors d'entrada/sortida a l...	28/12/12	22/02/13
Funcions auxiliars per al Pogamut	28/12/12	22/02/13
Funcions d'escriptura de dades als fitxers	7/01/13	22/02/13
Execució de la xarxa neural	7/01/13	22/02/13
Fase 3: Entrenament de la xarxa neural	25/02/13	30/04/13
Execució de la xarxa neural	25/02/13	30/04/13
Selecció dels millors individus	25/02/13	30/04/13
Combinació i mutació dels individus	25/02/13	30/04/13
Fase 4: experimentació amb jugadors humans	15/05/13	29/05/13
Fase 5: preparació de la documentació	20/05/13	20/06/13
IET Award exposition	20/05/13	31/05/13
Redacció de la memòria	22/05/13	7/06/13
Presentació	10/06/13	20/06/13

Figura 26: diagrama de Gantt amb la planificació final del projecte

8. Ampliacions i treballs futurs

Tots els projectes han de finalitzar en un termini de temps determinat, i no es pot anar millorant indefinidament, és per això que hi ha fites que no s'han pogut incloure en aquesta versió però que poden servir com a referència de millores per al futur, com són:

- Nous modes de joc.
- Listeners.
- Aprenentatge no supervisat.

La implementació de noves funcions com ara agafar una bandera o utilitzar un vehicle permetria al bot participar en altres tipus de modes i fer-lo més semblant al comportament d'una persona ja que sinó, en la situació actual, el bot podria participar en una partida deathmatch per equips i disparar als seus propis companys o bé no saber que fer en cas de trobar-se amb un vehicle.

Una característica que dóna molt poder al Pogamut és la implementació de listeners per poder obtenir de manera síncrona i asíncrona informació del joc independentment del punt d'execució d'on ens trobem. En el present projecte, durant la fase de contacte amb el Pogamut, es van implementar de manera senzilla diversos tipus de listeners com ara un detector d'explosions, que indica a quines coordenades s'ha produït una explosió i per quin objecte ha sigut produïda, si hi ha soroll de passes i a quina posició es troben o fins i tot per saber si el bot esta caient per un precipici. Durant les proves es van detectar errors a l'hora de localitzar la posició on s'havia realitzat una explosió ja que, es produís on es produís, el resultat obtingut sempre era a les coordenades [0,0,0]. Juntament amb altres errors detectats durant les proves amb listeners, es va informar a l'equip de desenvolupament del Pogamut i uns mesos més tard van treure una versió nova on, entre d'altres millores, s'arreglaven aquests errors.

Finalment la implementació d'aprenentatge no supervisat permetria que el bot aprengués sobre la marxa a mida que va realitzant accions en comptes d'aplicar algorismes genètics i notar els canvis aplicats cada cert temps. Això beneficiaria, entre altres coses, la implementació d'un millor sistema de navegació.

9. Annex

9.1 Funcionament del sistema

Un cop s'hagi obert el projecte s'haurà de posar en marxa el servidor que serà configurat per a un mapa, un temps i unes morts concretes. El mode de joc ha de ser sempre el deathmatch ja que qualsevol dels altres modes dels que disposa l'Unreal no garanteix el seu bon funcionament ja que no han sigut testejats. L'usuari només haurà d'intervenir cada vegada que es canviï el mapa del servidor ja que l'execució del programa s'aturarà en aquell moment, ja que el servidor expulsa a tots els bots.

Durant l'execució es podrà anar seguint l'execució del programa ja que hi apareix la informació més immediata.

A l'hora de posar en marxa el sistema s'hauran de seguir els següents passos i en aquest ordre:

- Posar en marxa el servidor configurat degudament.
- Carregar el projecte i configurar la seva execució segons si es desitja entrenar o jugar contra els bots.
- Executar el programa
- (Opcional): llançar una execució del joc i entrar a veure/participar a la partida.

9.2 Manual

El projecte consta de moltes variables de control que defineixen els paràmetres de la xarxa neuronal: threads, nom dels bots, radi de col·lisió, etcètera. Aquí només en comentarem els que realment serveixen per canviar la configuració de l'execució del programa. La configuració de l'execució té els següents paràmetres:

- **Arrel:** conté la carpeta arrel on es guardaran tots els fitxers generats per l'execució.
- **bots:** nombre de bots que participaran a la partida.
- **mode:** mode amb el qual es llançarà l'execució del programa. Pot ser de tres tipus:
    - Generació: elimina totes les carpetes i fitxers d'execucions anteriors per fer-ne de nous. Aquest mode serveix també per entrenar als bots.
    - Entrenament: entrena als bots. És com el tipus generation però sense eliminar els fitxers aconseguits fins al moment.
    - Jugar: posa els bots a jugar sense entrenar-los ni eliminar cap fitxer generat.
- **Dibuixar sensors:** habilita o deshabilita la visualització dels rajos de raycasting que li surten al bot de la cintura per detectar col·lisions.
- **Temps d'acció:** temps en milisegons. Indica cada quant les entrades i sortides de la xarxa neural processaran nova informació i per tant es duran a terme noves accions.
- **Temps de puntuació:** temps en milisegons. Indica cada quant es guardarà en un fitxer la puntuació, morts, morts i suïcidis que ha realitzat un bot.
- **Temps d'algorismes genètics:** temps en milisegons. Indica cada quant s'aturaran tots i s'aplicaran els algorismes evolutius d'elitisme, creuament i mutació.
- **Distància d'objectes:** unitats Unreal. Distància a partir de la qual els bots veuen els objectes.
- **Servidor:** permet connectar el projecte a un servidor concret al posar la seva ip. Per defecte la ip és localhost.
- **Port:** indica el port pel qual es connectaran els bots al servidor. El port per defecte per connectar els bots és el 3000.

