TRABAJO FINAL DE LA ASIGNATURA

TETRIS AUTOMATIZADO

INTELIGENCIA ARTIFICIAL

MÁSTER AUTOMÁTICA Y ROBÓTICA

AUTORES

David Thomas Ruiz

Jorge Gómez-Pamo González-Cela

Tomas Sánchez Villaluenga

Índice

Índice de Figuras

Índice de Tablas

1. Introducción

En este proyecto, se ha trabajado en la automatización del juego del Tetris mediante la implementación de algoritmos avanzados en el lenguaje de programación C++. Esta iniciativa surgió como una evolución de un trabajo previo realizado en lenguaje C durante el primer año de nuestra formación académica. Nuestro enfoque se centró en adaptar y mejorar este juego icónico con el objetivo de capacitarlo para jugar de forma autónoma y ajustarse automáticamente en busca de la estrategia óptima que maximizara la puntuación.

El objetivo primordial consistió en desarrollar un sistema capaz de tomar decisiones inteligentes en tiempo real, fundamentadas en algoritmos avanzados. Para lograr este cometido, nos concentramos en dos áreas fundamentales: la aplicación de algoritmos basados en la función de ponderación y la integración de algoritmos genéticos de evolución. Estas estrategias permitieron al juego analizar y evaluar las diversas opciones disponibles en cada momento, con la finalidad de seleccionar las decisiones más favorables para maximizar su puntuación.

Además, uno de los desafíos fundamentales abordados en el desarrollo del código de este programa ha sido la implementación de una visión predictiva en el juego. Esta habilidad permitiría al sistema anticiparse, considerando los posibles escenarios futuros inmediatos y ajustando sus movimientos en consecuencia. El propósito principal radicaba en dotar al juego de una capacidad estratégica que superara la simple reacción a eventos presentes, posibilitándole anticiparse a futuras situaciones y tomar decisiones más precisas y ventajosas.

2. Juego del Tetris

El Tetris es un juego de puzzle clásico en el que se deben encajar piezas geométricas descendentes para formar líneas horizontales completas. Cada vez que se completa una línea, esta desaparece y se obtienen puntos, aumentando si se eliminan múltiples líneas simultáneamente, conocido como hacer un "Tetris". Conforme avanza el juego, la velocidad de caída de las piezas se incrementa, desafiando al jugador a mantener el tablero despejado para acumular la máxima puntuación antes de que las piezas se acumulen hasta la parte superior y finalice la partida.

2.1. Explicación de cómo usar la aplicación

Al iniciar la aplicación el usuario puede elegir entre tres opciones. La primera opción es entrenar a la IA, la segunda opción es ver a la IA jugar al Tetris y la última opción es simplemente jugar al Tetris.

2.1.1. Entrenamiento de la IA

Si el usuario elige entrenar a la IA, se cambiará el menú y se le preguntará al usuario cual de tres métodos quiere usar para entrenar a la IA. Cuando elige qué entrenamiento usar, se muestra la interfaz gráfica de ese entrenamiento. La interfaz es de la siguiente forma:

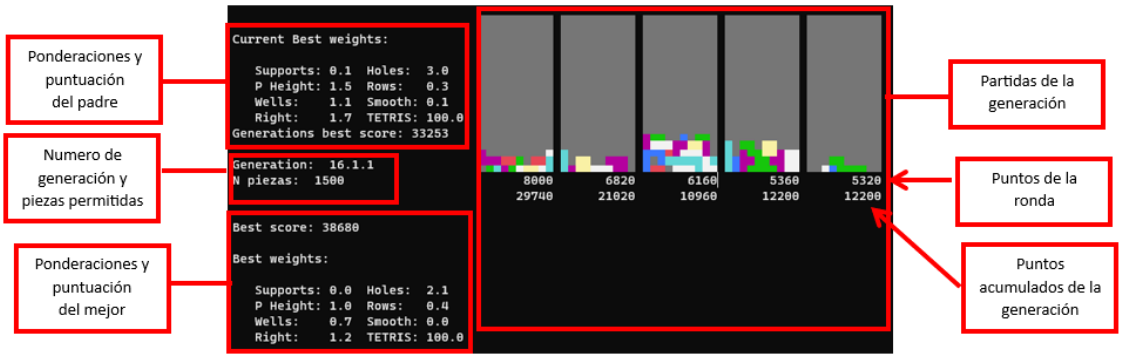


Figura 1. Pantalla de entrenamiento

2.1.2. Ver a la IA jugar al Tetris

En este modo el usuario ve jugar a la IA que ha sido entrenada, así poder ver el funcionamiento de la IA, poder ver hasta qué puntuación es capaz de llegar la IA y poder estudiar posibles comportamientos que no gusten. De estos posibles comportamientos se hablará más en el apartado de "Posibles Mejoras". A continuación, se muestra la interfaz gráfica de este menú, es el mismo que para el Tetris normal.

