GENERAL GAME PLAYING: ANÁLISIS DEL PROBLEMA. SOLUCIÓN BASADA EN UN ALGORITMO HÍBRIDO DE INTELIGENCIA COMPUTACIONAL

GENERAL GAME PLAYING: PROBLEM ANALYSIS. HYBRID COMPUTATIONAL INTELLIGENCE ALGORITHM BASED SOLUTION

Realizado por
ARIEL EDUARDO VÁZQUEZ NÚÑEZ
Tutorizado por
ANTONIO JOSÉ FERNÁNDEZ LEIVA
Departamento
LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD DE MÁLAGA

MÁLAGA, SEPTIEMBRE 2015

Fecha defensa:
El Secretario del Tribunal

INGENIERÍA DEL SOFTWARE

Resumen: El objetivo de este trabajo es presentar unas bases de conocimiento sobre el denominado *General Game Playing* (GGP) analizando los conceptos relacionados con esta área que ha surgido recientemente, de forma que nuestro trabajo pueda ser usado como base en futuras investigaciones y tesis relacionadas con la materia. Para ello, se hará un estudio de los enfoques que se han empleado para abordar el problema y se profundizará en otras técnicas algorítmicas, tales como por ejemplo la de *Montecarlo Tree Search* y los algoritmos bio-inspirados que no se han empleado (o se han empleado poco) en este contexto. Adicionalmente, se realiza una propuesta de un agente autónomo (es decir, un resolutor del problema), implementando un algoritmo bio-inspirado mixto, dentro de la plataforma para la *General Video Game Artificial Intelligence Competition* (GVGAI), analizando sus resultados y extrayendo conclusiones.

Palabras clave: Inteligencia artificial, General Game Playing, Montecarlo Tree Search, algoritmos evolutivos, optimización, atari 2600

Abstract: The objective of this final Project is to present some knowledge basis about General Game Playing (GGP), analyzing related concepts of this new research area in way that this work can be used as base for future research and thesis about the matter. To achieve this objective, actual approaches of the problem will be studied, focusing on other algorithmic techniques, such Montecarlo Tree Seach (MCTS) and bio-inspired algorithms. In addition, a solution based on bio-inspired algorithms will be presented, based on the General Video Game Artificial Intelligence Competition (GVGAI) Framework.

Keywords: Artificial intelligence, General Game Playing, Montecarlo Tree Search, evolutionary algorithms, optimization, atari 2600

ÍNDICE

1 INTRODUCCIÓN

En esta sección se informará, de forma resumida, del contenido de esta memoria de proyecto titulada “General Game Playing: Análisis del problema. Solución basada en un algoritmo híbrido de inteligencia computacional”.

1.1 OBJETIVOS

El objetivo de este trabajo es presentar unas bases de conocimiento sobre el denominado *General Game Playing* (GGP) analizando los conceptos relacionados con esta área que ha surgido recientemente, de forma que nuestro trabajo pueda ser usado como base en futuras investigaciones y tesis relacionadas con la materia. Para ello, se hará un estudio de los enfoques que se han empleado para abordar el problema y se profundizará en otras técnicas algorítmicas, tales como por ejemplo la de *Montecarlo Tree Search* y los algoritmos bio-inspirados que no se han empleado (o se han empleado poco) en este contexto. Adicionalmente, trabajaremos en una propuesta de un agente autónomo, implementando un algoritmo bio-inspirado mixto, dentro de la plataforma para la *General Video Game Artificial Intelligence* (GVGAI) Competition, analizando sus resultados y extrayendo conclusiones.

1.2 CONTENIDO DE LA MEMORIA EN CAPÍTULOS

La organización de la memoria se ha estructurado de tal forma que se asemeje a las etapas seguidas en cualquier desarrollo de software, permitiendo al lector observar la evolución de manera iterativa del planteamiento expuesto en la introducción.

1.2.1 CONCEPTOS PRELIMINARES

En esta sección se revisarán los principales conceptos relacionados con el *General Game Playing*, cuyo conocimiento ayudan a la comprensión del estado actual de esta investigación y del alcance de este proyecto.

1.2.2 DISEÑO E IMPLEMENTACIÓN

En esta fase del proyecto se implementarán varios controladores utilizando el Framework de la *General Video Game AI Competition* (GVGAI), aplicando técnicas como *Montecarlo Tree Search*, algoritmos genéticos, y versiones híbridas de estos.

1.2.3 EXPERIMENTACIÓN Y RESULTADOS

Tras la implementación, se analizarán los experimentos realizados, así como los resultados obtenidos por los diferentes algoritmos en los mismos, comparando sus rendimientos en los diferentes juegos

1.2.4 CONCLUSIONES

En esta última sección se extraerán las conclusiones obtenidas de los resultados anteriores, así como las conclusiones generales y personales obtenidas durante todo el proceso de desarrollo de este proyecto

1.2.5 TECNOLOGÍAS UTILIZADAS

A continuación se listan las tecnologías y herramientas utilizadas durante el desarrollo de este proyecto:

- **Lenguaje de programación:** El desarrollo de los controladores se ha realizado en el lenguaje Java, en concreto, utilizando JDK 1.7, debido a las exigencias del framework de la *General Video Game Playing AI Competition* (GVGAI)
- **Entorno de desarrollo:** El entorno de desarrollo (IDE) escogido ha sido eclipse, debido a su facilidad de instalación y la experiencia personal acumulada en su utilización.
- **Frameworks específicos:** El desarrollo al completo se ha realizado utilizando el framework específico de la GVGAI, que permite programar controladores y ejecutarlos en diferentes juegos.
- **Análisis estadísticos:** Para el tratar la información obtenida en los experimentos realizados, se ha utilizado MATLAB, debido a su potencial a la hora de tratar con datos y generar gráficos.
- **Test estadísticos:** Para la realización de los test estadísticos no paramétricos, se ha utilizado un software programado en Java, desarrollado por el grupo de investigación de Francisco Herrera, de la universidad de Granada.

2 CONCEPTOS PRELIMINARES

En esta sección se revisarán los conceptos cuyo conocimiento resulta relevante para la comprensión del proyecto

2.1 GENERAL GAME PLAYING

General Game Playing (GGP) [Levine y otros, 2013] es una parte de la Inteligencia Artificial (IA) cuyo objetivo es construir un programa (denominado igualmente como agente o bot) que pueda jugar de manera eficaz a diferentes tipos de juegos sin necesidad de realizar cambios en el programa y sin entrenarlo previamente en estos. La única información que se le proporciona al agente/bot son las reglas del juego, y a partir de ahí se asume que debe ser capaz de jugar por sí mismo (lo que en realidad es lo que hacemos los humanos). La GGP nos presenta un problema que entra en la categoría de la obtención de Inteligencias Artificiales que muestren un comportamiento humano.

Bajo un enfoque clásico, cada juego requiere una implementación de un algoritmo diferente, en el que se planifica una estrategia teniendo en función del dominio del juego. Debido a la especialización de la implementación, es imposible utilizar el mismo programa en diferentes juegos, o incluso podría ser inservible dentro del mismo juego, si se realizase algún cambio en sus reglas. Por ejemplo, un programa podría ser muy bueno jugando al ajedrez, pero resultar desastroso si le presentamos alguna de las múltiples variaciones de este.

Entre los juegos que se utilizan como marco para el estudio de la GGP podemos considerar los juegos clásicos de la consola ATARI 2600, entre los cuales se encuentran títulos mundialmente conocidos como Pac-man, Asteroids, o Space Invaders.

Uno de los pioneros en General Game Playing es el *Stanford Logic Group* de la universidad de Stanford, California. Con el objetivo de profundizar en este campo, crearon una plataforma propia para el GGP. En esta plataforma, la representación de los juegos se realiza mediante una serie de reglas, definidas bajo un lenguaje propio llamado *Game Description Language* (GDL) [Ebner y otros, 2013]. De esta forma, se unifica la representación de los juegos bajo un único lenguaje de forma que cualquier jugador pueda obtener la información completa del juego a través de sus reglas. Además, usando esta plataforma, crearon una competición que se celebra anualmente en la que los participantes presentan sus propios controladores, que compiten entre sí. La *General Video Game AI Competition* (GVGAI Competition).

2.2 ATARI 2600 COMO MARCO DEL GENERAL VIDEO GAME PLAYING

Para estudiar en profundidad el campo del *General Video Game Playing*, se ha escogido la consola clásica *Atari 2600* y sus juegos como marco sobre el que realizar el desarrollo y experimentación. Se trata de un marco idóneo ya que se trata de juegos cuyas reglas son fáciles de comprender y modelar, y apenas requieren un mínimo procesamiento para su ejecución. Esto hace posible que se puedan generar grandes y variadas baterías de juegos sobre los que lanzar los diferentes agentes.

Gracias a todas estas ventajas, en la actualidad existen varios frameworks de desarrollo basados en *Atari 2600* disponibles para que cualquier interesado en el *General Game Playing* pueda crear un agente sin tener que preocuparse de emular y modelar los juegos. Entre ellos, podemos encontrar el *Atari 2600 Learning Enviroment* (A.L.E) [Bellemare y otros, 2013] y el *GVGAI Framework*. Este último será estudiado en detalle y utilizado para la implementación propuesta en este proyecto.

Algunos de estos frameworks han acabado por inspirar algunas competiciones en las que los participantes se enfrentan por conseguir el agente con mejores resultados en una serie de juegos, como es el caso de la *General Video Game Playing AI Competition* (GVGAI).

Cabe destacar la importancia de los juegos de *Atari 2600* haciendo mención al interés mostrado por Google al desarrollar su programa *DeepMind*, del cual hablaremos más adelante, sobre este marco.

2.3 GAME DESCRIPTION LANGUAGE

Una de las características principales del *General Game Playing* es el desconocimiento de las reglas del juego antes de que comiencen por parte de los agentes o bots. Las reglas se comunican en tiempo de ejecución, y los agentes o bots deben ser capaces de interpretarlas correctamente.

Para ello se crea el *Game Description Language* (GDL) [Ebner y otros, 2013], un lenguaje diseñado por Michael Genesereth como parte de del proyecto de General Game Playing de la universidad de Stanford. Utilizando este lenguaje se pueden representar todos los juegos como conjuntos reglas.

El GDL es un lenguaje de programación lógico, similar a *Prolog*, aunque tienen importantes diferencias como:

1. La semántica de GDL es puramente declarativa
2. GDL tiene restricciones para asegurar que todas las cuestiones son decidibles
3. Existen palabras reservadas para orientar el lenguaje a la definición de juegos

2.3.1 ELEMENTOS DEL GDL

Utilizando GDL se conceptualizan los juegos en base a entidades, acciones, proposiciones y jugadores:

- **Entidades:** Las entidades incluyen todos los objetos relevantes para el estado del juego: personajes, filas, columnas, casillas del tablero entre otros.
  Por ejemplo, en el juego del ajedrez, se puede definir el rey blanco como wk, una fila del tablero con una letra, por ejemplo e, y la columna con un número como 5. También se pueden combinar para crear entidades como una casilla(e,5) , representando la casilla en la fila e, columna 5.

- **Acciones:** Las acciones las realizan los jugadores. Se pueden definir de manera primitiva o compuesta, de la manera que las entidades.
  Por ejemplo, se puede definir una acción primitiva noop (no realizar ninguna acción) o una compuesta move(a1,a2) para mover una pieza de la casilla a1 a la a2.