9.3 Javadoc

PACKAGE COM.MYCOMPANY.MAVENPROJECT2

Class Summary	
Class	**Description**
Actions	Actions realized by the bot
AGGen	Generates new bots using old bots, mixing them and applying mutation
EnemyInfo	Creates a structure to store information about the enemies that kill the bot
EnemySelection	Helps to choose to which enemy the bot should shoot
FileManager	Manage all info files such as genetics, init net, input and output ANN values,..
Functions	Contains generic functions
Movement	Allows the bot to move around
SmartBot	Creates, executes and controls the bot

CLASS ACTIONS

java.lang.Object

com.mycompany.mavenproject2.Actions

```
public class Actions
extends Object
```

Actions realized by the bot

Method Summary

Methods	
Modifier and Type	**Method and Description**
void	**resetEnemy**() reset enemy target
void	**setAccuracy**(int _accuracy) Sets the aim accuracy bot
void	**shootPrimary**(**Player** _player, **Weaponry** _weaponry, **String** _weaponName, boolean _incoming, **IncomingProjectile** _projectile, **AgentInfo** _info) shots with the primary shot to the enemy.
void	**shootSecondary**(**Player** _player) shoots with the secondary fire
void	**stopShooting**() Stops shooting

Method Detail

resetEnemy

```
public void resetEnemy()
```

reset enemy target

setAccuracy

```
public void setAccuracy(int _accuracy)
```

Sets the aim accuracy bot

Parameters:

`_accuracy` - Int

shootPrimary

```
public void shootPrimary(Player _player, Weaponry _weaponry,
String _weaponName,boolean _incoming,IncomingProjectile
_projectile,AgentInfo _info)
```

shots with the primary shot to the enemy. The weapon is selected by the desire of the bot.

Parameters:

`_player` - player. Enemy to shoot

`_weaponry` - weaponry. information related to the bot's weapons and ammo

`_weaponName` - String. Selected weapon

`_incoming` - boolean. If there's a projectile flying

`_projectile` - projectile.

`_info` - AgentInfo.

shootSecondary

```
public void shootSecondary(Player _player)
```

shoots with the secondary fire

Parameters:

`_player` - Player. Enemy to shoot

stopShooting

```
public void stopShooting()
```

Stops shooting

CLASS AGGEN

java.lang.Object

com.mycompany.mavenproject2.AGGen

```
public class AGGen
```

```
extends Object
```

Generates new bots using old bots, mixing them and applying mutation

Author: Joan Marc Llargués Asensio

Constructor Summary

Constructors
Constructor and Description
`AGGen()`

Method Summary

Methods	
Modifier and Type	**Method and Description**
`static String`	`generate_genetic_file(String _folder, int _generation, long[] _bots)` This function generates the genetic file which contains all bot's information
`static ArrayList<float[]>`	`genetics(String file, long[] _bots_folder,boolean[] _boolean_bots)` Reads the main file with all the genetic's information and modifies them
`static float[]`	`Mutacio(float[] chromosome)` modifies the chromosome

Constructor Detail

AGGen

```
public AGGen()
```

Method Detail

generate_genetic_file

```
public static String generate_genetic_file(String _folder,
int _generation,long[] _bots)
```

This function generates the genetic file which contains all bot's information

Parameters:

`_folder` - String Route to the main folder where belongs all the genetics information

`_generation` - Int Current value of the actual generation

`_bots` - Array Long Contains the main id of each bot

Returns:

String File name containing the chromosome info of all bots

genetics

```
public static ArrayList<float[]> genetics(String file,
long[] _bots_folder,boolean[] _boolean_bots)
```

Reads the main file with all the genetic's information and modifies them

Parameters:

`file` - String File which contains all bot's information

`_bots_folder` - long[].

`_boolean_bots` - long[].

Returns:

ArrayList float Contains all modified chromosomes from each bot.

Mutacio

```
public static float[] Mutacio(float[] chromosome)
```

modifies the chromosome

Parameters:

`chromosome` - []

Returns:

Array float containing the modified chromosome info

CLASS ENEMYINFO

java.lang.Object

com.mycompany.mavenproject2.EnemyInfo

```
public class EnemyInfo
extends Object
```

Creates a structure to store information about the enemies that kill the bot

Author:

Joan Marc Llargués Asensio

Method Summary

Methods	
Modifier and Type	**Method and Description**
void	**addDeath**() Increases number of deaths in 1
int	**getDeaths**() Returns number of death's
String	**getName**() Returns enemy's name
float	**getPriority**() Returns the priority value of the enemy
double	**getTime**() Returns the time since last kill
void	**lessPriority**() Decreasses enemy priority
void	**morePriority**(float _p) Increases the priority of the enemy
void	**setName**(**String** _name) Sets enemy's name
void	**setTime**(long _time) Sets in which moment the enemy killed the bot

Method Detail

setName

```
public void setName(String _name)
```

Sets enemy's name

Parameters:

`_name` - String. Sets enemy's name

addDeath

```
public void addDeath()
```

Increases number of deaths in 1

setTime

```
public void setTime(long _time)
```

Sets in which moment the enemy killed the bot

Parameters:

`_time` - Long. Time in milliseconds

morePriority

```
public void morePriority(float _p)
```

Increases the priority of the enemy

Parameters:

`_p` - float. Increase enemy priority value

lessPriority

```
public void lessPriority()
```

Decreasses enemy priority

getName

```
public String getName()
```

Returns enemy's name

Returns:

String.

getDeaths

```
public int getDeaths()
```

Returns number of death's

Returns:

int

getTime

```
public double getTime()
```

Returns the time since last kill

Returns:

double.

getPriority

```
public float getPriority()
```

Returns the priority value of the enemy

Returns:

float.