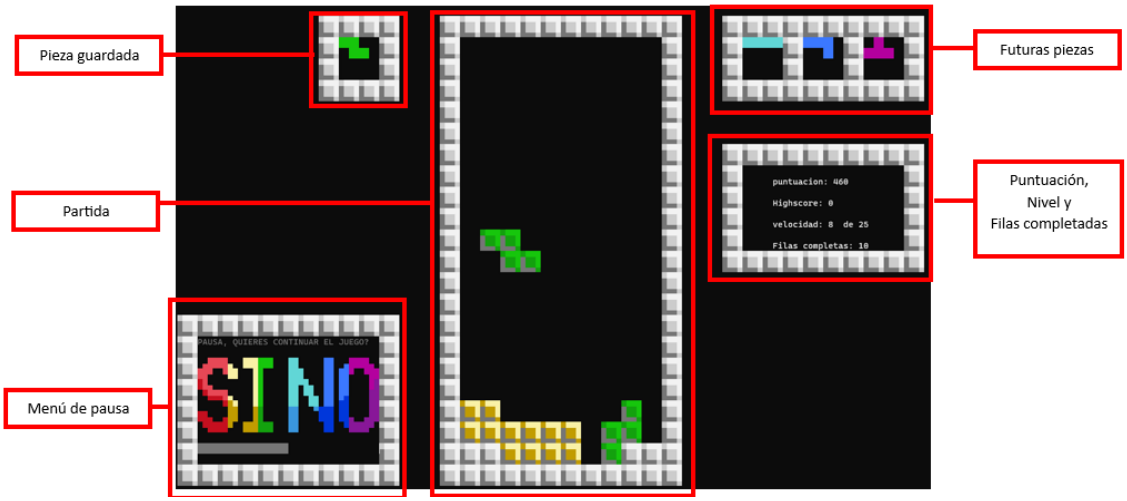


Figura 2. Pantalla visualización IA

2.1.3. Tetris normal

Este modo de juego representa la versión clásica, permite al usuario jugar al Tetris utilizando las teclas del ordenador. A medida que avanza el tiempo de juego y se acumulan puntos, la velocidad de descenso de las piezas también aumenta progresivamente. El usuario puede mover la pieza hacia la izquierda con la tecla 'a', hacia

la derecha con la tecla ‘d’, moverla hacia abajo con la tecla ‘s’ y rotar la pieza en sentido horario con la tecla ‘w’, estas son las teclas del teclado comúnmente usadas en videojuegos. Con la tecla ‘tab’, el usuario puede depositar la pieza en la zona de guardado para utilizarla en otro momento y con la tecla ‘escape’ puede pausar el juego y decidir si empezar una nueva partida o dejar de jugar o continuar la partida. Nuestro récord es una puntuación de 19140 puntos, ¡mucha suerte!

3. Funcionamiento de la IA

Al comienzo de la caída de una pieza se llama a la clase AI, esta busca todos los posibles lugares donde puede acabar la pieza haciendo uso del algoritmo de búsqueda Breadth First Search, empezando desde la posición de la pieza. Los posibles lugares donde puede acabar la pieza son aquellos donde la pieza pueda llegar haciendo uso de los movimientos que permite Tetris y la pieza esté apoyada encima de otra pieza. En cada posición se comprueba cuál de los cuatro movimientos puede realizar la pieza, estos siendo, moverse a la derecha, izquierda o abajo o rotar. Por cada movimiento permitido se añade un nodo del grafo conectado al nodo de la posición actual, si una posición exitosa ya forma parte del grafo, entonces no se tiene en cuenta. Cuando el BFS se queda sin lugares que visitar se sale del algoritmo.

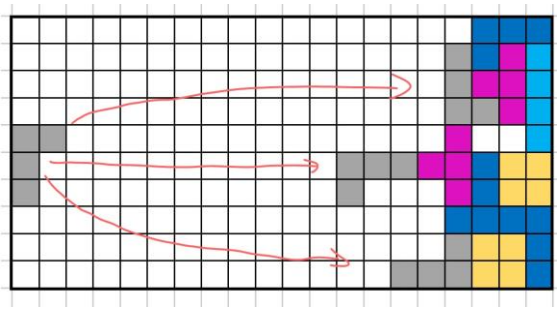


Figura 3. Funcionamiento del algoritmo de búsqueda

Una vez que la IA ha encontrado todos los posibles lugares, se encarga de determinar cuál es el mejor lugar para la pieza. Para esto se obtienen una serie de características, actualmente se hace uso de 8 características, pero se pueden implementar tantas como se vea razonable. Cada característica tiene una ponderación, que determina el peso que tiene esa característica sobre la puntuación de la posición. Estas ponderaciones son las que se ajustan al entrenar la IA, esto se verá en la sección titulada “Entrenamiento de la IA”.

Una vez determinada la mejor posición y puntuación de la figura, se repite, pero con la figura que hay guardada, para determinar cuál de las dos piezas sería la mejor para usar. Si la pieza guardada encuentra una posición con una puntuación mayor entonces la IA mandará la instrucción al juego de cambiar la pieza.

Una vez que se te tiene la mejor de las dos piezas y cuál sería esa posición, se hace uso del algoritmo de búsqueda A* desde la posición final hasta donde empieza la pieza. Como se puede ver, a diferencia del BFS, el A* realiza la búsqueda al revés, esto es debido que la zona de la posición final está más restringida con obstáculos que la posición inicial, por lo que empezando desde el final el algoritmo tendrá menos ramas de búsquedas. La

función de evaluación de cada nodo está formada por dos partes, el coste del camino óptimo desde el comienzo hasta el nodo, g(x) y de la estimación del camino óptimo hasta el final, h(x).