- **Proposiciones:** Son condiciones que se evalúan verdadero o falso en cada estado del juego. Las proposiciones se definen usando constantes de objetos o términos compuestos.
  Por ejemplo, se podría definir el término compuesto on(wk,a1) para definir la proposición en que el rey blanco está en la casilla a1.
  El valor (verdadero/falso) de las proposiciones puede cambiar de un estado a otro.

- **Jugadores:** Los jugadores son las entidades activas del juego. En cada paso del juego, un jugador tiene un conjunto de acciones legales y ejecuta una de estas acciones. Los jugadores se definen con constantes, y rara vez se utilizan términos compuestos.

En GDL, existen términos cuyo significado puede variar en función del juego (*game-specific vocabulary*), y términos reservados cuyo significado es el mismo para todos los juegos (*game-independent vocabulary*).

Dentro de los términos reservados, se encuentran las representaciones numéricas de los 100 primeros enteros (0...100) y además, 10 constantes reservadas independientes (*game-independent vocabulary*):

- ***role(a)* :** *a* es un rol del juego.
- ***base(p)*:** *p* es una proposición base del juego.
- ***input(r,a)* :** *a* es una posible acción para el rol *r.*
- ***init(p)*** : la proposición *p* es verdadera en el estado inicial.
- ***true(p)* :** la proposición *p* es verdadera en el estado actual del juego.
- ***does(r,a)* :** el jugador *r* realiza la acción *a* en el estado actual.
- ***next(p)* :** la proposición *p* es verdadera en el estado siguiente.
- ***legal(r,a)* :** la acción *a* es una acción legal para el rol *r* en el estado actual.
- ***goal(r,n)* :** el estado actual tiene un valor de utilidad *n* para el rol *r*.
- ***terminal* :** el estado actual es un estado terminal

2.4 VIDEO GAME DESCRIPTION LANGUAGE

El *Video Game Description Language* (VGDL) es un lenguaje de alto nivel diseñado por Tom Schaul [Schaul, 2013] implementado originalmente en Python utilizando el framework py-game.

VGDL permite definir juegos de manera concisa con tan solo unas pocas líneas de código plano. De esta manera se pueden generar grandes portfolios de juegos que permitan la evaluación de algoritmos orientados al General Game Playing.

Utilizando VGDL, la definición de un juego se realiza con dos componentes separados:

- La **descripción del nivel**, que describe las posiciones de los objetos del nivel, mediante el mapeo de caracteres-objetos en una cuadrícula de texto plano. Podemos observar un ejemplo en la Figura 2.1
- La **descripción del juego**, (cuyo ejemplo podemos observar en la Figura 2.2), que describe las dinámicas y las interacciones entre los objetos del juego, dividida en 4 secciones:
    - **SpriteSet:** Define el conjunto de sprites del juego, incluyendo sus parámetros de visualización. Se pueden organizar herencias usando una estructura de árbol.
    - **LevelMapping:** Define la relación entre los caracteres usados en la descripción de los niveles y los sprites disponibles
    - **InteractionSet**: Especifica los eventos que ocurren cuando colisionan dos sprites en el juego
    - **TerminationSet**: Especifica las condiciones de terminación, definiendo si el jugador gana o no.

```
wwwwwwwwwwwww
wA          w   w
w   w           w
w     w     w  +ww
www  w1    wwwww
w           w  G  w
w  1              ww
w        1        ww
wwwwwwwwwwwww
```

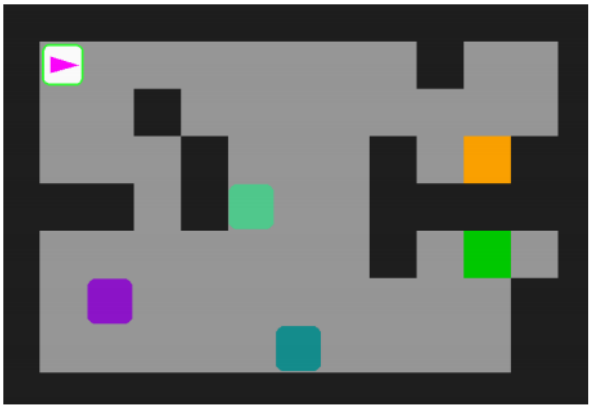


Figura 2.1.Ejemplo de definición de nivel representando un juego estilo 'Zelda'. El avatar (A) tiene que obtener la llave (+) y llegar a la salida (G) evitando los enemigos (1)

```
BasicGame frame_rate=30
    SpriteSet
        hole   > Immovable color=DARKBLUE img=hole
        avatar > MovingAvatar #cooldown=4
        box    > Passive img=box
    LevelMapping
        0 > hole
        1 > box
    InteractionSet
        avatar wall > stepBack
        box avatar  > bounceForward
        box wall    > undoAll
        box box     > undoAll
        box hole    > killSprite scoreChange=1
    TerminationSet
        SpriteCounter stype=box    limit=0 win=True
```

Figura 2.2. Ejemplo de definición de juego correspondiente al juego Sokoban, en el que el avatar tiene que empujar una serie de cajas para colocarlas en los agujeros

Lo que permite unas descripciones de juego tan breves y concisas es la ontología subyacente que definen conceptos de alto nivel como los tipos de físicas, físicas de movimiento, o físicas de colisión entre objetos. Estos efectos y comportamientos se traducen en unas pocas líneas de código en Python, por lo que se pueden combinar y extender con relativa facilidad.

Algunos de los elementos que podemos encontrar en la ontología actual son:

- Creación, clonación, y transformación de objetos
- Comportamientos de persecución o huidas no deterministas
- Comportamientos de proyectiles con trayectorias programadas
- Físicas continuas como inercia, gravedad o fricción

2.5 General Video Game AI Competition

La *General Video Game AI Competition* (GVGAI Competition) es una competición creada por el *Stanford Logic Group* [Genesereth y otros, 2005], en la que los participantes compiten por lograr el controlador que mejor puntuación consiga jugando a una serie de juegos desconocidos hasta el momento de la competición.

2.5.1 Objetivo

El objetivo de la GVGAI es explorar el problema de crear controladores (o agentes) para *General Video Game Playing*, de manera que puedan jugar a cualquier juego. Se trata, en definitiva, de programar un agente que pueda jugar a una amplia variedad de juegos sin conocer previamente de que juegos se trata.

2.5.2 Funcionamiento

Para participar, se debe entregar un controlador escrito en Java que pueda jugar a cualquiera de los juegos que soporta el VGDL Framework. Para ello, es necesario descargar el *starter kit* y seguir las instrucciones.

El *starter kit* contiene todo el código necesario del VGDL Framework para el desarrollo y ejecución de controladores, así como conjunto de 30 juegos diferentes sobre las que probar el controlador.

Además, previamente a la competición, se permite evaluar el controlador en el servidor de la competición, pudiendo elegir algunos de los sets de entrenamiento o de validación, cuyos resultados estarán disponibles en el perfil del competidor.

Todas estas evaluaciones estarán disponibles en el servidor y podrán consultarse en un ranking provisional, de manera que el competidor pueda comparar sus resultados con los de otros controladores.

Finalmente, llegada la fecha, se evaluarán los controladores con un conjunto secreto de juegos, y se publicarán los resultados. Una vez publicados, se podrán descargar los controladores participantes.

2.5.3 Especificaciones generales

Existen una serie de especificaciones generales que deben cumplir todos los participantes de la competición, que regulan algunos aspectos de la programación de los controladores, así como de los tiempos disponibles para su ejecución.

2.5.3.1 ESPECIFICACIONES DE PROGRAMACIÓN

- La participación en la competición implica que el código fuente del controlador estará disponible públicamente para su descarga una vez termine la competición
- Los controladores estarán escritos en Java. La clase principal debe heredar de core.player.AbstractPlayer.java y debe llamarse Agent.java . El paquete debe tener el mismo nombre de usuario registrado en la web.
- Java-VGDL utiliza la versión 1.7 de JDK
- El envío del controlador se realizará en un archivo zip, que podrá incluir todos los archivos necesarios para la ejecución del agente
- No se admiten técnicas de multi-threading
- Se permite la lectura de ficheros. La escritura solo se permite en la misma carpeta del controlador
- Los controladores dispondrán de la información del estado del juego, pudiendo evaluar los estados siguientes. La modificación del estado incurrirá en la descalificación del controlador y del usuario de la competición

2.5.3.2 ESPECIFICACIONES DE TIEMPO:

- El constructor del agente debe finalizar antes de 1 segundo desde su llamada, o será descalificado para el nivel en cuestión
- El controlador dispone de 40 milisegundos para devolver una acción desde que se llama a el método act(…)
- Si se supera ese tiempo, se pueden dar dos situaciones:
    - Si la función tardó entre 40 y 50 milisegundos, se aplicará la acción ACTIONS.ACTION_NIL (no hacer nada)
    - Si se tarda más de 50 milisegundos, el controlador será descalificado del juego, obteniendo -1000 puntos
- Se debe tener como objetivo alcanzar los 40 milisegundos de tiempo, ya que el intervalo 40-50 solo está para prevenir descalificaciones para casos puntuales. En ningún caso se debe intentar aprovechar ese tiempo de computación

2.5.4 RESULTADOS

La última edición de la competición se corresponde con la realizada en la GECCO (Generic and Evolutionary Computation Conference) de 2015, cuyos resultados y mejores participantes podemos observar en la Figura 2.3

Rank	Username	Country	Description	G-1	G-2	G-3	G-4	G-5	G-6	G-7	G-8	G-9	G-10	Total
1	YOLOBOT [3867]	Germany	Description	**25**	**25**	0	**25**	8	0	0	18	15	**25**	**141**
2	thorbjrn [3723]	Denmark	Description	0	15	**25**	18	0	10	12	8	**25**	2	**115**
3	psuko [3744]	Germany	Description	12	0	1	6	**25**	0	**25**	0	12	8	**89**
4	Return42 [3980]	Germany	Description	15	0	18	0	12	0	10	10	0	10	**75**
5	tragon [3837]	Germany	Description	10	0	15	0	15	0	18	0	1	4	**63**
6	jaydee [3842]	Germany	Description	1	10	10	0	0	0	4	12	10	15	**62**
7	TeamTopbug [3803]	Germany	Description	0	18	0	1	0	0	1	15	8	18	**61**
8	MaastCTS [3995]	Germany	Description	18	8	0	0	0	18	6	0	0	0	**50**
9	NovTea [3971]	Argentina	Description	0	0	0	8	6	0	0	**25**	0	6	**45**
10	TUDarmstadtTeam2 [3865]	Germany	Description	8	0	0	2	0	**25**	0	6	4	0	**45**

Figura 2.3. Ranking de participantes y resultados obtenidos en la última competición realizada por la GVGAI Competition en la GECCO 2015

Podemos observar las estrategias de los participantes más significativos (+100 puntos) leyendo su descripción:

YOLOBOT:

Para juegos deterministas, utiliza una búsqueda en anchura. Para los no deterministas, utiliza Montecarlo Tree Search (MCTS). Analizando el estado actual del juego, se crea una lista de tipos de *sprite* alcanzables por los sprites más cercanos. Utilizando esta información y el estado actual, se determina el objetivo más interesante. Se utiliza MCTS para acercarse al objetivo evitando perder el juego.