CLASS ENEMYSELECTION

java.lang.Object

com.mycompany.mavenproject2.EnemySelection

```
public class EnemySelection
extends Object
```

Helps to choose to which enemy the bot should shoot

Author:

Joan Marc Llargués Asensio

Method Summary

Methods	
Modifier and Type	**Method and Description**
Player	chooseEnemy(Players _players) returns the enemy to shoot
Player	getEnemy()
void	printLog(long _bot_num) Displays the information of every enemy that have killed the bot
void	resetEnemy() reset enemy value
void	setEnemy(Player _enemy) Sets the enemey
void	updateInfo(String _name, long _time) updates the bot's time

Method Detail printLog

```
public void printLog(long _bot_num)
```

Displays the information of every enemy that have killed the bot

Parameters:

_bot_num - long. Number of bot to retrieve the information

updateInfo

```
public void updateInfo(String _name,long _time)
```

updates the bot's time

Parameters:

`_name` - String.

`_time` - long.

setEnemy

```
public void setEnemy(Player _enemy)
```

Sets the enemey

Parameters:

`_player` -

resetEnemy

```
public void resetEnemy()
```

reset enemy value

getEnemy

```
public Player getEnemy()
```

chooseEnemy

```
public Player chooseEnemy(Players _players)
```

returns the enemy to shoot

Parameters:

`_players` - players.

Returns:

Player.

CLASS FILEMANAGER

java.lang.Object

com.mycompany.mavenproject2.FileManager

```
public class FileManager
extends Object
```

Manage all info files such as genetics, init net, input and output ANN values,..

Author:

Joan Marc Llargués Asensio

Constructor Summary

Constructors	
Constructor and Description	
`FileManager()`	

Method Summary

Methods	
Modifier and Type	**Method and Description**
`static void`	`botStatistics(long _id_num, int _deaths, int _suicides, int _killedOthers)` This function writes in a file the gameplay information of a bot.
`static void`	`clearBotsNames()` resets the file that contains current game bots names
`static void`	`clearFitness()` Resets the fitness file
`static void`	`clearKDS()` clear the file that contains Kills-Deaths-Suicides
`static void`	`clearScoreJM(long bot)` resets the score file
`static void`	`clearScoreJuan(long bot)` resets the score file
`static void`	`create_folder(String target)` Creates a new folder with the bot id and copy the required files
`static void`	`currentGenerationValue(int generation)`

	Stores the current generation value
boolean	**existBotName**(**String** name) Checks if name exist in the file that cointains the names of the bots.
static int	**getCurrentGenerationValue**() Returns the current generation value
long[]	**getStatistics**(int generation, long[] _bots_folder, **String**[] name) This functions returns an array with the fitness value of each bot sorted by the biggest value
static void	**guardarBot**(int generation, long id_bot) This functions creates a backup of the current bot before apply genetic algorithms
static void	**Init ANN creator**(**String** _folder, long _id_num, int _input, int _hidden, int _output) creates the Init_ANN file with randowm weights
static **ArrayList**<**Float**>	**Init ANN parser**(**String** _folder, long _id_num) This function parses the Init_ANN file of a bot and returns its values inside an array
static void	**new generations file**(**String** _folder, long[] _bots, int _input, int _hidden, int _output, **ArrayList**<float[]> _genetics) This function creates new Init_ANN files after the bots have been selected and modified.
static void	**prepare thread**(**String** _date, long _id_num, int _input_ANN, int _hidden_ANN, int _output_ANN) Creates and fills a txt file with the configuration of the ANN and some batch files to execute them
static void	**printScoreJM**(long bot, int generation, int score, int kills, int deaths, int suicides) Stores the score of the bot
static void	**printScoreJuan**(long bot, int generation, int score, int kills, int deaths, int suicides) Stores the score of the bot
static void	**resetBotStatistics**(long _id_num) This function resets gameplay information of a bot.

Constructor Detail

FileManager

```
public FileManager()
```

Method Detail

create_folder

```
public static void create_folder(String target)
```

Creates a new folder with the bot id and copy the required files

Parameters:

`target` - String. Target folder (bot's id)

prepare_thread

```
public static void prepare_thread(String _date,long _id_num,
int _input_ANN,int _hidden_ANN,int _output_ANN)
```

Creates and fills a txt file with the configuration of the ANN and some batch files to execute them

Parameters:

`_date` - String. String with the current date

`_id_num` - long. Bot's id thread.

`_input_ANN` - int. Number of input nodes

`_hidden_ANN` - int. Number of hidden nodes

`_output_ANN` - int. Number of output nodes

Init_ANN_creator

```
public
static void Init_ANN_creator(String _folder,long _id_num,
int _input,int _hidden,int _output)
```

creates the Init_ANN file with randowm weights

Parameters:

`_folder` - String. Root target folder

`_id_num` - int. Bot's id folder

`_input` - int. Number of input nodes

`_hidden` - int. Number of hidden nodes

`_output` - int. Number of output nodes

Init_ANN_parser

```
public static ArrayList<Float> Init_ANN_parser(String _folder,
long _id_num)
```

This function parses the Init_ANN file of a bot and returns its values inside an array

Parameters:

`_folder` - String. Root folder.