El valor de g(x) se obtiene sumando todos los movimientos que se ha realizado hasta llegar al nodo y el valor de h(x), se hace sumando los movimientos que se ha de realizar para llegar al final sin tener en cuenta los obstáculos. Un cambio que se ha realizado con respecto al algoritmo oficial es que la función de evaluación se incrementa en cinco si el nodo hijo es debido a un movimiento hacia abajo, esto se ha hecho así para fomentar que la IA primero mueva y gire la pieza al lugar correcto antes de empezar a mover la pieza hacia abajo. De esta forma permite que cuando es necesario mover la pieza hacía abajo para moverse por obstáculo pueda, pero que sino evite hacerlo hasta el final.

Una vez ha determinado el camino óptimo para llevar a la pieza hasta su destino, obtiene los movimientos que necesita realizar y los invierte, ya que el algoritmo se hace desde el destino hasta el inicio.

Finalmente, cada vez que el juego permita a la IA introducir un movimiento está dará el movimiento que toque de lista. Es importante recordar que Tetris cada cierto tiempo baja la pieza de forma automática. Por lo tanto, mientras la IA va dando los comandos en orden, mira si el comando que da es de mover la pieza hacia abajo o no. En el caso de que lo sea, comprueba que no vaya a hacer que se choque la pieza, ya que la IA espera que la pieza esté en una posición, pero como Tetris mueve las piezas hacia abajo cada cierto tiempo, puede no estar en esa posición. Si la instrucción de mover la pieza hacia abajo hiciese que chocase con una pieza, pasará a la siguiente instrucción. Una vez que la pieza ha llegado al destino y la siguiente pieza comienza a caer, se repite el proceso.

3.1. Parámetros de la puntuación

- El número de bloques que forman la pase de la pieza, para favorecer que tenga la base más grande y la altura de la pieza sea la menor.
- El número de agujeros que introduce la pieza, estos son huecos que quedan debajo de la pieza a los que no se puede acceder, por lo tanto, no se podrán rellenar y hacen que la montaña vaya a ser más alta, esto es muy indeseable, a medida que se crean más agujeros la partida se vuelve más complicada.
- La altura a la que se queda la pieza, cuanto más baja se quede la pieza es mejor.
- El número de filas que elimina la pieza y cuantas más se eliminen de una mejor.
- El número de pozos que crea la pieza, pozos son agujeros de más de tres bloques de altura, estos solo pueden ser correctamente rellenados con la pieza larga, ya que cualquier otra pieza dejaría huecos. La creación de estos pozos incrementa la necesidad de obtener una pieza larga, que si no llega complicará la partida.
- Como de plana queda la montaña, se obtiene restando la altura entre las columnas consecutivas. Cuanto más plano es el mapa, más fácil es la partida.
- Otra característica es evitar dejar piezas en la columna más a la derecha, de esta forma se crea un pozo en esa zona, que puede ser rellenado por una pieza larga para conseguir un Tetris e incrementar los puntos que se pueden conseguir.
- La última propiedad es si se consigue un Tetris, que es eliminar cuatro filas, conseguir un Tetris es la mejor forma de incrementar la puntuación. Si existe la posibilidad de hacer un Tetris, no lo duda pone allí la piezas.

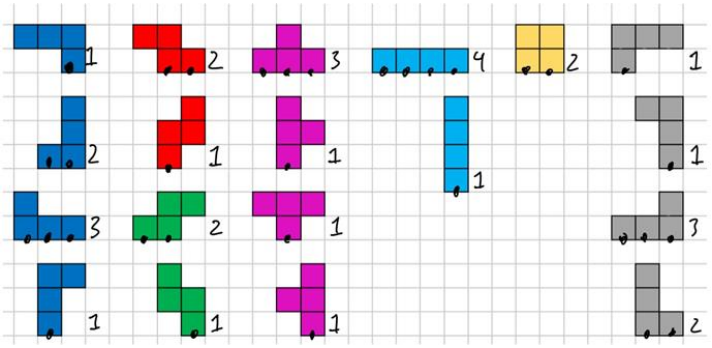
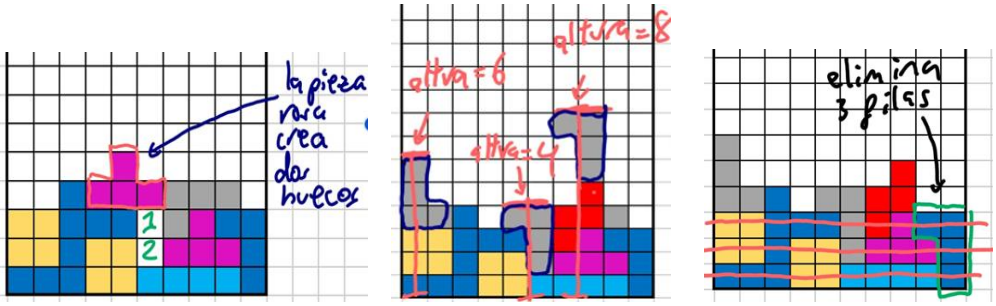
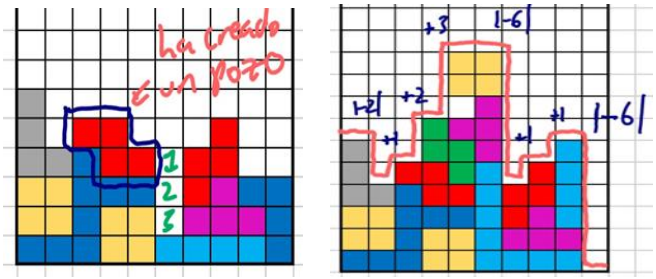