Thorbjrn:

Para juegos deterministas, se utiliza búsqueda en anchura. Para el resto, se aplica una versión simplificada del Montecarlo Tree Search (MCTS) en la que solo se analizan los nodos adyacentes (alcanzables en un solo movimiento), expandiendo esos nodos con acciones aleatorias.

2.5.5 GVGAI Framework

La GVGAI Competition ofrece un Framework sobre el que los participantes deberán programar sus controladores. El contenido de este framework se divide en los siguientes paquetes:

- ***controllers:*** Contiene los paquetes de ejemplo distribuidos con el *starter kit*. Cada paquete contiene un controlador, cuyo archivo principal de la implementación es Agent.java. También contiene controladores para jugar como humano

- ***core:*** Contiene el código principal del framework VGDL, el cual se divide en los siguientes subpaquetes:
    1. ***competition:*** Contiene algunos de los parámetros para ejecutar en la competición
    2. ***content:*** Contiene las clases encargadas de crear sprites para los juegos
    3. ***game:*** Contiene clases que permiten jugar a los juegos, así como la observación de estados (actual y futuros)
    4. ***player:*** Contiene la clase que todos los controladores deben heredar : AbstractPlayer.java
    5. ***termination:*** Clases para la terminación de los juegos, de acuerdo a las definiciones del VGDL

Además, el paquete core contiene otras clases encargadas de traducir los juegos VGDL (VGDLParser.java, VGDLFactory.java) , así como las clases base para todos los sprites y la clase encargada de los gráficos.

- ***ontology:*** Contiene paquetes con las definiciones para todos los sprites, tipos de avatar, tipos de físicas, y efectos de colisiones.
- ***tools:*** Contiene algunas herramientas de utilidad como temporizadores, funcionalidades de entrada/salidas, manejadores de teclado y herramientas estadísticas
- ***Test.java:*** Es el fichero principal para ejecutar el framework. Utiliza algunos métodos de la clase ArcadeMachine.java y permite ejecutar el framework de diferentes maneras:
    1. Jugar a un juego/nivel como humano (ArcadeMachine.playOneGame(...))
    2. Jugar a un juego/nivel con un controlador (ArcadeMachine.runOneGame(...))
    3. Re jugar un juego/nivel desde un fichero obtenido en una jugada anterior (ArcadeMachine.replayGame(...))
    4. Jugar a un juego con N niveles, M veces (ArcadeMachine.runGames(...)) . Con este modo podemos obtener estadísticas de las partidas jugadas

Tal como ha visto en la sección anterior, para crear un controlador, solo tenemos que implementar un Agent.java que herede AbstractPlayer.java dentro de su paquete correspondiente y que implemente el método act(…)

Para probar los resultados, tan solo será necesario modificar la clase Test.java para que ejecute el controlador bajo los parámetros de juego deseados.

2.6 GOOGLE DEEPMIND

Como muestra del creciente interés por el *General Game Playing*, recientemente ha sido noticia el lanzamiento del proyecto *DeepMind* de Google [DeepMind], que consiste precisamente en un agente capaz de jugar de una manera óptima a una gran variedad de juegos clásicos de la consola *Atari 2600*. La demostración y el buen nivel de los resultados obtenidos ha causado un gran impacto en los medios especializados, ya que el algoritmo no se limita a aprender las reglas del juego, sino que descubre estrategias óptimas de juego para maximizar su puntuación, de la misma forma que lo haría un jugador humano. Por ejemplo, jugando al juego *Arkanoid*, *DeepMind* acabó descubriendo que la mejor estrategia consistía en crear un túnel para colar la bola por encima de los bloques.

La base del funcionamiento de este algoritmo es la combinación de redes neuronales (*Deep Q Network*) [Mnih y otros, 2015] con aprendizaje por refuerzo. La red neuronal consiste en 3 capas ocultas que reciben como entrada la información de la pantalla del juego, así como la puntuación. Las neuronas de salida se corresponden con las diferentes acciones disponibles del juego, y la red se va entrenando a medida que se juegan partidas.

En la Figura 2.4 se puede visualizar el esquema de funcionamiento de la red neuronal:

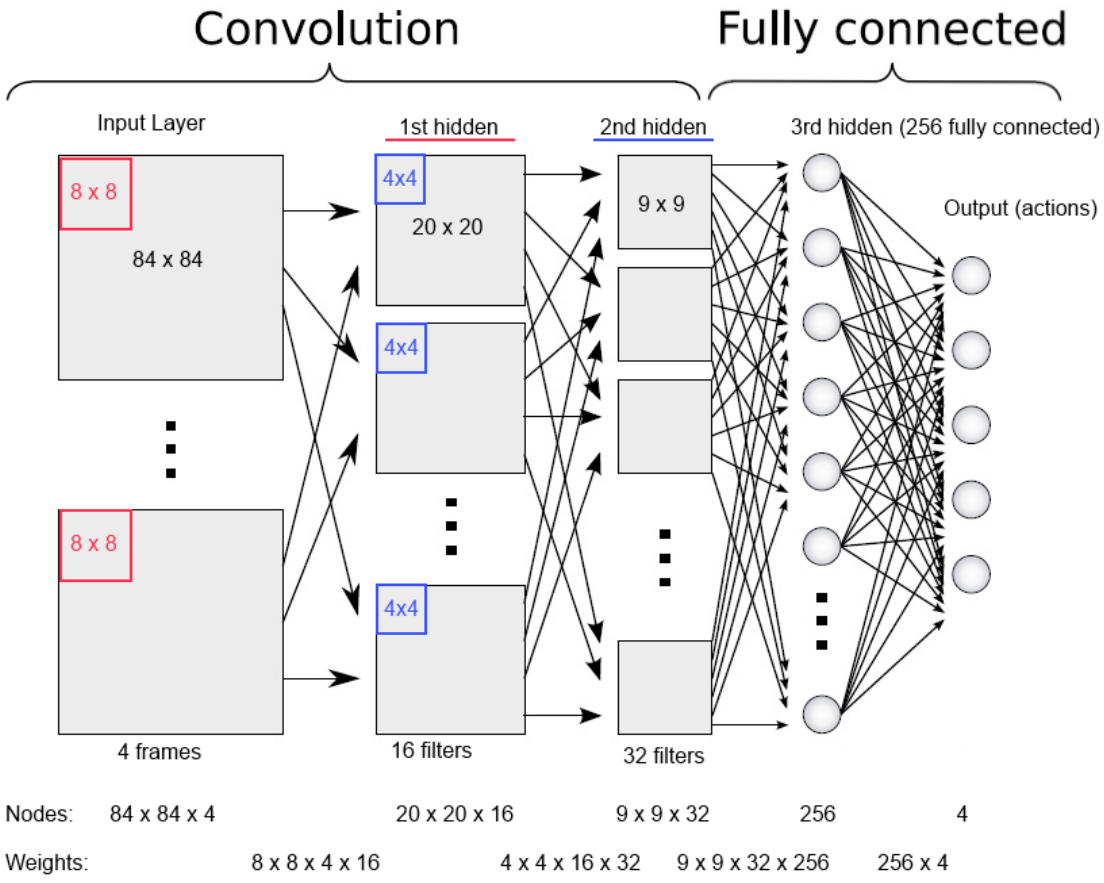


Figura 2.4. Esquema de funcionamiento de la red neuronal Deep Q Network de DeepMind

A pesar de tratarse de un sistema no excesivamente complejo, los resultados han sido asombrosos, demostrando que no siempre es necesario un gran sistema cargado de algoritmos complejos para resolver algunos de estos problemas.

Se trata sin duda de un gran avance que acerca aún más la posibilidad de crear una IA que se comporte de una manera similar que la mente humana.

2.7 ALGORITMOS

A continuación se explicarán las características generales de los principales algoritmos presentes en este proyecto.

2.7.1 ALGORITMO GENÉTICO

Los algoritmos genéticos se inspiran en la evolución biológica de los genes para aplicarlos a la inteligencia artificial [Gen y Cheng, 1996]. El proceso de evolución se realiza de forma iterativa repitiéndolo hasta que se cumpla una condición de parada. Este proceso se divide principalmente en 3 etapas [Goldberg, 1989]:

- Selección
- Cruce
- Mutación

Fuera del cuerpo del bucle principal, como primer paso, es necesaria la inicialización de la población, que se suele hacer de forma aleatoria, comprobando opcionalmente que las soluciones sean viables para el problema.

Para ello, será necesario introducir los conceptos de **población** e **individuo**.

- **Población:** Básicamente es el conjunto de individuos sobre los que se aplica el algoritmo. La población cambia en cada iteración hasta que finalmente (con tiempo suficiente) converge a una población inmutable.
- **Individuo:** Cada individuo de la población constituye una solución para el problema. Los individuos se construyen en base a una serie de **genes**, los cuales, en función del problema, pueden representar una codificación binaria (Figura 2.5), variables numéricas o, como en este caso, secuencias de acciones.

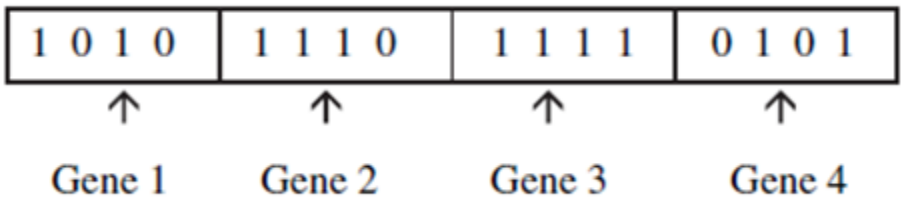


Figura 2.5: Representación de los genes de un individuo en codificación binaria

2.7.1.1 FUNCIONAMIENTO

En la figura 2.6, podemos observar un diagrama general de funcionamiento del algoritmo, antes de entrar en detalle sobre cada una de las fases.

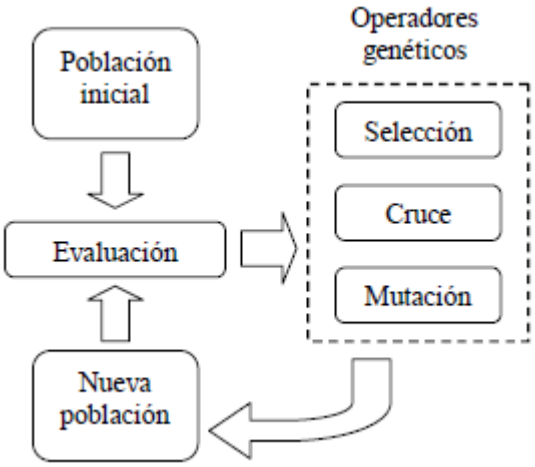


Figura 2.6: Diagrama general del funcionamiento de un algoritmo genético

2.7.1.2 SELECCIÓN

Basándose en la teoría de la evolución de Darwin, los individuos más capacitados de la población serán los que lleven a engendrar nuevos individuos que mejorarán la especie. Esta misma teoría se aplica a los algoritmos genéticos, donde se pueden aplicar diferentes técnicas de selección, adaptándolos a la casuística de cada problema.

Posibles técnicas de selección:

- Selección por ruleta
- Selección elitista
- Selección por torneo
- Selección por estado estacionario

Para la evaluación de un individuo, es necesario implementar una función de evaluación, también conocida como *Fitness*, la cual puntúa a cada individuo según lo idónea que resulte para el problema la solución que representa.