`_id_num` - long. id of the folder that belongs to the selected bot.

Returns:

new_generations_file

```
public
static void new_generations_file(String _folder,long[] _bots,
int _input,int _hidden,int _output,ArrayList<float[]>
_genetics)
```

This function creates new Init_ANN files after the bots have been selected and modified.

Parameters:

`_folder` - String. Root folder.

`_bots` - Long. Array with the folder id's of each bot.

`_input` - int. Number of input nodes.

`_hidden` - int. Number of hidden neuron nodes.

`_output` - int. Number of output nodes.

`_genetics` - ArrayList. Array containing all the chromosomes.

botStatistics

```
public static void botStatistics(long _id_num,int _deaths,
int _suicides,int _killedOthers)
```

This function writes in a file the gameplay information of a bot.

Parameters:

`_id_num` - long. id of the bot.

`_deaths` - int. Number of deaths

`_suicides` - int. number of suicides

`_killedOthers` - int. number of kills

resetBotStatistics

```
public static void resetBotStatistics(long _id_num)
```

This function resets gameplay information of a bot.

Parameters:

`_id_num` - long. id of the bot.

getStatistics

```
public long[] getStatistics(int generation,long[]
_bots_folder, String[] name)
```

This functions returns an array with the fitness value of each bot sorted by the biggest value

Parameters:

`generation` - int. Current generation value.

`_bots_folder` - long. Array containing all the bots ids folders

`name` - string. Names of the bots.

Returns:

long[].

clearBotsNames

```
public static void clearBotsNames()
```

resets the file that contains current game bots names

existBotName

```
public boolean existBotName(String name)
```

Checks if name exist in the file that cointains the names of the bots.

Parameters:

`name` - String.

Returns:

boolean.

clearFitness

```
public static void clearFitness()
```

Resets the fitness file

currentGenerationValue

```
public static void currentGenerationValue(int generation)
```

Stores the current generation value

Parameters:

`generation` - int. Current generation.

getCurrentGenerationValue

```
public static int getCurrentGenerationValue()
```

Returns the current generation value

Returns:

int

clearScoreJM

```
public static void clearScoreJM(long bot)
```

resets the score file

Parameters:

`bot` - long.

printScoreJM

```
public
static void printScoreJM(long bot,int generation,int score,
int kills,int deaths,int suicides)
```

Stores the score of the bot

Parameters:

`bot` - long. Current bot

`generation` - int. Current generation.

`score` - ing. Current bot's score

`kills` - ing. Current bot's kills

`deaths` - ing. Current bot's deaths

`suicides` - ing. Current bot's suïcides

clearKDS

```
public static void clearKDS()
```

clear the file that contains Kills-Deaths-Suicides

guardarBot

```
public static void guardarBot(int generation,long id_bot)
```

This functions creates a backup of the current bot before apply genetic algorithms

Parameters:

`generation` - int. current generation.

`id_bot` - long. Bot id.

clearScoreJuan

```
public static void clearScoreJuan(long bot)
```

resets the score file

Parameters:

`bot` - long.

printScoreJuan

```
public
static void printScoreJuan(long bot,int generation,int score,
int kills,int deaths,int suicides)
```

Stores the score of the bot

Parameters:

`bot` - long. Current bot

`generation` - int. Current generation.

`score` - ing. Current bot's score

`kills` - ing. Current bot's kills

`deaths` - ing. Current bot's deaths

`suicides` - ing. Current bot's suicides

CLASS FUNCTIONS

java.lang.Object

cz.cuni.amis.pogamut.ut2004.bot.impl.UT2004BotController<BOT>
cz.cuni.amis.pogamut.ut2004.bot.impl.UT2004BotLogicController<BOT>
cz.cuni.amis.pogamut.ut2004.bot.impl.UT2004BotModuleController
com.mycompany.mavenproject2.Functions

All Implemented Interfaces:

cz.cuni.amis.pogamut.base.agent.module.IAgentLogic, IUT2004BotController, IUT2004BotLogicController

---

```
public class Functions
extends UT2004BotModuleController
```

Contains generic functions

Constructor Summary

Constructors
Constructor and Description
Functions()

Method Summary

Methods	
Modifier and Type	**Method and Description**
double	**ammoBag**(**Weaponry** _weaponry, **String** weaponName) returns the amount of ammo of the selected weapon
boolean[]	**andDirection**(boolean[] IN_ANN_boolean, boolean[] OUT_ANN_boolean)
int	**angleDirection**(double[] OUT_ANN_double, boolean[] andDir) return the angle the bot should rotate
Boolean[]	**botsToBoolean**(long[] _sorted_bots_folder, long[] _sorted_bots) returns a boolean array that indicates which bot have to be modified.
double	**collideDistance**(int radius, cz.cuni.amis.pogamut.base3d.worldview.object.Location _bot, cz.cuni.amis.pogamut.base3d.worldview.object.Location collision) returns the distance between the bot and the location to check
void	**findItem**(**String** itemName) navigates to the nearest itemName (if its possible)