Figura 4. Tipos de piezas y movimientos



(a) Huecos *(b) Altura* *(c) Filas*



(d) Pozos *(e) Columna derecha*

Figura 5. Representación de parámetros

3.2. Parámetros adicionales

A pesar de que los parámetros presentados previamente en el método basado en la función de ponderación funcionaban bien, su limitación reside en su enfoque actual. Estos parámetros únicamente otorgan ponderaciones según el desempeño de una pieza en el momento presente del juego, sin considerar las piezas que se jugarán en el futuro cercano. Esto implica la falta de una visión prospectiva en el juego.

Para abordar esta limitación, se ha intentado mejorar el método mediante la implementación de algoritmos adicionales. Estos algoritmos, además de basarse en los parámetros existentes, evalúan las próximas piezas con el objetivo de prever movimientos futuros. Esta estrategia permite actuar en los movimientos actuales teniendo en cuenta las posibles situaciones futuras del juego.

3.2.1. Método de ajuste perfecto propuesto

Se propuso un enfoque para otorgar mayor importancia al ajuste de las piezas dentro de la estructura que va desarrollándose en el transcurso del juego. Este método buscaba replicar de manera más fiel la dinámica real del ser humano al jugar al Tetris, donde se acostumbra a dejar espacios que se correspondan con la forma de otras piezas, permitiendo un ajuste perfecto cuando una pieza con esa misma forma aparezca.

El objetivo principal de este método era evaluar constantemente la morfología presente durante el juego y determinar la posición óptima para encajar una nueva pieza, evitando áreas donde otras piezas (ya sea la pieza en espera o las tres siguientes en la secuencia) pudieran ajustarse perfectamente.

Para implementar esta lógica, el programa necesitaba tener conocimiento de las diversas formas de las piezas existentes y sus posibles rotaciones. Se desarrolló una función específica para cada tipo de figura y rotación. Estas funciones evaluaban la compatibilidad de una pieza en particular, con su respectiva rotación, en todas las posiciones disponibles en la pantalla.

Para determinar esta compatibilidad, se establecieron escenarios ideales en los que una pieza encajaría perfectamente en un espacio vacío, ponderando este escenario en función de cuántas piezas podrían ajustarse en dicho espacio. Por ejemplo, si había una columna con 5 bloques ocupados verticalmente en la posición "i", 5 bloques vacíos en la posición "i+1" y nuevamente 5 bloques ocupados en la posición "i+2", solo la pieza en forma de palo de 5 bloques (*Figura 6*) encajaría perfectamente. Por ende, se priorizaba colocar esa pieza en ese espacio, ya que ninguna otra tendría un ajuste igual o mejor. Esta pieza recibía la ponderación máxima. No obstante, había espacios donde varias piezas podían encajar de manera igual, por lo que su ponderación sería similar en casos más generales, o variaría en casos más específicos (*Figura 7*).

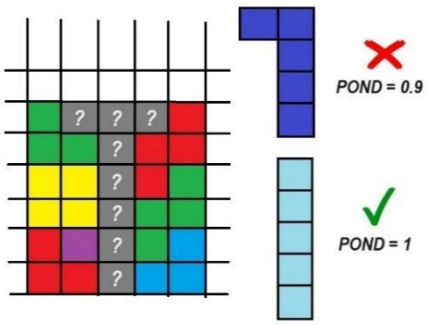


Figura 6. Ejemplo de ajuste perfecto

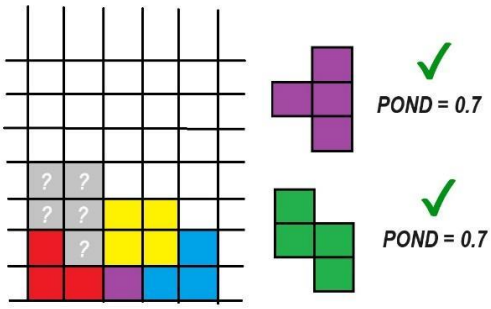


Figura 7. Ejemplo de ajuste similar

La intención era considerar este aspecto como un parámetro adicional, aunque crucial (inicialmente asignándole un valor muy alto), con el propósito de, durante el proceso de entrenamiento, determinar su verdadera relevancia. Se buscaba integrar este parámetro en la ponderación global de puntos, junto con otros factores, para influir en la toma de decisiones finales. Sin embargo, este parámetro tenía un submétodo de ponderación interno que confería un peso significativo en la ponderación general.

En la práctica, al intentar implementar esta funcionalidad, surgieron problemas de ineficiencia durante el entrenamiento. Evaluar todos los casos celda por celda en la pantalla requería una gran capacidad computacional, lo que limitaba la posibilidad de

realizar entrenamientos prolongados. Esta limitación resultaba en una evolución lenta de las ponderaciones del juego. Por consiguiente, debido a esta dificultad, se tomó la decisión de no incluir esta funcionalidad en el código final.