2.7.1.3 CRUCE

Una vez seleccionados mediante alguna de las técnicas los dos individuos “padres”, se procede a combinar sus genes para generar un nuevo individuo “hijo”. También es posible realizar la clonación de uno de los padres sustituir a alguno de los miembros, consiguiendo así perpetuar los genes dominantes, aunque no siempre resulta una buena opción.

Existen diferentes técnicas de cruce:

- Cruce de un punto
- Cruce de dos puntos
- Cruce uniforme
- Cruce aritmético

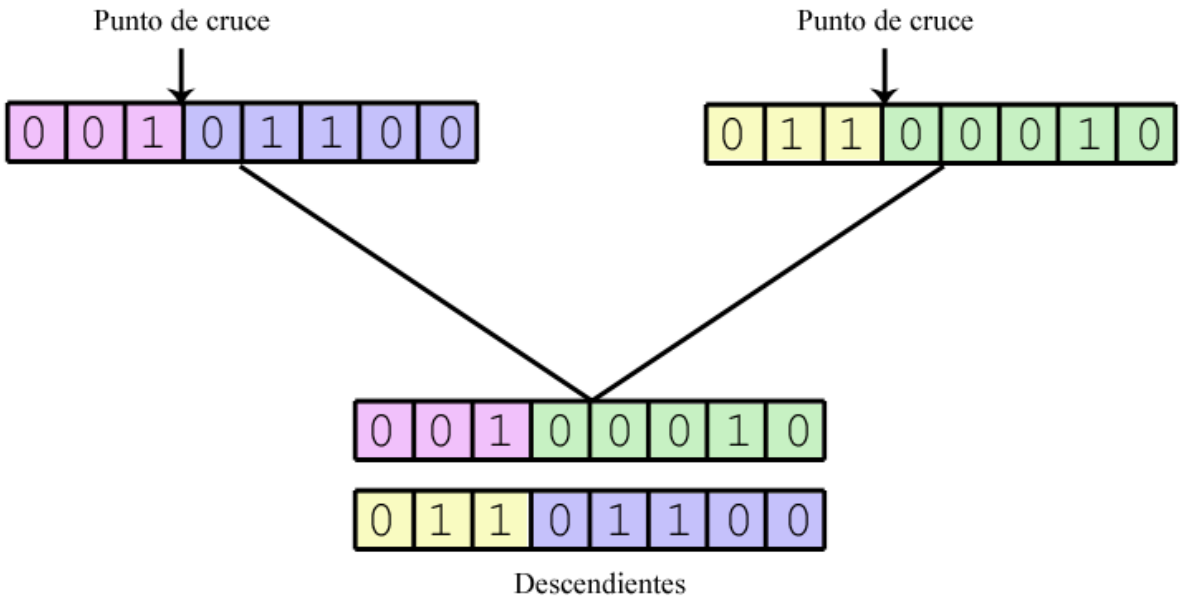


Figura 2.7: Ejemplo de cruce de un punto

2.7.1.4 MUTACIÓN

Basándonos nuevamente en la evolución natural de una especie, existe una posibilidad de que un nuevo individuo de la población sufra una mutación en sus genes. Generalmente, la mutación consiste en, bajo cierta probabilidad, cambiar uno o varios genes por un valor aleatorio. Esto contribuye a aumentar la diversidad de los individuos de la población

2.7.1.6 ALGORITMOS GENÉTICOS EN GENERAL GAME PLAYING

En el caso del General Game Playing, las principales características de un algoritmo genético básico es el siguiente:

- **Representación de la población:** Cada individuo representa una serie de acciones (genes) partiendo del estado actual.
- **Función de *Fitness*:** La puntuación de *Fitness* de cada individuo corresponde al resultado de aplicar secuencialmente todas sus acciones.
- **Condición de terminación:** El algoritmo se ejecuta mientras disponga de tiempo suficiente y, al finalizar, devuelve la primera acción del individuo de mayor puntuación.

2.7.2 MONTECARLO TREE SEARCH (MCTS)

El algoritmo MCTS [MCTS Wiki] es un algoritmo de búsqueda en árbol que ha tenido un gran impacto en el *General Game Playing* (GGP) y hoy en día, casi todos los participantes en la GVGAI Competition utilizan alguna variante de este algoritmo.

MCTS estima el valor de los nodos intermedios utilizando un muestreo aleatorio para recopilar información, y construye el árbol de manera incremental basándose en las recompensas estimadas.

El algoritmo comienza con un árbol formado únicamente por la raíz, y va expendiendo el árbol de manera asimétrica, centrándose principalmente en las ramas más prometedoras.

MCTS procesa el árbol en 4 fases (Figura 2.6), que se repiten hasta que se agota el tempo disponible. Entonces selecciona la acción más prometedora en función de la información recopilada:

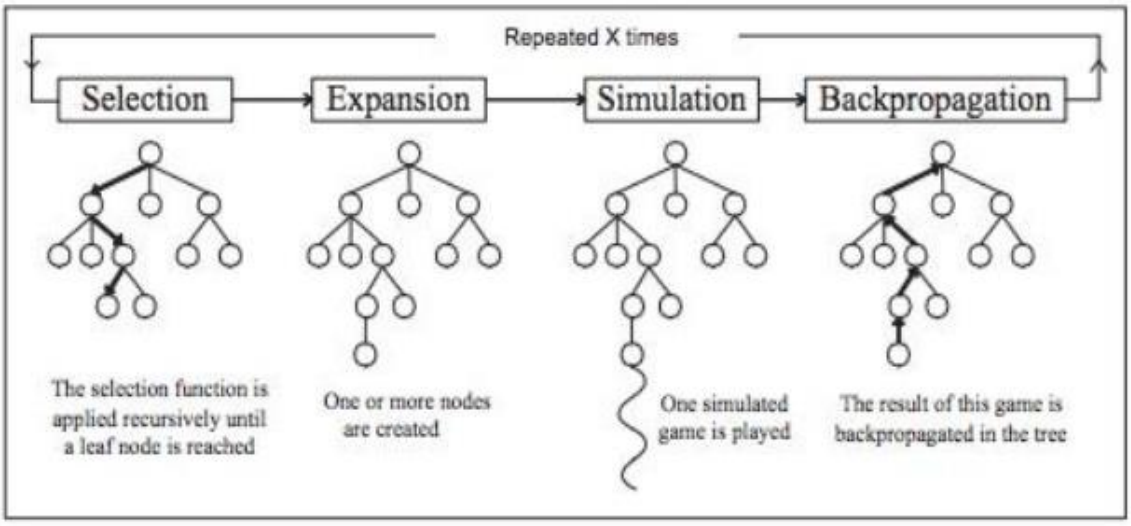


Figura 2.8: Esquema de funcionamiento general de Montecarlo Tree Search

2.7.2.1 SELECCIÓN

El agente selecciona un nodo dando prioridad a los nodos inexplorados. Si todos los hijos del nodo ya han sido visitados, se selecciona el que mayor utilidad estimada tenga hasta el momento

La fórmula típica para calcular la utilidad es *Upper Confidence Bounds* (UCB), buscando un equilibrio entre Exploración y Explotación:

$$v_i + C \times \sqrt{\frac{\ln N}{n_i}}$$

vi: Utilidad estimada del nodo

c: Constante para balancear entre exploración y explotación

ni: Número de visitas del nodo padre

N: Número de visitas del nodo actual

En la actualidad, se ha extendido la fórmula UCB para aplicarla a búsqueda en árboles, creando la fórmula *Upper Confidence Bounds for Trees* (UCT) [Browne y otros, 2012].

2.7.2.2 EXPANSIÓN

En esta fase se selecciona y se expande uno de los nodos hijos no expandidos hasta el momento.

2.7.2.3 SIMULACIÓN

En esta fase se simula el juego desde el nodo seleccionado, tomando acciones aleatorias del mismo modo hasta llegar a un nodo terminal.

2.7.2.4 PROPAGACIÓN

En esta fase, partiendo del nodo seleccionado, se actualiza su utilidad con el resultado de la simulación, y se propaga sucesivamente a sus antecesores.

2.7.2.5 VENTAJAS E INCONVENIENTES DE MCTS

Lista de las principales ventajas e inconvenientes del *Montecarlo Tree Search*:

- **Principales ventajas:**
  - **Independiente del dominio:**

    MCTS no necesita heurísticas ni conocimiento del dominio del problema para tomar decisiones razonables. Eso provoca que una misma implementación de MCTS se pueda utilizar en numerosos juegos sin ninguna modificación.

  - **Asimétrico**

    MCTS construye el árbol de forma asimétrica, ya que visitando los nodos más prometedores, se construye el árbol centrado en las ramas más relevantes.

  - **Detención**

    MCTS puede detenerse en cualquier momento y producir una decisión razonable.

- **Principales inconvenientes:**
  - **Optimista**

    Pensar que todos los jugadores juegan de una manera aleatoria puede ser optimista. Por ejemplo, si estamos jugando al ajedrez, puede no resultar buena idea escoger una acción cuya mayoría de sondas llegan a un estado favorable, si una de ellas nos conduce a un jaque mate del oponente.

  - **No tiene en cuenta la estructura del juego**

    Al no tener en cuenta la estructura, puede pasar por alto simetrías en un tablero o situaciones idénticas que podrían reducir drásticamente el coste de búsqueda.

3 DISEÑO E IMPLEMENTACIÓN

En esta sección se listarán y detallarán las características de los algoritmos implementados en este proyecto

3.1 ALGORITMOS IMPLEMENTADOS

Existen infinidad de posibilidades de algoritmos aplicables a la hora de explorar y elegir la siguiente acción de juego. La gran mayoría (aunque no necesariamente) se basan en diferentes tipos de búsquedas en árbol, así como redes neuronales o algoritmos evolutivos.

En la actualidad, los algoritmos que mejores resultados han obtenido son el *Montecarlo Tree Search* (MCTS) como mejora el *Montecarlo Search* (MCS) y el *Deep Q Network*, recientemente aplicado por Google para su proyecto *Deepmind*.

Dentro de este proyecto se analizarán y compararán los resultados de los siguientes algoritmos:

- *Random*
- *One Step Ahead*
- *Montecarlo Tree Search (*Variante UCT)
- *Montecarlo Tree Search (*Variante Greedy)
- Algoritmo Genético (Variante de profundidad 7)
- Algoritmo Genético (Variante de profundidad 10)
- Algoritmos híbridos de algoritmo genético y MCTS

3.1.1 RANDOM

Se trata de uno de los algoritmos más simples. Consiste simplemente en seleccionar una acción aleatoria de entre las acciones disponibles.

3.1.2 ONE STEP AHEAD

Otro algoritmo de gran simplicidad. Su funcionamiento consiste en observar el estado siguiente a la aplicación de cada una de las acciones disponibles y elegir la que mejor resultado produzca. Este algoritmo se puede modificar para que mire los N siguientes pasos (*N Step Ahead*).

3.1.3 MONTECARLO TREE SEARCH (MCTS)

Para aplicar el algoritmo MCTS al problema que se trata en este proyecto, es necesario tener en cuenta las siguientes consideraciones:

- Se construye el árbol en el que cada nodo representa un estado del juego.
- Cada nodo del árbol, tendrá como hijos los estados resultantes de aplicar cada una de las acciones disponibles.
- Debido a la naturaleza estocástica de los juegos y del desconocimiento de la profundidad del árbol al completo, así como la limitación de tiempo para cada jugada, tan solo se simula el juego hasta una profundidad determinada.
- La puntuación de un nodo se simula de la siguiente forma:
    - Si el nodo es terminal, se devuelve un gran número positivo en caso de que el jugador gane, y gran número negativo en caso de que pierda.
    - Para los nodos intermedios, se devuelve la puntuación del juego.
- Para cada paso, se sitúa la raíz del árbol en el estado actual del juego, y se devuelve la mejor acción resultante de la ejecución del algoritmo para ese estado.