int	**fitnessWalk**(boolean[] IN_ANN_values, boolean[] OUT_ANN_boolean) Deprecated
Item	**getNearestPossiblySpawnedItemOfType**(**ItemType** type) return the nearets possibly spawned point of the selected item
boolean	**insideRadius**(int radius, cz.cuni.amis.pogamut.base3d.worldview.object.Location position, cz.cuni.amis.pogamut.base3d.worldview.object.Location target) checks if the target is inside the bots radius.
boolean	**isItemVisible**(**Items** _items, **String** typeName) checks if the item is visible or not.
String	**selectWeapon**(double out_15, double out_16, double out_17, boolean ratio, boolean primary, boolean secondary) returns the selected weapon
ItemType	**typeAmmo**(**String** weapon) return the type of ammo for the selected weapon name.
boolean	**weaponBag**(**Weaponry** _weaponry, **String** weaponName) checks if the bot owns a weapon.
String	**weaponrySorted**(**Weaponry** _weaponry, **String** sort) Sorts the weapons and returns the chosen one

Constructor Detail

Functions

```
public Functions()
```

Method Detail

weaponrySorted

```
public String weaponrySorted(Weaponry _weaponry,String sort)
```

Sorts the weapons and returns the chosen one

Parameters:

`_weaponry` - weaponry. weapons that the bot owns

`sort` - String. type of sorting

Returns:

String. selected weapon

typeAmmo

```
public ItemType typeAmmo(String weapon)
```

return the type of ammo for the selected weapon name.

Parameters:

`weapon` - String.

Returns:

ItemType.

andDirection

```
public boolean[] andDirection(boolean[] IN_ANN_boolean,boolean
[] OUT_ANN_boolean)
```

Parameters:

`IN_ANN_boolean` -

`OUT_ANN_boolean` -

Returns:

fitnessWalk

```
public int fitnessWalk(boolean[] IN_ANN_values,boolean[]
OUT_ANN_boolean)
```

Deprecated

Parameters:

`IN_ANN_values` -

`OUT_ANN_boolean` -

Returns:

angleDirection

```
public int angleDirection(double[] OUT_ANN_double,boolean[] an
dDir)
```

return the angle the bot should rotate

Parameters:

`OUT_ANN_double` - double[]. ANN outputs

`andDir` - boolean[]. booelan ANN outputs

Returns:

int. Degree

getNearestPossiblySpawnedItemOfType

```
public Item getNearestPossiblySpawnedItemOfType(ItemType type)
```

return the nearets possibly spawned point of the selected item

Parameters:

`type` - itemType.

Returns:

Item.

findItem

```
public void findItem(String itemName)
```

navigates to the nearest itemName (if its possible)

Parameters:

`itemName` - String.

isItemVisible

```
public boolean isItemVisible(Items _items,String typeName)
```

checks if the item is visible or not.

Parameters:

`_items` - Items.

`typeName` - String.

Returns:

boolean

weaponBag

```
public boolean weaponBag(Weaponry _weaponry,String weaponName)
```

checks if the bot owns a weapon.

Parameters:

`_weaponry` - weaponry.

`weaponName` - String.

Returns:

boolean.

ammoBag

```
public double ammoBag(Weaponry _weaponry,String weaponName)
```

returns the amount of ammo of the selected weapon

Parameters:

`_weaponry` - weaponry.

`weaponName` - String.

Returns:

double.

collideDistance

```
public double collideDistance(int radius,

cz.cuni.amis.pogamut.base3d.worldview.object.Location _bot,

cz.cuni.amis.pogamut.base3d.worldview.object.Location collisio
n)
```

returns the distance between the bot and the location to check

Parameters:

`radius` - int.

`_bot` - Location.

`collision` - Location.

Returns:

double.

insideRadius

```
public boolean insideRadius(int radius,

cz.cuni.amis.pogamut.base3d.worldview.object.Location position
,

cz.cuni.amis.pogamut.base3d.worldview.object.Location target)
```

checks if the target is inside the bots radius.

Parameters:

`radius` - int.

`position` - Location.

`target` - Locatoin.

Returns:

boolean.

botsToBoolean

```
public boolean[] botsToBoolean(long[] _sorted_bots_folder,
long[] _sorted_bots)
```

returns a boolean array that indicates which bot have to be modified.

Parameters:

`_sorted_bots_folder` - long[]

`_sorted_bots` - long[]

Returns:

boolean[]

selectWeapon

```
public String selectWeapon(double out_15,double out_16,double
out_17,
boolean ratio,boolean primary,boolean secondary)
```

returns the selected weapon

Parameters:

`out_15` - double. ANN output

`out_16` - double. ANN output

`out_17` - double. ANN output

`ratio` - boolean. ANN output

`primary` - boolean. ANN output

`secondary` - boolean. ANN output

Returns:

String.