4. Entrenamiento de la IA

Con todo lo anterior ya se ha conseguido un programa que codifique el funcionamiento de una máquina que sea capaz de jugar al Tetris mediante el ajuste de las ponderaciones mencionadas. De esta forma, el problema actual reside en encontrar los valores reales que deben tener estas ponderaciones para que el programa pueda conseguir la mejor puntuación posible al Tetris y evitar morir. Para ello, se han desarrollado tres métodos de entrenamiento para ajustar estos parámetros.

4.1. Entrenamiento por competiciones

El primero de estos métodos es el entrenamiento por competición, donde se ejecutan dos partidas de tetris por cada parámetro que tiene la IA, siendo estos parámetros con los que determina la mejor posición para cada pieza. Se deja a cada partida jugar hasta que hayan caído un número determinado de piezas, este número irá incrementado con el número de generaciones. Se limita el número de piezas para que el entrenamiento sea más rápido y ninguna partida se quede jugando de forma indefinida y también para que al principio se busque optimizar la puntuación más que la supervivencia y con el paso de las generaciones se irá fomentado más la supervivencia. Una vez que todas las partidas han perdido o han llegado al número máximo de piezas, se vuelven a dejar jugar todas sin cambiar sus ponderaciones otras dos veces. Una vez que se han terminado las tres rondas, se escoge la partida cuya media de puntuación sea la mejor y sus ponderaciones serán las que se pasan a la siguiente generación de partidas. Así, en cada generación se toma el cambio de ponderación que mejor puntuación haya tenido.

Todas las partidas son iguales, es decir, se usa la misma semilla para el generador de piezas aleatorias, por lo que el orden de caída de piezas es el mismo para todas las partidas, de esta forma evitamos que unas partidas sean más sencillas, sino que sean uniformes, pero en cada ronda se cambia la semilla, para evitar que la IA se sobreentrene con esa semilla y las ponderaciones no sean buenas para el resto de las semillas. En cada pareja de partidas se cambia el valor de la ponderación del parámetro correspondiente, sumándole un valor aleatorio entre 0.1 y 0.5 para la primera partida de la pareja y restándole a la segunda partida.

Por lo tanto, cada una de las partidas tendrá las mismas ponderaciones que el padre exceptuando una ponderación que habrá cambiado, siendo mayor o menor. Para mejorar el rendimiento del entrenamiento, no se ejecutan partidas que tengan ponderaciones negativas, ni la partida cuya ponderación es el cambio contrario a la ponderación que venció en la generación anterior, este último cambio es porque se espera que el mejor resultado no sea revertir el cambio hecho previamente, y de esta forma mejorar el rendimiento al ejecutar una partida menos.

4.2. Entrenamiento por gradiente

Este método de entrenamiento es muy similar al anterior, en este caso en cada generación se realizan cambios sobre un único parámetro. Se ejecutan cinco juegos cambiando este parámetro, el primero es incrementando en 0.5 ese parámetro, el segundo es incrementando en 0.1 el parámetro, el tercero es sin cambiar el parámetro, el cuarto es reduciéndolo en 0.1 y el último reduciéndolo en 0.5. Se deja jugar las partidas tres rondas durante un número limitado de piezas como en el entrenamiento anterior y se escoge el cambio que mejor resultado ofrece. Se llama entrenamiento por gradiente porque en cada generación se escoge el cambio que mejor resultado de, por lo que se van haciendo cambios en el gradiente hacia las ponderaciones óptimas. Al igual que en el entrenamiento anterior, para mejorar el rendimiento no se ejecutan aquellas partidas con ponderaciones negativas.

4.3. Algoritmo genético

El último método del que se ha optado por usar es un algoritmo de optimización genético, cuya idea es la de simular un conjunto (llamado generación) de instancias (que representa la población). Al final de la simulación, se analizan los resultados y se seleccionan los mejores individuos para pasar su genotipo a la siguiente generación, que se vuelve a simular. Así, se consigue que poco a poco se vayan ajustando los genes de los individuos para quedarse con los genes que conllevan un mejor comportamiento y optimizándolos.

Las características del algoritmo genético desarrollado son las siguientes:

4.3.1. Parámetros de la simulación

En las simulaciones, las generaciones tendrán una población de 14 individuos, tal y como está desarrollado en el *Punto 3.1*. De esta forma, hemos dejado este número igual, quedando como posible mejora el ver cómo este número puede influir a la velocidad de convergencia del algoritmo.

Así en la población, el fenotipo de cada individuo viene representado por su estrategia durante la simulación, y su genotipo por la combinación de ponderaciones, a partir de ahora “genes”, que son valores reales positivos.

Además, para cada partida simulada, el juego tendrá un límite máximo de 1135 piezas jugables. Hay que tener en cuenta que una vez alcanzado cierto nivel, los individuos se vuelven relativamente buenos, por lo que sin este límite el algoritmo tardaría una cantidad de tiempo enorme para avanzar.