Para este proyecto, se han implementado dos variantes diferentes del algoritmo MCTS, que afectan especialmente a la política de selección

3.1.3.1 MONTECARLO TREE SEARCH – GREEDY

En esta variante, se utiliza una selección voraz (*Greedy*), en la que el valor un nodo es simplemente su puntuación acumulada

3.1.3.2 MONTECARLO TREE SEARCH – UCT

En esta variante de MCTS, se ha modificado la función de selección, así como los métodos de propagación para aplicar la formula *Upper Confidence Bounds for Trees* (UCT) como valor estimado de un nodo:

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln t}{n_i}}$$

w_i: *Puntuación acumulada del nodo*

n_i: *Número de visitas totales del nodo*

c: *Parámetro para equilibrar exploración y explotación*

t: *Número de visitas del nodo padre*

3.1.4 ALGORITMOS GENÉTICOS

Los algoritmos genéticos que se han implementado tienen las siguientes características:

- **Individuos:** Cada individuo está formado por una serie de genes, que se corresponden con una secuencia de acciones hasta una profundidad determinada.
- **Población:** La población se genera de la siguiente manera: Partiendo de un estado inicial, se genera un número determinado de individuos para cada una de las acciones disponibles.
- **Inicialización:** Los individuos se generan de manera aleatoria, exceptuando la primera de las acciones, que se corresponde con la acción principal a la que pertenecen.
- **Fitness:** Se utiliza una función de evaluación (*Fitness*) normalizado. Consiste en la puntuación obtenida tras simular la secuencia de acciones del individuo, ponderada respecto a la profundidad y normalizada respecto a las puntuaciones de los otros individuos.
- **Selección:** La selección de individuos "padre" se realiza mediante *torneo binario*. Consiste en seleccionar al azar dos individuos y comparar sus puntuaciones. El perdedor será reemplazado por el nuevo individuo
- **Cruce:** La generación del nuevo individuo se realiza mediante *cruce uniforme.* Cada uno de sus genes pertenecerá a uno de los padres según una probabilidad aleatoria
- **Mutación:** Existe una probabilidad de que el nuevo individuo sufra alguna mutación en sus genes, generando una acción aleatoria en su lugar.
- **Bucle principal:** El algoritmo se ejecuta un número determinado de iteraciones siempre que tenga suficiente tiempo disponible. Para cada iteración, se realiza el proceso completo para la población correspondiente a cada una de las acciones.
- **Retorno:** Al terminar el bucle principal, devuelve la acción que haya obtenido la máxima puntuación en alguno de sus individuos.

3.1.4.1 VARIANTES DEL ALGORITMO GENÉTICO

En este proyecto, se han implementado dos variantes del algoritmo genético, variando la profundidad de exploración (número de acciones) de cada individuo, es decir, la longitud de su genoma.

La principal diferencia radica en que a mayor profundidad, mayor capacidad de predicción del resultado, pero sin embargo, aumenta el tiempo necesario para la simulación, realizando menos iteraciones.

En concreto, se han implementado variantes para profundidad 7(**AG_D7**) y 10(**AG_D10**).

3.1.5 ALGORITMOS HÍBRIDOS

Una de los objetivos principales de este proyecto, es encontrar algún algoritmo híbrido que combine el algoritmo MCTS y los algoritmos genéticos [Benbassat y Sipper, 2013].

Para ello, se han realizado modificaciones sobre ambos algoritmos de manera que evalúen de igual manera las posibles acciones obtenidas tras su ejecución.

Las propuestas de hibridación son las siguientes:

- Método colaborativo (**MCTS_AG_SEQ**)
- Método competitivo (**MCTS_AG_PAR**)

3.1.5.1 MÉTODO COLABORATIVO

En esta propuesta, se ejecutan los algoritmos de manera colaborativa (o secuencial). Para cada paso de juego, se ejecuta alternativamente uno de los dos algoritmos, utilizando como base el resultado del algoritmo anterior.

3.1.5.2 MÉTODO COMPETITIVO

En el algoritmo híbrido competitivo (o paralelo), en cada paso de juego se ejecutan ambos algoritmos (con la mitad de tiempo disponible para cada uno). Sus resultados se evalúan y se escoge el resultado que produzca mayor puntuación

4 EXPERIMENTACIÓN Y RESULTADOS

En esta sección se mostrarán y analizarán los resultados obtenidos en los experimentos por los diferentes algoritmos implementados

4.1 MARCO EXPERIMENTAL

Para poder analizar y comparar la eficacia de los diferentes algoritmos, se realizarán pruebas ejecutándolos sobre diferentes juegos. Debido a los componentes estocásticos de estos juegos, los algoritmos se ejecutarán 10 veces por cada uno de los juegos, de manera que los resultados obtenidos sean más robustos.

Los juegos seleccionados para la experimentación son los correspondientes al *Validation Set* de la competición del CIG15 (*IEEE Conference on **C**omputational **I**ntelligence and **G**ames)* [IEEE CIG]. Los juegos correspondientes a esta lista son:

Camel Race	El avatar debe llegar a la meta antes que ningún otro camello.
Digdug	El avatar debe recoger las gemas y las monedas de oro en de la cueva, excavando en ella. También hay enemigos que mataran al jugador al tocarlo. El jugador puede lanzar rocas presionando USE dos veces consecutivas, las cuales eliminarán a los enemigos.
Firestorms	El avatar debe encontrar el camino de salida evitando las llamas del nivel, creadas por los portales infernales. El avatar puede recoger agua, la cual protegerá al jugador de un encuentro con una llama.
Infection	El avatar se infecta mediante el contacto con los insectos esparcidos por el nivel, o tocando a otros animales infectados. El objetivo es infectar todos los animales saludables. Los sprites azules son médicos que curan los animales infectados y al jugador, pero pueden ser eliminados con su espada
Firecaster	El avatar debe encontrar el camino a la salida quemando las cajas de madera. Para poder disparar, el avatar necesita recolectar munición (maná), esparcida por el nivel. Las llamas se expanden, pudiendo destruir más de una caja, pero también pueden dañar al avatar. Si la salud del avatar llega a 0, el jugador pierde

Overload	El avatar debe alcanzar la salida con un número determinado de monedas, pero si la cantidad recolectada es mayor de la requerida, el avatar queda atrapado en las enredaderas del pantano y el juego acaba. El avatar puede destruir las plantas con su espada, si la obtiene antes.
Pacman	El avatar debe limpiar el laberinto comiendo todas las bolas y pastillas de poder. Hay fantasmas que matarán al jugador si no ha comido un poder previamente. También hay frutas que deben recogerse.
Seaquest	El jugador controla un submarino que debe evitar ser destruido por animales, y rescatar submarinistas llevándolos a la superficie. Además, el submarino debe volver a la superficie con regularidad para recargar oxígeno, o el jugador perderá. El submarino tiene capacidad para 4 buceadores, y puede disparar torpedos a los animales.
Whackamole	El avatar debe recoger los topos que aparecen de los agujeros. También hay un gato que hace lo mismo en el nivel. Si el jugador choca con el gato, pierde.
Eggomania	Hay una gallina en la parte superior del nivel lanzando huevos. El avatar debe moverse hacia los lados para evitar que los huevos se rompan contra el suelo. Cuando el avatar recoja suficientes huevos, puede disparar a la gallina para ganar el juego. Si se rompe un solo huevo, el jugador pierde.

La descripción de los juegos se puede obtener en la web de la *GVGAI Competition* [GVGAI].

Además, existe una lista de reproducción con videos de cada juego:

https://www.youtube.com/watch?v=cERATvRagP4&list=PLe89c3ir1UJeAkWVKhvxeUS8IqV5Tk3JQ

4.2 PARAMETRIZACIÓN DE LOS EXPERIMENTOS

Con la finalidad de garantizar la replicabilidad de los experimentos, a continuación se listarán los parámetros usados para los algoritmos durante la experimentación:

4.2.1 ALGORITMOS MCTS

- Profundidad de exploración (*ROLLOUT_DEPTH*) : 10
- (Solo para UCT) Constante de balanceo (*K*): √2

4.2.2 ALGORITMOS GENÉTICOS

- Constante de ponderación de la profundidad (*GAMMA*): 0.90
- Tiempo mínimo disponible para permitir la ejecución (*Break_MS*) : 35
- Número de acciones de los individuos (*SIMULATION_DEPH*) : Según la variante, valdrá 7 o 10
- Tamaño de la población para cada acción (*POPULATION_SIZE*) : 5
- Probabilidad de sustituir un gen del individuo perdedor del torneo con el del ganador (*RECPROB*): 0.1
- Probabilidad de mutación (*MUT*): 1 / *SIMULATION_DEPH*

4.3 RESULTADOS DE LA EXPERIMENTACIÓN

A continuación, se muestra la tabla con las puntuaciones máximas obtenidas por cada algoritmo en cada uno de los juegos, además de una tabla que muestra el resultado promedio de las 10 ejecuciones, así como la desviación típica.

4.3.1 TABLAS DE PUNTUACIONES MÁXIMAS

	Random	One Step Ahead	MCTS UCT	MCTS GREEDY	AG D7	AG D10	MCTS_AG SEQ	MCTS_AG PAR
J1	-1.0	**1.0**	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
J2	3.0	4.0	32.0	20.0	11.0	3.0	**38.0**	35.0
J3	-2.0	0.0	0.0	-1.0	0.0	0.0	**1.0**	0.0
J4	42.0	128.0	69.0	**190.0**	84.0	37.0	92.0	111.0
J5	15.0	2.0	15.0	12.0	18.0	18.0	18.0	**20.0**
J6	11.0	1.0	14.0	15.0	**20.0**	19.0	19.0	**20.0**
J7	107.0	6.0	225.0	237.0	88.0	**292.0**	283.0	285.0
J8	15.0	0.0	1031.0	1038.0	2089.0	1017.0	1079.0	**2106.0**
J9	-5.0	-15.0	2.0	-5.0	13.0	4.0	14.0	**19.0**
J10	5.0	1.0	7.0	2.0	**10.0**	4.0	3.0	4.0

Tabla 4.1 Tabla de puntuaciones máximas de los algoritmos en cada uno de los juegos del Validation Set. Cada celda representa la puntuación máxima obtenida por cada algoritmo tras ejecutarlos 10 veces