CLASS MOVEMENT

java.lang.Object

com.mycompany.mavenproject2.Movement

```
public class Movement
extends Object
```

Allows the bot to move around

Constructor Summary

Constructors	
Constructor and Description	
Movement(**AdvancedLocomotion** _move) Constructor	

Method Summary

Methods	
Modifier and Type	**Method and Description**
void	**dodge**(cz.cuni.amis.pogamut.base3d.worldview.object.ILocated botPosition, cz.cuni.amis.pogamut.base3d.worldview.object.ILocated inFrontOfTheBot) the bot dodges.
void	**dodgeBack**(cz.cuni.amis.pogamut.base3d.worldview.object.ILocated botPosition, cz.cuni.amis.pogamut.base3d.worldview.object.ILocated inFrontOfTheBot) the target dodges back
void	**dodgeLeft**(cz.cuni.amis.pogamut.base3d.worldview.object.ILocated botPosition, cz.cuni.amis.pogamut.base3d.worldview.object.ILocated inFrontOfTheBot) the bot dodges left.
void	**dodgeRight**(cz.cuni.amis.pogamut.base3d.worldview.object.ILocated botPosition, cz.cuni.amis.pogamut.base3d.worldview.object.ILocated inFrontOfTheBot) the bot dodges right
void	**DoubleJump**() the bot jump double

void	**Jump**() the bot jump
void	**Move**() the bot move forward
void	**StopMove**() the bot stops
void	**strafeLeft**(double distance) the bot strafes left a distance
void	**strafeRight**(double distance) the bot strafes right a distance
void	**turnHoritzontal**(int angle) the bot rotates angle degrees

Constructor Detail

Movement

```
public Movement(AdvancedLocomotion _move)
```

Constructor

Parameters:

`_move` - AdvancedLocomotion.

Method Detail

Jump

```
public void Jump()
```

the bot jump

DoubleJump

```
public void DoubleJump()
```

the bot jump double

Move

```
public void Move()
```

the bot move forward

StopMove

```
public void StopMove()
```

the bot stops

turnHoritzontal

```
public void turnHoritzontal(int angle)
```

the bot rotates angle degrees

Parameters:

`angle` - int.

strafeLeft

```
public void strafeLeft(double distance)
```

the bot strafes left a distance

Parameters:

`distance` - double.

strafeRight

```
public void strafeRight(double distance)
```

the bot strafes right a distance

Parameters:

`distance` - double.

dodgeLeft

```
public void dodgeLeft(cz.cuni.amis.pogamut.base3d.worldview.object.ILocated botPosition,

cz.cuni.amis.pogamut.base3d.worldview.object.ILocated inFrontOfTheBot)
```

the bot dodges left.

Parameters:

`botPosition` - ILocated. current bot position.

`inFrontOfTheBot` - ILocated. target to dodge

dodgeRight

```
public void dodgeRight(cz.cuni.amis.pogamut.base3d.worldview.object.ILocated botPosition,

cz.cuni.amis.pogamut.base3d.worldview.object.ILocated inFrontOfTheBot)
```

the bot dodges right

Parameters:

`botPosition` - ILocated. current bot position.

`inFrontOfTheBot` - ILocated. target to dodge

dodgeBack

```
public void dodgeBack(cz.cuni.amis.pogamut.base3d.worldview.object.ILocated botPosition,

cz.cuni.amis.pogamut.base3d.worldview.object.ILocated inFrontOfTheBot)
```

the target dodges back

Parameters:

`botPosition` - ILocated. current bot position.

`inFrontOfTheBot` - ILocated. target to dodge

dodge

```
public void dodge(cz.cuni.amis.pogamut.base3d.worldview.object.ILocated botPosition,

cz.cuni.amis.pogamut.base3d.worldview.object.ILocated inFrontOfTheBot)
```

the bot dodges.

Parameters:

`botPosition` - ILocated. current bot position.

`inFrontOfTheBot` - ILocated. target to dodge

CLASS SMARTBOT

java.lang.Object

cz.cuni.amis.pogamut.ut2004.bot.impl.UT2004BotController<BOT>
cz.cuni.amis.pogamut.ut2004.bot.impl.UT2004BotLogicController<BOT>
cz.cuni.amis.pogamut.ut2004.bot.impl.UT2004BotModuleController
com.mycompany.mavenproject2.SmartBot

All Implemented Interfaces:

cz.cuni.amis.pogamut.base.agent.module.IAgentLogic, IUT2004BotController, IUT2004BotLogicController

---

```
public class SmartBot
extends UT2004BotModuleController
```

Creates, executes and controls the bot

Field Summary

Fields	
Modifier and Type	**Field and Description**
protected AgentInfo	agentInfo
boolean	bot_paused
protected static String	DOWNLong
long	elapsedBotTime
long	elapsedMasterBotTime
long	elapsedScoreTime
protected static String	FRONT
protected static String	FRONTDOWN30
protected static String	FRONTUP30
protected List<Item>	itemsToRunAround
protected static String	LEFT30
protected static String	LEFT60
protected static String	LEFT90
protected static String	RIGHT30
protected static String	RIGHT60
protected static String	RIGHT90
long	scoreTime
long	startBotTime

long	**startMasterBotTime**
protected **TabooSet**<**Item**>	**tabooItems**
protected static **String**	**UPLong**

Constructor Summary

Constructors
Constructor and Description
SmartBot()

Method Summary

Methods	
Modifier and Type	**Method and Description**
void	**ANN**() executes the ANN proces
void	**beforeFirstLogic**() called after botFirstSpawn method, right before the first call of logic() method.
void	**botFirstSpawn**(**GameInfo** gameInfo, **ConfigChange** config, **InitedMessage** init, **Self** self) called when the bot is spawned in the game for the first time.
void	**botInitialized**(**GameInfo** gameInfo, **ConfigChange** currentConfig, **InitedMessage** init) called when the bot received INITED message from the server.
void	**botKilled**(**BotKilled** event) called each time the bot has died
Initialize	**getInitializeCommand**() used for setting initial properties of the agent such as it's name, starting location etc.
Item	**getItem**(int _distance, **ItemType** _object) returns the nearest item selected
static **String**	**getSkin**() returns a skin for the bot
void	**logic**() method called periodically by an internal thread associated with the bot.
static void	**main**(**String**[] args)
static **String**	**nameBot**() returns a name for the bot
String	**prepare in ANN**() prepares the input string to be writen at the file
void	**prepareBot**(**UT2004Bot** bot) Called before the bot connects to the environment, but after UT2004Bot is constructed.