Por último, para cada generación se simularán 3 partidas, siendo estas iguales para todos los individuos de la población. Una vez terminada la fase de juego, se procederá a la construcción de la siguiente generación y a su posterior simulación. Para construir la generación, se ejecutan los procesos explicados a continuación:

4.3.2. Selección de los Padres

Cada individuo necesitará dos individuos padres para su creación. Será así necesaria la programación de una lógica de selección que permita que los mejores individuos de la generación anterior pasen más veces sus genes a la nueva generación, pero sin eliminar por completo la posibilidad de que el resto de los individuos también pasen su información genética. El proceso que se ha programado en este trabajo para desarrollar este aspecto es el siguiente:

Primero, se clasifican todos los individuos de la generación anterior en base a la media de la puntuación en las 3 rondas jugadas (lo que se corresponde con la función de fitness elegida). Así, se introducen todos los individuos en las siguientes categorías:

- Categoría A: Con los tres mejores individuos de la generación
- Categoría B: Con los siguientes cinco mejores individuos
- Categoría C: Con los seis peores individuos de la generación

En segundo lugar, se elegirán los padres de cada nuevo individuo de la nueva generación. Estos se elegirán al azar entre los individuos de la generación antigua con una probabilidad dependiendo de la categoría a la que pertenezca. Las probabilidades de los individuos en función de su categoría aparecen en la Tabla 1.

Categoría del individuo	**Probabilidad de ser elegido**
Categoría A	16.67 %
Categoría B	6.00 %
Categoría C	3.33 %

Tabla 1. Probabilidad de selección de cada individuo para ser padre en función de su categoría.

Por último, se comprueba que los dos padres de un nuevo individuo no sean el mismo individuo de la generación anterior.

El único caso en el que esto no se cumple es para el mejor individuo de la generación antigua, que se elige automáticamente como padre por las dos partes para el primer individuo de la nueva generación. De forma que el mejor individuo de cada generación pasa siempre a la siguiente (aunque aún puede sufrir cambios en la mutación). Así, se puede asegurar que los mejores genes vayan pasando de generación en generación basándose menos en la probabilidad.

Este proceso puede mejorarse significativamente, sobre todo en la selección de las categorías de individuos y en las probabilidades de selección de cada individuo en base a esta. También, se puede eliminar la parte en la que el mejor individuo pase directamente a la siguiente generación. De esta forma, queda como posible mejora variar todos estos parámetros y analizar cómo esto afecta a la calidad y velocidad de convergencia del algoritmo.

Una vez elegidos los padres de cada individuo, se procede a la creación de estos nuevos individuos, detallada a continuación.

4.3.3. Creación de la nueva generación

Esta parte se puede dividir en dos tareas a realizar: un crossover de los genes de los padres para crear los del nuevo individuo, y un proceso de mutación de los genes del nuevo individuo.

El crossover realiza la función del algoritmo genético de exploración. Es decir, genera una descendencia ligeramente distinta a los padres, probando otras zonas del espacio de búsqueda para buscar nuevos posibles máximos. Para ello, en este trabajo se ha implementado un crossover de dos puntos.

Tras tener el genotipo del nuevo individuo creado, se procede a un proceso de mutación. Este busca optimizar los resultados, realizando pequeños cambios a los genes de forma aleatoria para buscar mejoras en la puntuación. En este trabajo, la mutación se ha programado como una función que afecta aleatoriamente al 10% de los genes y cuyo cambio está programado como la suma del gen a mutar y de una variable aleatoria distribuida mediante una normal fija de desviación estándar 0,5.

Por último, se comprueba que en la nueva generación no haya dos individuos idénticos (lo que gastaría recursos del algoritmo sin probar nuevas opciones). Y de esta forma, la nueva generación estaría lista para su simulación, con lo que empieza un nuevo ciclo.

Nótese que en este apartado también hay muchos parámetros que son susceptibles de cambios y, por lo tanto, de mejora: el procedimiento del crossover se podría cambiar, por un crossover aritmético, o uniforme. Por otra parte, el proceso de mutación puede mejorar en el cambio de los valores fijos escogidos, o añadiendo una mutación adaptativa para cambiar el radio de mutación a medida que la solución va convergiendo para optimizar con más finura.

5. Explicación del código

Todo el código incluyendo el juego y la IA, están programados en C++, el código cuenta con nueve clases, la clase Tetris que recoge todo lo necesario para que funcione el juego, la clase IA que recoge todas las propiedades y funciones que permiten a la IA tomar las decisiones y comunicarlas. La clase partida se encarga de llevar los tiempos del juego e ir llamando a las funciones correspondientes de la IA y de Tetris en cada momento de la partida. Hay una clase para cada uno de los tres métodos de entrenamiento y dos clases adicionales para el método genético que se encargan de obtener las mutaciones y las uniones de cromosomas de cada generación. Por último, está la clase semáforo, que se encarga de gestionar el uso de la pantalla del terminal por parte de las partidas que se están jugando a la vez.

El uso del semáforo es esencial, ya que el hecho de mover el cursor, cambiar el color de lo que se va a imprimir e imprimir la partida no es un proceso atómico, por lo que, si no se coordina, cuando una partida cambie la posición del cursor otra estará imprimiendo y se termina viendo dibujos aleatorios en los mapas del Tetris. Por ello, en cada momento solo una partida tiene acceso a modificar la pantalla del terminal. Esto es lo que más ralentiza el entrenamiento, ya que las partidas se quedan esperando a poder imprimir, por ello se evita ejecutar partidas que no ayudan al entrenamiento.