4.3.2 TABLA DE PUNTUACIONES MEDIAS

	Random	One Step Ahead	MCTS UCT	MCTS GREEDY	AG D7	AG D10	MCTS AG SEQ	MCTS AG PAR
J1	-1±0	1±0	-1±0	-1±0	-1±0	-1±0	-1±0	-1±0
J2	0,3±1,34	4±0	16,6±6,93	12,6±4,27	1,7±3,5	0,2±1,03	23,7±6,13	**21,6±9,31**
J3	-2,1±0,32	**0±0**	-2,7±1,77	-2,4±1,43	**0±0**	**0±0**	-2±2,75	**0±0**
J4	7,2±20,18	10,6±49,34	26,3±30,32	36,7±55,47	34,6±30,8	7,7±27,41	41,1±31,36	**63,8±24,12**
J5	7,9±3,35	2±0	10,6±3,5	9,1±2,88	11,6±6,08	10,2±4,16	11,7±4,3	**12,5±4,55**
J6	7,6±2,32	1±0	11,9±1,6	10,8±2,1	11±3,68	15,4±2,12	14,3±2,67	**17,7±2,79**
J7	68,3±30,53	6±0	164,9±46,75	128,5±65,6	19,4±24,52	88,2±88,42	**234,9±48,01**	225,1±48,85
J8	5,2±4,8	0±0	135,6±315,34	237,5±418,23	634,1±997,59	105,4±320,32	643,7±534,87	**865,5±941,4**
J9	-7,2±1,93	-15±0	-1,1±4,15	-10,4±3,37	**3±6,85**	-1,4±3,84	-3,3±6,83	1,7±8,79
J10	0,9±1,66	1±0	2,7±2,45	1,7±0,67	**2,8±2,82**	1,8±1,23	1,5±1,08	1,4±1,35

Tabla 4.2 Tabla de puntuaciones medias de los algoritmos en los distintos juegos. Cada celda representa la puntuación media obtenida así como su desviación típica tras ejecutarlos 10 veces

4.3.2.1 ANÁLISIS DE LOS RESULTADOS

Analizando los resultados, podemos observar que, dependiendo del juego, los algoritmos obtienen valores bastante dispares, aunque puede apreciarse una tendencia en la que los algoritmos genéticos e híbridos obtienen las mejores puntuaciones. Siendo la versión híbrida competitiva la que parece obtener la puntuación máxima en un mayor número de juegos. Con la finalidad de poder comparar los algoritmos entre ellos de una forma más precisa, se ha realizado una tabla comparativa por ranking.

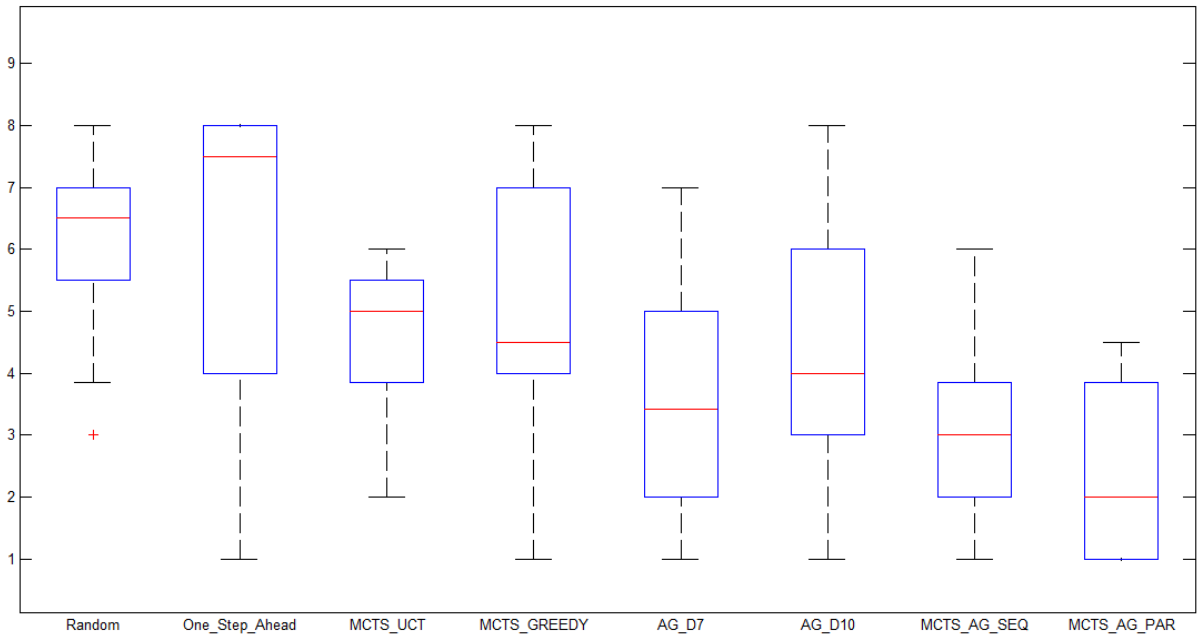
4.3.3 TABLA DE COMPARATIVA DE RANKINGS

Debido al amplio número de algoritmos y de problemas que se han considerado, resulta complicado comparar entre sí los distintos algoritmos observando únicamente sus resultados obtenidos. Para mejorar ese aspecto, se ha realizado una comparación basada en rankings [Hui y otros, 2014], asignando rankings r_{ij} a los resultados obtenidos por cada algoritmo j en cada problema i. Esto es, para cada problema, se asigna un ranking $1 \leq r_{ij} \leq k$, donde k es el número de algoritmos a comparar. Estos rankings se asignan de forma ascendente, es decir, 1 al mejor resultado, 2 al segundo, etc. (en caso de haber empates, se asignan rankings medios).

	Random	One Step Ahead	MCTS UCT	MCTS GREEDY	AG D7	AG D10	MCTS_AG SEQ	MCTS_AG PAR
J1	3.85	**1**	3.85	3.85	3.85	3.85	3.85	3.85
J2	7.5	6	3	4	5	7.5	**1**	2
J3	8	4	4	7	4	4	**1**	4
J4	7	2	6	**1**	5	8	4	3
J5	5.5	8	5.5	7	3	3	3	**1**
J6	7	8	6	5	**1.5**	2.5	2.5	**1.5**
J7	6	8	5	4	7	**1**	3	2
J8	7	8	5	4	2	6	3	**1**
J9	6	7	5	8	3	4	2	**1**
J10	3	8	2	7	1	4.5	6	4.5

Tabla 4.3 Tabla de rankings de algoritmo por juego. Cada celda representa la posición en el ranking de cada juego obtenida por cada algoritmo, en base a su puntuación máxima obtenida.

4.4 GRÁFICA DE DISTRIBUCIÓN DE ALGORITMOS POR RANKING



Gráfica 4.4 Diagrama de cajas representando la distribución de posiciones respecto al ranking de cada uno de los algoritmos. En este caso se intenta minimizar sus valores

4.4.1.1 ANÁLISIS DE LOS RESULTADOS

Observando la comparativa por ranking, tanto en la tabla como en la gráfica de distribución, se puede apreciar con mayor facilidad el dominio del algoritmo hibrido competitivo sobre los demás, obteniendo la primera posición en 4 de los 10 juegos, compartiendo una de esas posiciones (Juego 6) con el algoritmo genético de profundidad 7. Además, su distribución de valores, así como el valor de la mediana, se encuentran en las posiciones más bajas.

4.5 TEST ESTADÍSTICO DE LOS RESULTADOS

Antes de poder concluir una victoria del algoritmo hibrido competitivo (MCTS_AG_PAR) sobre los demás, resulta necesario realizar una serie de test estadísticos para verificar que las distribuciones de valores de los algoritmos son realmente independientes, de forma que validen las conclusiones obtenidas anteriormente.

Para ello, se realizará el test de Friedman [Derrac y otros, 2011], el cual, basándose en los rankings de los algoritmos, el cual establece una hipótesis nula que afirma que todos los algoritmos se comportan por igual. Entonces se calcula un valor-p (o *p-value*) asociado a la distribución.

En la tabla 4.5 podemos observar el resultado del test, que presenta el ranking de cada algoritmo, además de los *p-values* calculados por el test de Friedman y por el test de Iman y Davenport.

Algoritmo	Ranking
Random	6.25
OneStepAhead	6.1
MCTS_GREEDY	5.05
MCTS_UCT	4.65
AG_D10	4.65
AG_D7	3,65
MCTS_AG_SEQ	3.15
MCTS_AG_PAR	2.5

Tabla 4.5. Valores de ranking establecidos por el test de Friedman

Valor estadístico para el test de Friedman (distribuidos de acuerdo a una distribución chi-cuadrado con 7 grados de libertad: 20.858333333333405.

P-value calculado por el test de Friedman: 0.003985580003154743.

P-value calculado por el test de Imanand Daveport Test: 0.0016134689456791516.

A continuación, en la tabla 4.6, se muestran los resultados Post-Hoc del test de Friedman, utilizando como MCTS_AG_PAR método de control.

i	algoritmo	z = (R0-Ri)/SE	p-value	Holm/ Hochberg /Hommel	Holland	Rom	Finner	Li
7	Random	3	0,000618	0.007142	0.007300	0.007512	0.007300	0.023529
6	OneStepAhead	3,286335	0,001015	0.008333	0.008512	0.008764	0.014548	0.023529
5	MCTS-GREEDY	2,327820	0,019921	0.01	0.010206	0.010515	0.021742	0.023529
4	MCTS-UCT	1,962672	0,049684	0.0125	0.012741	0.013109	0.028885	0.023529
3	AG-D10	1,962672	0,049684	0.016666	0.016952	0.016666	0.035975	0.023529
2	AG-D7	1,049801	0,293809	0.025	0.025320	0.025	0.043013	0.023529
1	MCTS-AG-SEQ	0,593366	0,552936	0.05	0.050000	0.05	0.050000	0.05

Tabla 4.6. Resultados Post-Hoc del test de Friedman

Bonferroni-Dunn's rechaza las hipótesis con un p-value <0:0071428571428571435.

Holm's rechaza las hipótesis con un p-value <0:01.

Hochberg's rechaza las hipótesis con un p-value <0:008333333333333333.

Hommel's rechaza las hipótesis con un p-value <0:01.

Holland's rechaza las hipótesis con un p-value <0:010206218313011495.

Rom's rechaza las hipótesis con un p-value <0:008764162596519848.

Finner's rechaza las hipótesis con un p-value <0:028885068789519686.

Li's rechaza las hipótesis con un p-value <0:02352967535527367.

Si analizamos los resultados, podemos observar que ninguno de los *p-values,* a excepción de los correspondientes a Ranom y OneStepAhead, es suficientemente bajo para rechazar las hipótesis (recordamos que se establece como hipótesis nula, que las distribuciones de valores no son significativamente diferentes), por lo que no podemos afirmar que los resultados de los algoritmos más importantes sean significativamente diferentes.

4.6 NUEVO MARCO DE EXPERIMENTOS

Aunque en principio un conjunto de 10 juegos debería ofrecer suficiente variedad, se ha considerado conveniente repetir las pruebas sobre un nuevo conjunto de 10 juegos, para comprobar que los resultados no se hayan visto afectados por las circunstancias específicas de los juegos sobre los que se han realizado.

En este caso, presentaremos únicamente la tabla de puntuaciones máximas y la comparativa de rankings obtenidos por los algoritmos.