void	**reset_in_ANN**() reset ANN inputs
void	**reset_out_ANN**() reset ANN outputs
void	**set_out_ANN**(double[] out_ANN) reads the ANN output values and decides if its enabled or not by a threshold
void	**smartbot_does**() performs the triggered actions

Field Detail

bot_paused

```
public boolean bot_paused
```

startBotTime

```
public long startBotTime
```

startMasterBotTime

```
public long startMasterBotTime
```

elapsedBotTime

```
public long elapsedBotTime
```

elapsedMasterBotTime

```
public long elapsedMasterBotTime
```

scoreTime

```
public long scoreTime
```

elapsedScoreTime

```
public long elapsedScoreTime
```

FRONT

```
protected static final String FRONT
```

See Also:

Constant Field Values

LEFT30

```
protected static final String LEFT30
```

See Also:

Constant Field Values

LEFT60

```
protected static final String LEFT60
```

See Also:

Constant Field Values

LEFT90

```
protected static final String LEFT90
```

See Also:

Constant Field Values

RIGHT30

```
protected static final String RIGHT30
```

See Also:

Constant Field Values

RIGHT60

```
protected static final String RIGHT60
```

See Also:

Constant Field Values

RIGHT90

```
protected static final String RIGHT90
```

See Also:

Constant Field Values

UPLong

```
protected static final String UPLong
```

See Also:

Constant Field Values

DOWNLong

```
protected static final String DOWNLong
```

See Also:

Constant Field Values

FRONTUP30

```
protected static final String FRONTUP30
```

See Also:

Constant Field Values

FRONTDOWN30

```
protected static final String FRONTDOWN30
```

See Also:

Constant Field Values

agentInfo

```
protected AgentInfo agentInfo
```

itemsToRunAround

```
protected List<Item> itemsToRunAround
```

- ***tabooItems***

```
protected TabooSet<Item> tabooItems
```

Constructor Detail

SmartBot

```
public SmartBot()
```

Method Detail

prepareBot

```
public void prepareBot(UT2004Bot bot)
```

Called before the bot connects to the environment, but after UT2004Bot is constructed. This is the right place to initialize Pogamut bot modules

Specified by:

prepareBot in interface IUT2004BotController

Overrides:

prepareBot in class UT2004BotController

Parameters:

bot -

getInitializeCommand

```
public Initialize getInitializeCommand()
```

used for setting initial properties of the agent such as it's name, starting location etc. This method is used by Pogamut to get the initialize command for the bot.

Specified by:

getInitializeCommand in interface IUT2004BotController

Overrides:

getInitializeCommand in class UT2004BotController

Returns:

botInitialized

```
public void botInitialized(GameInfo gameInfo,ConfigChange
currentConfig,InitedMessage init)
```

called when the bot received INITED message from the server. This means that the INIT command succeeded, the handshake between bot and server is over and the bot is ready to receive other commands. Note that this does not mean the bot is already spawned in the environment! If it is set bAutoSpawn=False in GameBots ini file, the bot won't be spawned automatically and user needs to issue command Respawn to spawn the bot. Then botInitialized() method would be the right place to send first Respawn command. Now briefly about method parameters. info object holds information about the game type, map etc. config object holding information about bot current configuration - his vision time delay, autotracing setting, spawn setting etc. init object holds information about bot variables such as his speed in the environment, maximum possible health etc.

Specified by:

botInitialized in interface IUT2004BotController

Overrides:

botInitialized in class UT2004BotController

Parameters:

gameInfo -

currentConfig -

init -

botFirstSpawn

```
public void botFirstSpawn(GameInfo gameInfo,ConfigChange
config,InitedMessage init,Self self)
```

called when the bot is spawned in the game for the first time. This means that the bot graphical represantation is visible in the game and Selfobject was received from the environment holding information about bot location and current state. This is the last place to do some preparations before logic() method will be called periodically.

Specified by:

botFirstSpawn in interface IUT2004BotController

Overrides:

botFirstSpawn in class UT2004BotController

Parameters:

gameInfo -

config -

init -

self -

beforeFirstLogic

```
public void beforeFirstLogic()
```

called after botFirstSpawn method, right before the first call of logic() method. This method is an ideal place to make last preparations of your custom modules before the logic() method gets executed. The advantage here is that when this method gets called your bot is fully initialized and present in the environment.

Specified by:

beforeFirstLogic in interface cz.cuni.amis.pogamut.base.agent.module.IAgentLogic

Overrides:

beforeFirstLogic in class UT2004BotLogicController

botKilled

```
public void botKilled(BotKilled event)
```

called each time the bot has died

Specified by:

botKilled in interface IUT2004BotController

Overrides:

botKilled in class UT2004BotController

Parameters:

event -

logic

```
public void logic()throws
cz.cuni.amis.utils.exception.PogamutException
```

method called periodically by an internal thread associated with the bot. It enables the bot to be proactive, that is to act "on his own" without any external stimuli.