Las ponderaciones victoriosas junto con la puntuación y el número de piezas permitidas en cada generación se van guardando en un archivo de texto para después poder ser mostradas en gráficas. Las ponderaciones que han dado la mejor puntuación se guardan en otro archivo, el cual se lee al empezar el entrenamiento, para recapitular desde el mejor resultado del entrenamiento anterior.

6. Resultados

6.1. Resultados del entrenamiento de competición

Se ha entenado a la IA con este modo de entrenamiento dos veces, cada entrenamiento ha durado entorno a dos días. En ambos se ha comenzado con valores nulos de ponderaciones, la ponderación para la propiedad de si se hace Tetris se ha dejado a 100 y no se ha entrenado, ya que siempre se quiere que cuando pueda hacer un Tetris lo realicé sin dudarlo.

En el primer entrenamiento se ha permitido inicialmente partidas de 1000 piezas, cada treinta generaciones este numero se ha incrementado en 1.5. Finalmente, se paró el entrenamiento después de 350 generaciones, con un número máximo de piezas permitidas de 11390 y una puntuación máxima de 172700. Como se puede observar en las gráficas, la puntuación tiene la misma forma que el número de piezas permitidas, esto quiere decir, que las partidas han tendido a sobrevivir hasta que se ha llegado al numero de piezas máximas. Se ve que la puntuación por piezas ha ido decreciendo a medida que se alargaban las partidas, esto es debido a que para poder sobrevivir se tiene que ser menos agresivo, finalmente se ha conseguido una media de 15 puntos por pieza.

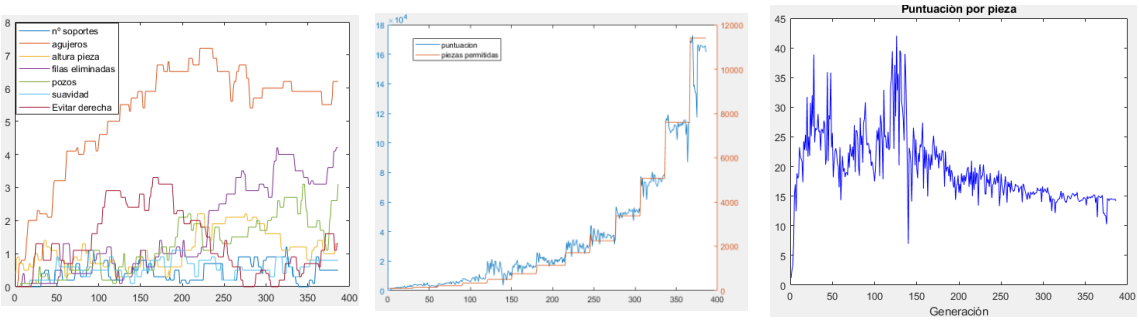


Figura 8. Gráficas evolución primer entrenamiento de competición

En el segundo entrenamiento se redujo el numero de generaciones que han de pasar para aumentar el número de piezas permitidas a 10, para así acelerar el entrenamiento. Se ha permitido entrenar durante 150 generaciones hasta un numero máximo de piezas de 25627 piezas y una puntuación máxima de 394340 puntos. Finalmente, también se obtuvo una media de 15 puntos por pieza.

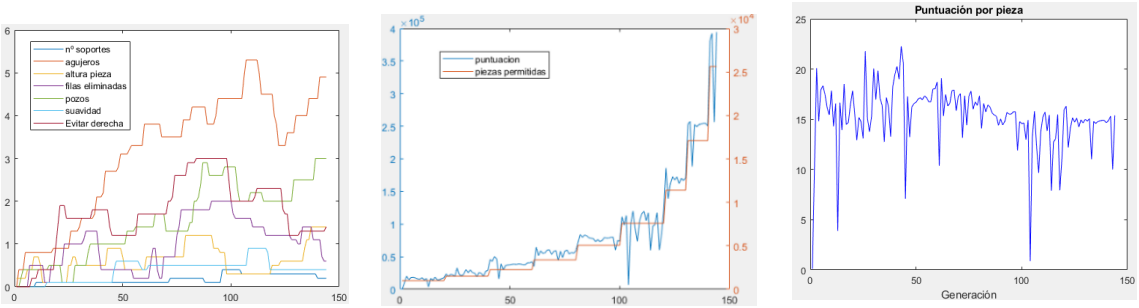


Figura 9. Gráficas evolución segundo entrenamiento de competición

A continuación, se muestran las ponderaciones resultantes de los dos entrenamientos, se observa que se le da poca importancia a las propiedades support y smooth, media importancia a la propiedades de height, y rights y mucha importancia a las de holes, rows y Wells.