La nueva lista de juegos sobre las que se han repetido los experimentos son los correspondientes al *Training Set* de la CIG14:

- Aliens
- Boulderdash
- Butterflies
- Chase

- Frogs
- Missile Command
- Portals
- Sokoban
- Survive zombies
- Zelda

4.6.1 TABLA DE PUNTUACIONES MÁXIMAS

	Random	One Step Ahead	MCTS UCT	MCTS GREEDY	AG D7	AG D10	MCTS_AG SEQ	MCTS_AG PAR
J1	66.0	45.0	74.0	78.0	80.0	80.0	80.0	**84.0**
J2	3.0	2.0	22.0	6.0	7.0	5.0	**26.0**	22.0
J3	52.0	26.0	72.0	16.0	62.0	**74.0**	64.0	40.0
J4	**7.0**	1.0	**7.0**	3.0	**7.0**	**7.0**	**7.0**	**7.0**
J5	-2.0	**1.0**	**1.0**	-2.0	**1.0**	**1.0**	**1.0**	**1.0**
J6	-1.0	-3.0	2.0	2.0	**8.0**	5.0	2.0	5.0
J7	0.0	0.0	0.0	0.0	**1.0**	**1.0**	**1.0**	**1.0**
J8	0.0	0.0	0.0	0.0	**1.0**	0.0	0.0	**1.0**
J9	16.0	4.0	32.0	10.0	34.0	34.0	**45.0**	43.0
J10	7.0	1.0	4.0	6.0	**8.0**	**8.0**	**8.0**	**8.0**

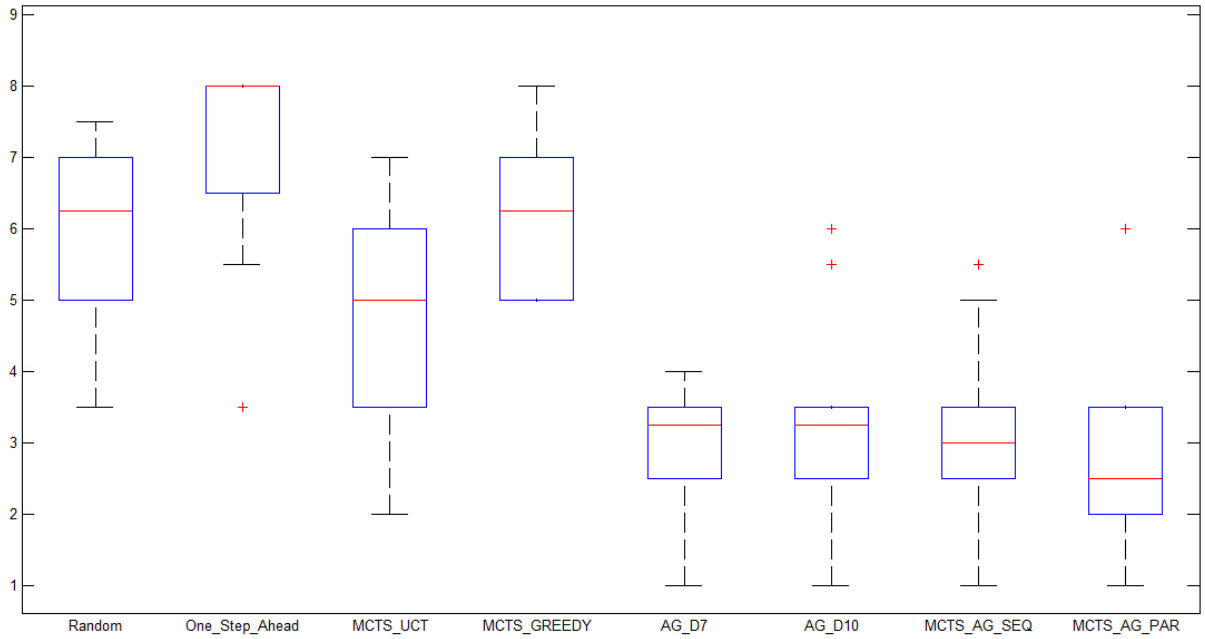
Tabla 4.7 Tabla de puntuaciones máximas de los algoritmos en cada uno de los juegos del Training Set CIG14. Cada celda representa la puntuación máxima obtenida por cada algoritmo obtenida tras ejecutarlos 10 veces

4.6.2 TABLA COMPARATIVA DE ALGORITMOS POR RANKING

	Random	One Step Ahead	MCTS UCT	MCTS GREEDY	AG D7	AG D10	MCTS_AG SEQ	MCTS_AG PAR
J1	7	8	6	5	3	3	3	**1**
J2	7	8	2.5	5	4	6	**1**	2.5
J3	5	7	2	8	4	**1**	3	6
J4	**3.5**	8	**3.5**	7	**3.5**	**3.5**	**3.5**	**3.5**
J5	7.5	**3.5**	**3.5**	7.5	**3.5**	**3.5**	**3.5**	**3.5**
J6	7	8	5	5	**1**	2.5	5	2.5
J7	6.5	6.5	6.5	6.5	**2.5**	**2.5**	**2.5**	**2.5**
J8	5.5	5.5	5.5	5.5	**1.5**	5.5	5.5	**1.5**
J9	6	8	5	7	3.5	3.5	**1**	2
J10	5	8	7	6	**2.5**	**2.5**	**2.5**	**2.5**

Tabla 4.8 Tabla de rankings de los algoritmos por cada juego. Cada celda representa la posición obtenida por cada juego respecto a los demás en función de su puntuación máxima

4.7 GRÁFICA DE DISTRIBUCIÓN DE ALGORITMOS POR RANKING



Gráfica 4.9 Diagrama de cajas representando la distribución de posiciones respecto al ranking de cada uno de los algoritmos. En este caso se intenta minimizar sus valores

4.7.1.1 ANÁLISIS DE LOS RESULTADOS

Aunque en este caso la diferencia es algo menor, aun es observable la tendencia del algoritmo híbrido competitivo a situarse levemente por encima (minimizando su posición de ranking), demostrando que aún en un marco de 10 juegos completamente diferentes, las conclusiones obtenidas anteriormente siguen siendo válidas.

5 CONCLUSIONES

Esta última sección está dedicada a las conclusiones obtenidas tanto de forma personal, del desarrollo y académicas en el resultado final. También se comentarán futuras modificaciones pensadas para continuar el desarrollo de la investigación.

A lo largo de este proyecto se ha estudiado el área de investigación del *General Game Playing,* una rama de la Inteligencia Artificial de gran complejidad debido a su naturaleza, ya que propone imitar a la inteligencia humana cuando se le plantea un nuevo juego desconocido, explicando simplemente las reglas del mismo. En los últimos años ha aumentado el interés y los investigadores que se han decidido por este tema, aunque a pesar del elevado número de publicaciones, resulta muy difícil la obtención de resultados realmente concluyentes, debido nuevamente a la propia naturaleza del problema. En el caso particular de este proyecto, se han implementado varios algoritmos con la finalidad de compararlos entre sí y evaluar su comportamiento en diferentes juegos. Para ello se realizaron experimentos para extraer información cuyo análisis nos ha permitido sacar conclusiones.

Analizando estos resultados, podemos concluir que de los algoritmos comparados no se encuentra un claro ganador. Aunque se observa una tendencia a la victoria del algoritmo híbrido competitivo sobre los demás, el resultado de los test estadísticos no han podido demostrar que sus distribuciones de valores sean significativamente diferentes.

La variedad de las puntuaciones se debe a que diferentes juegos han producido diferentes ganadores, por lo que afirmar que a día de hoy, todavía resulta muy difícil encontrar un algoritmo que se convierta en experto jugador de cualquier tipo posible de juego. La conclusión no se aleja mucho de lo esperado, ya que si la base del *General Game Playing*, es la de imitar la naturaleza humana a la hora de enfrentarse con juegos nuevos desconocidos, precisamente resulta difícil encontrar un jugador humano que destaque realmente en cualquier tipo de juego, y no solamente en algún/os tipos concretos.

5.1 MEJORAS Y TRABAJO FUTURO

Para un trabajo futuro, sería de gran interés seguir profundizando en técnicas relacionadas con el algoritmo MCTS, ya que ofrece grandes oportunidades de expansión, como por ejemplo, añadir aprendizaje sobre el estado del juego o reconocimiento de patrones [Pérez y otros, 2014], así como seguir trabajando en propuestas avanzadas de hibridación que combinen las ventajas de los diferentes algoritmos. También resultaría de gran interés continuar con la experimentación, realizando nuevos experimentos y profundizando en el análisis de los resultados. Además, podrían considerarse nuevos conjuntos de juegos, o realizar comparativas con jugadores humanos.

En general, la investigación sobre *General Game Playing* se encuentra en una etapa temprana, de forma que se podría ampliar de manera que no sólo se limite a juegos de un jugador, o juegos simples juegos clásicos, sino aplicarlo a juegos multijugador, con físicas en tiempo real, o juegos parcialmente observables.

Además, los resultados de futuras investigaciones no tienen que limitarse al entorno de los videojuegos, sino que pueden aprovecharse de este marco para desarrollar soluciones que se apliquen a diferentes áreas como la planificación, robótica, o resolución de problemas aplicados a cualquier área.

5.2 APRENDIZAJE PERSONAL

A nivel personal, este proyecto me ha supuesto un gran reto, ya que ha supuesto una primera toma de contacto con la investigación. Este tipo de trabajo se aleja en gran medida del tipo de trabajos realizados durante la realización del grado, ya que ha implicado formarme desde cero en un área que hasta el momento era completamente desconocida para mí, así como en técnicas de investigación y experimentación.

Como aportes positivos, este proyecto me ha enseñado a investigar y recopilar información de diferentes publicaciones, sintetizando y extrayendo las ideas más importantes, así como el descubrimiento de nuevas e importantes técnicas como *Montecarlo Tree Seach*.

También me ha ayudado a dar un primer paso en las técnicas de experimentación para la investigación, aprendiendo algunas metodologías de experimentación y presentación de resultados, así como su análisis estadístico, que me servirán como base para futuras tesis o investigaciones.

5.3 PROBLEMAS TÉCNICOS ENCONTRADOS

Los principales problemas con los que me he encontrado han sido la falta de conocimientos previos en el área, la cual se encuentra en pleno desarrollo y en muchas ocasiones, al igual que ha sucedido con este proyecto, no se obtienen resultados plenamente concluyentes.

Otro de los principales problemas que he sufrido ha sido la realización de los experimentos, ya que una ejecución completa de todos los algoritmos con todos los juegos, de manera que pudiese extraer toda la información necesaria, suponía un tiempo no inferior a 5 horas debido a la naturaleza de los problemas. Por lo tanto, cualquier fallo detectado o modificación necesaria obligaba a repetir nuevamente los experimentos al completo.

El análisis de los resultados de los experimentos ha supuesto otro problema debido a enfrentarme por primera vez con este tipo de metodología, lo cual me ha llevado a emplear mucho tiempo para estudiar y aprender el significado de muchos de los aspectos que esto implica.

Estos problemas, añadidos a la dificultad para disponer de tiempo al tener que compaginar el desarrollo con un trabajo a jornada completa, han provocado que no pudiese profundizar en la experimentación tanto como esperaba.

Pero a pesar de lo aquí nombrado, el desarrollo de este proyecto ha sido una importante experiencia altamente satisfactoria.

6 BIBLIOGRAFÍA

[Bellemare y otros, 2013] **M. G. Bellemare, Y. Naddaf, J. Veness and M. Bowling:** *The Arcade Learning Environment: An Evaluation Platform for General Agents.* Journal of Artificial Intelligence Research 47, pp. 253-279, 2013

[Benbassat y Sipper, 2013] **Amit Benbassat and Moshe Sipper:** *EvoMCTS: Enhancing MCTSbased*

Players Through Genetic Programming. Proceedings of the Conference on Computational Intelligence and Games (CIG), pages 57–64, 2013.