Specified by:

logic in interface cz.cuni.amis.pogamut.base.agent.module.IAgentLogic

Overrides:

logic in class UT2004BotLogicController

Throws:

cz.cuni.amis.utils.exception.PogamutException

reset_in_ANN

```
public void reset_in_ANN()
```

reset ANN inputs

reset_out_ANN

```
public void reset_out_ANN()
```

reset ANN outputs

prepare_in_ANN

```
public String prepare_in_ANN()
```

prepares the input string to be writen at the file

Returns:

ANN

```
public void ANN()
```

executes the ANN proces

set_out_ANN

```
public void set_out_ANN(double[] out_ANN)
```

reads the ANN output values and decides if its enabled or not by a threshold

Parameters:

`out_ANN` - double[]

smartbot_does

```
public void smartbot_does()
```

performs the triggered actions

getItem

```
public Item getItem(int _distance,ItemType _object)
```

returns the nearest item selected

Parameters:

`_distance` - int.

`_object` - itemType

Returns:

nameBot

```
public static String nameBot()
```

returns a name for the bot

Returns:

String

getSkin

```
public static String getSkin()
```

returns a skin for the bot

Returns:

String.

main

```
public static void main(String[] args)throws
cz.cuni.amis.utils.exception.PogamutException
```

Parameters:

`args` -

Throws:

`cz.cuni.amis.utils.exception.PogamutException`

10. Referències bibliogràfiques

[1] NAVARRO-ARRIBAS, Guillermo i MEGÍAS, David - *Modeling decisions for AI* [en línia]. Disponible a la World Wide Web: < http://www.mdai.cat/mdai2013/MDAI2013.pdf > [consulta 20-22 de novembre de 2012].

[2] Wikipedia - Unreal Tournament 2004 [en línia]. Disponible a la World Wide Web: < http://es.wikipedia.org/wiki/Unreal_Tournament_2004 > [consulta 18 d'abril de 2013].

[3] JAIN, Anil K. - *Artificial Neural Networks: a tutorial* (març de 1996) [en línia]. Disponible a la World Wide Web: < http://web.iitd.ac.in/~sumeet/Jain.pdf > [consulta 19 de març de 2013].

[4] LLARGUÉS ASENSIO, Joan Marc (2012). *Modelitzant l'evolució*.

[5] MILLINGTON, Ian. *Artificial Intelligence for Games*. ISBN 978-0-12-497782-2.

[6] Pogamut [en línia]. Disponible a la World Wide Web: < http://pogamut.cuni.cz/main/tiki-index.php > [consulta 21 de febrer de 2013].

[7] MICHALEWICZ, Zbigniew - *Heuristic Methods for Evolutionary Computation Techniques.* [en línia]. Disponible a la World Wide Web: < http://cs.adelaide.edu.au/~zbyszek/Papers/p20.pdf> [consulta 20 de març de 2013].

[8] Genetic algorithms . [en línia]. Disponible a la World Wide Web: <http://www.geneticprogramming.com/ga/index.htm > [consulta 17 d'abril de 2013].

[9] WEISE, Thomas. *Global optimitzation algorithms* (2009). [en línia]. Disponible a la World Wide Web: < http://www.it-weise.de/projects/book.pdf > [consulta 8 de maig de 2013].

[10] Mario AI championship [en línia]. Disponible a la World Wide Web: < http://www.marioai.org/ > [consulta 2 de març de 2013].

[11] Neural Bot [en línia]. Disponible a la World Wide Web:

< http://homepages.paradise.net.nz/nickamy/neuralbot/nb_about.htm > [consulta 15 d'abril de 2013].

[12] Botprize championship [en línia]. Disponible a la World Wide Web: < http://botprize.org/ > [consulta 6 de maig de 2013].

[13] Conscious-robots [en línia]. Disponible a la World Wide Web: < http://www.conscious-robots.com/ > [consulta 18 de maig de 2013].

[14] Conscious-robots papers [en línia]. Disponible a la World Wide Web: < http://www.conscious-robots.com/en/publications/papers/3.html > [consulta 22 de maig de 2013].

[15] Netbeans [en línia]. Disponible a la World Wide Web: < https://netbeans.org/ >

[16] CYBENKO, G. V. (1989). *Approximation by superpositions of a sigmoidal function.*

[17] ROSENBLATT, Frank. (1962). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.*

[18] CYBENKO, G. V. *Artificial Neural Networks: an introduction*. (apartats 7.2.4 i 7.2.5). [en línia]. Disponible a la World Wide Web: < http://tinyurl.com/cybenkoPFC > [consulta 4 de maig de 2013].

[19] CYBENKO, G. V. *Approximation Capabilities of Feedforward Neural Networks for Continuous Functions*. [en línia]. Disponible a la World Wide Web: < http://neuron.eng.wayne.edu/tarek/MITbook/chap2/2_3.html > [consulta 4 de maig de 2013].

[20] PERALTA, J; GUTIERREZ, G; SANCHIS, A. PhD Thesis. (maig de 2012). *Diseño automatico de redes de neuronas artificiales para la prediccion de series temporales.* Carlos III Universidad de Madrid.

[21] SRINIVASAN, S; MITAL, D.P; HAQUE, S. *A novel solution for maze tranversal problems using Aritifical neural networks.*

[22] WERBOS, Paul J; PANG, Xiaozhong. *Generalized Maze Navigation: SRN Critics Solve What Feedforward or Hebbian Nets Cannot*. [en línia]. Disponible a la World Wide Web: < http://www.werbos.com/Neural/WCNN96_SRN_SMC.htm > [consulta 10 de maig de 2013].