[Browne y otros, 2012] **Cameron Browne, Edward J. Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez,Spyridon Samothrakis, Simon Colton:** *A Survey of Monte Carlo Tree Search Methods.* IEEE Trans. Comput. Intellig. and AI in Games 4(1): 1-43 (2012)

[Derrac y otros, 2011] **J. Derrac, S. García, D. Molina, F. Herrera:** *A Practical Tutorial on the Use of Nonparametric Statistical Tests as a Methodology for Comparing Evolutionary and Swarm Intelligence Algorithms*. Swarm and Evolutionary Computation 1:1 (2011) 3-18

[Ebner y otros, 2013] **Marc Ebner, John Levine, Simon M. Lucas, Tom Schaul, Tommy Thompson, Julian Togelius:** *Towards a Video Game Description Language*. Artificial and Computational Intelligence in Games 2013: 85-100

[Finnson y Björnsson ,2011] **H. Finnson, Y. Björnsson:** *Game-tree properties and MCTS performance. :* Proceedings of the 11th IJCAI Workshop on General Game Playing (GIGA '11).

[Frydenberg y otros, 2015] **Frederik Frydenberg , Kasper R. Andersen , Sebastian Risi, Julian Togelius:** *Investigating MCTS Modifications in General Video Game Playing.* Proceedings of the IEEE Conference on Computational Intelligence and Games, 2015.

[Gen y Cheng, 1996] **Gen, M. and Cheng, R. :** *Foundations of Genetic Algorithms, in Genetic Algorithms and Engineering Design*.John Wiley & Sons, Inc., Hoboken, NJ, USA., 1996

[Genesereth y otros, 2005] **Michael R. Genesereth, Nathaniel Love, Barney Pell**: *General Game Playing: Overview of the AAAI Competition*. AI Magazine 26(2): 62-72 (2005)

[Goldberg, 1989] **David E. Goldberg:** *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA ©1989

[Hui y otros, 2014] **S. Hui, P.N. Suganthan, F. Herrera**: *Analyzing Convergence Performance of Evolutionary Algorithms: A Statistical Approach*. Information Sciences 289 (2014) 41-58

[Lara-Cabrera y otros, 2015] **Raúl Lara-Cabrera, Mariela Nogueira Collazo, Carlos Cotta, Antonio José Fernández Leiva**: *Game Artificial Intelligence: Challenges for the Scientific Community*. CoSECivi 2015: 1-12

[Levine y otros, 2013] **John Levine, Clare Bates Congdon, Marc Ebner, Graham Kendall, Simon M. Lucas, Risto Miikkulainen, Tom Schaul, Tommy Thompson:** *General Video Game Playing*. Artificial and Computational Intelligence in Games 2013: 77-83

[Maes y otros, 2013] **Francis Maes, David Lupien St-Pierre, Damien Ernst:** *Monte Carlo Search Algorithm Discovery for Single-Player Games.* IEEE Trans. Comput. Intellig. and AI in Games 5(3): 201-213 (2013)

[Mandziuk y Świechowski ,2012] **Jacek Mandziuk, Maciej Swiechowski:** *Generic Heuristic Approach to General Game Playing*. SOFSEM 2012: 649-660

[Mnih y otros, 2015] **V Mnih K Kavukcuoglu D Silver A Rusu J Veness M Bellemare A Graves M Riedmiller A Fidjeland G Ostrovski S Petersen C Beattie A Sadik I Antonoglou H King D Kumaran D Wierstra S Legg D Hassabis:** *Human Level Control Through Deep Reinforcement Learning*. Nature 2015

[Pérez y otros, 2014] **Diego Pérez, Spyridon Samothrakis, Simon M. Lucas:** *Knowledge-based fast evolutionary MCTS for general video game playing.* CIG 2014: 1-8

[Schaul, 2013] **Tom Schaul:** *A video game description language for model-based or interactive learning*. CIG 2013: 1-8

[Świechowski y otros, 2015] **Maciej Świechowski,HyunSoo Park, Jacek Mańdziuk, Kyung-Joong Kim:** *Recent Advances in General Game Playing*. The Scientific World Journal. Vol. 2015.

[Coursera GGP] Coursera General Game Playing Course. [Accedido el: 5 de 9 de 2015.] https://es.coursera.org/course/ggp

[DeepMind] Google DeepMind [Accedido el: 5 de 9 de 2015] http://deepmind.com/index.html

[GVGAI] GVG AI Competition. [Accedido el: 5 de 9 de 2015.] http://gvgai.net

[IEEE CIG] IEEE Computational Intelligence and Games [Accedido el: 5 de 9 de 2015.] http://www.ieee-cig.org/

[MCTS] Montecarlo Tree Search. [Accedido el: 5 de 9 de 2015.] www.mcts.ai.

[GGP Wiki] General Game Playing. Wikipedia. [Accedido el: 5 de 9 de 2015.] http://en.wikipedia.org/wiki/General_game_playing.

[MCTS Wiki] Monte Carlo tree search.Wikipedia. [Accedido el: 5 de 9 de 2015.] https://en.wikipedia.org/wiki/Monte_Carlo_tree_search.

ANEXO I: GUÍA DE INSTALACIÓN Y MANEJO DEL GVGAI FRAMEWORK

DESCARGA E INSTALACIÓN

El Framework se puede obtener en la página oficial de la GVGAI Competition: http://www.gvgai.net/software.php de manera gratuita y sin necesidad de registro.

También es posible acceder y clonar directamente su repositorio disponible en: https://github.com/EssexUniversityMCTS/gvgai

Una vez descargado, solo tenemos que extraer los ficheros en nuestro espacio de trabajo de *eclipse* y podremos empezar a trabajar con él.

CREANDO UN NUEVO CONTROLADOR

Para crear un nuevo controlador, tan solo tenemos que crear un nuevo paquete con el nombre de usuario que ha registrado para la competición. Si no se va a participar en la competición, se puede poner cualquier nombre, o incluir el paquete dentro del paquete *controllers* (como se ha hecho en este proyecto).

Dentro de ese paquete debe incluir como mínimo la clase principal del agente, que se debe llamar **Agent.java** y heredar de *AbstractPlayer,* aunque se pueden añadir todos los ficheros adicionales que sean necesarios. La estructura de paquetes resultante debe tener el siguiente formato:

- controllers
- core
- ontology
- tools
- nombre
    |- Agent.java
    |- FicheroAdicional1.java
    |- FicheroAdicional2.java

Para su funcionamiento, el controlador solo requiere que se implementen dos métodos:

- `public Agent(StateObservation stateObs, ElapsedCpuTimer elapsedTimer)` (*Constructor*)
- `public Types.ACTIONS act(StateObservation stateObs, ElapsedCpuTimer elapsedTimer)`

donde:

- `StateObservation stateObs` es un estado observable del juego
- `ElapsedCpuTimer elapsedTimer` es el temporizador para controlar la duración del método
- `Types.ACTIONS` es el valor de retorno que representa la siguiente acción a tomar

EJECUTANDO UN CONTROLADOR

La clase encargada de ejecutar los controladores sobre los distintos juegos es **Test.java**, localizada en el directorio raíz.

Lo primero que nos encontramos, son variables con la localización de los juegos, y una lista con los juegos disponibles:

```
    //Available games:
    public static String gamesPath = "examples/gridphysics/";

    //CIG 2014 Training Set Games
    public static String games[] = new String[]{"aliens", "boulderdash",
"butterflies", "chase", "frogs",
            "missilecommand", "portals", "sokoban", "survivezombies", "zelda"};
```

Además, algunos parámetros generales que permiten decidir si mostrar visualmente el juego, y definir un fichero de texto donde se registren todas las acciones tomadas por el controlador.

```
//Other settings
    static boolean visuals = true;
    static String recordActionsFile = null;
```

Dentro del método principal, tenemos que guardar la ruta de cada controlador en una variable y agregarlos a una lista de agentes:

```
//Available controllers:
String randomController = "controllers.sampleRandom.Agent";
String OneStepController = "controllers.sampleonesteplookahead.Agent";
String MCTS_UCT = "controllers.MCTS_UCT.Agent";
String MCTS_GREEDY = "controllers.MCTS_GREEDY.Agent";
String AG_D7 = "controllers.AG_D7.Agent";
String AG_D10 = "controllers.AG_D10.Agent";
String MCTS_GA_PAR = "controllers.MCTS_GA_PAR.Agent";
String MCTS_GA_SEQ = "controllers.MCTS_GA_SEQ.Agent";

agentList.add(randomController);

…
```

Una vez recogidos los juegos y los controladores, disponemos de 4 modos de ejecución diferentes:

- Jugar a un juego/nivel como humano:
  `ArcadeMachine.playOneGame(game, level1, recordActionsFile, seed);`
- Jugar a un juego/nivel con un controlador:
  `ArcadeMachine.runOneGame(game, level1, visuals, CONTROLADOR, recordActionsFile, seed);`
- Re jugar un juego/nivel desde un fichero obtenido en una jugada anterior
  `ArcadeMachine.replayGame(game, level1, visuals, readActionsFile);`
- Jugar a un juego con N niveles, M veces:
  `ArcadeMachine.runGames(game,levels, M, sampleRandomController, null, seed);`

Para facilitar la experimentación, se ha creado un método adicional llamado **fullAgentTest()** , que ejecuta todos los controladores en N juegos, L niveles, M veces, y guarda los resultados en un fichero csv con el nombre de cada controlador en una carpeta llamada estadísticas.

El fichero csv contiene línea de cabeceras y una de valores por cada uno de los juegos. Un ejemplo de línea para un juego es el siguiente:

pacman	Jugadas	Minima	Maxima	Media	DesviacionTipica	Valores			
	10	93.0	225.0	164,9	46,75	196.0	134.0	126.0	…

ANEXO 2: GUÍA USO DE HERRAMIENTA PARA TEST ESTADÍSTICOS

Para la realización de los test estadísticos, se ha utilizado una herramienta desarrollada por el grupo de investigación SCI2S de la universidad de Granada. Se trata de una herramienta programada en Java que podemos ejecutar por línea de comandos.

DESCARGA E INSTALACIÓN

El software se puede obtener gratuitamente, descargándolo desde su página web: http://sci2s.ugr.es/sicidm en la sección titulada: Software and User's Guide.

No es necesario realizar ningún tipo de instalación, tan solo extraer los ficheros descargados, y el programa estará listo para funcionar.

EJECUCIÓN DEL PROGRAMA Y ENTRADA DE DATOS

El programa se ejecuta con una simple llamada por consola, escribiendo lo siguiente: **java Friedman < fichero de datos > > < fichero de salida >**

Donde fichero de datos es un fichero con formato csv que debe seguir el siguiente formato:

La primera fila contiene la palabra **Dataset ,**seguida de los nombres de los algoritmos separados por comas.

Y por cada uno de los casos de prueba, una línea con su nombre y los valores de cada algoritmo separados por comas.

Por ejemplo:

Dataset,C4.5,1NN,NaiveBayes,Kernel,CN2

Cleveland,0.54,0.531397,0.55784,0.4387,0.541

Glass,0.67442,0.7361,0.720,0.355,0.7041498

…

RESULTADOS Y FORMATO DE SALIDA

Una vez ejecutado, el programa realiza varios test estadísticos sobre los datos, y devuelve las tablas de resultado en un documento en formato LaTeX, el cual podemos abrir con un editor y exportarlo a formato PDF.