	Support	Holes	Height	Rows	Wells	Smooth	Right	Tetris
Ent. 1	0.5	5.5	1.0	3.1	1.3	0.8	1.1	100.0
Ent. 2	0.2	4.9	1.4	0.6	3.0	0.4	1.4	100.0

Tabla 2. Ponderaciones entrenamientos de competición

6.2. Resultados del entrenamiento por gradiente

La IA se ha entrenado con este algoritmo varias veces, a continuación, se muestra el entrenamiento mas extenso que se ha realizado. Las ponderaciones también se han inicializado a cero, excepto la de Tetris que se ha dejado a 100 y no se ha entrenado. Se ha permitido inicialmente partidas de hasta 1000 piezas y cada 70 generaciones se ha incrementado el numero de piezas máximas. Finalmente, se ha parado el entrenamiento en la generación 380, donde se ha permitido un número máximo de piezas de 11390 y se ha conseguido una puntuación máxima de 199860 puntos. La media de puntos por medias también se ha quedado en 15.

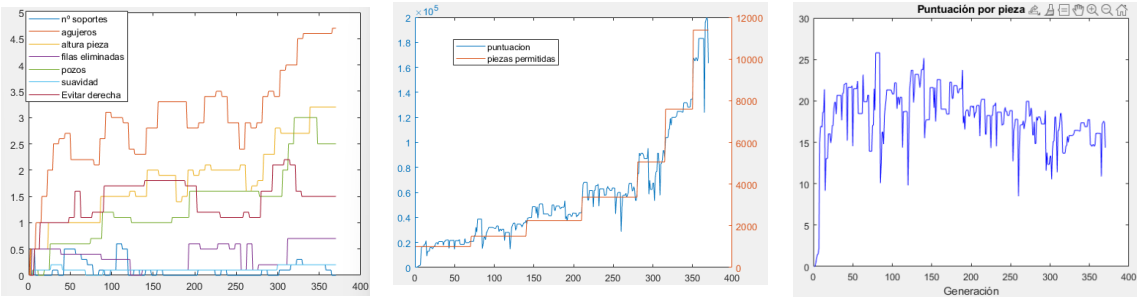


Figura 10. Gráficas evolución entrenamiento por gradiente

A continuación, se muestran las ponderaciones resultantes del entrenamiento

Supports	Holes	Height	Rows	Wells	Smooth	Right	Tetris
0.0	4.7	3.2	0.7	2.5	0.2	1.5	100.0

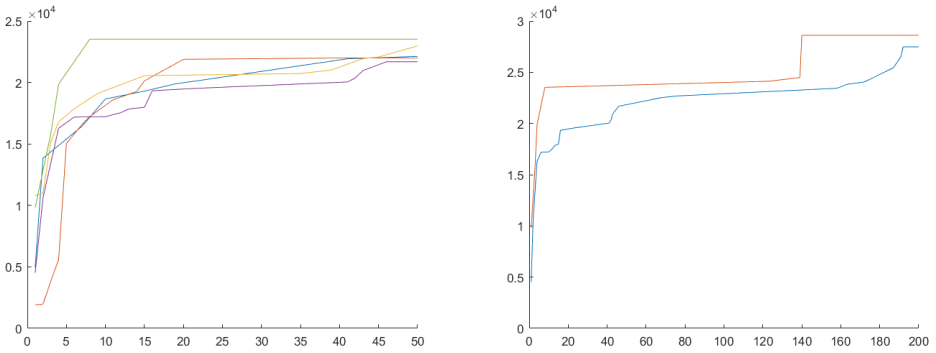
Tabla 3. Ponderaciones entrenamiento por gradiente

Se observa que los valores son similares a los del entrenamiento anterior, se le ha terminado dando más importancia a heights y menos a row.

6.3. Resultados del Algoritmo Genético

Dado que se trata de un algoritmo en el que intervienen procesos probabilísticos, para demostrar su robustez y buenos resultados es necesario ejecutar un gran número de simulaciones. Sin embargo, para el caso de este trabajo, al ser el tiempo más limitado, solo se han podido realizar dos simulaciones de 200 generaciones de longitud, y cinco de 50 generaciones (contando con las primeras 50 de las dos anteriores).

Así, fijándose únicamente en la puntuación máxima alcanzada por el algoritmo a lo largo de las generaciones, se tienen los resultados de la Figura 11.



a) *b)*

Figura 11. Evolución de la puntuación máxima del algoritmo a) para 50 generaciones y b) para 200

En la figura se puede observar a simple vista como ambas simulaciones tienen la forma de la evolución de la puntuación de un algoritmo evolutivo típico, llamada *"Anytime behaviour"*. Más en concreto se puede ver cómo una vez los individuos han alcanzado cierto nivel, es difícil encontrar mejores soluciones, lo que provoca ese "aplanamiento" a medida que las generaciones aumentan. Aun así, se puede ver como para las simulaciones de 200 generaciones, la puntuación alcanzada es significativamente mayor.

Al ser varias simulaciones, mostrar la evolución de las ponderaciones con el paso de generaciones se vuelve confuso, por lo que en la Tabla 4, a continuación, se muestran únicamente las ponderaciones de la mejor solución encontrada por el algoritmo genético en las cinco simulaciones de 50 generaciones.

	Supports	**Holes**	**Height**	**Rows**	**Wells**	**Smooth**	**Right**	**Tetris**
Sim 1	0.0	4.9	3.6	4.67	1.7	0.0	3.2	100.0
Sim 2	0.5	4.0	0.4	2.9	0.5	0.4	1.5	100.0
Sim 3	0.1	6.6	1.0	2.1	1.7	0.3	2.4	100.0
Sim 4	0.0	5.3	4.2	3.8	1.8	0.0	2.9	100.0
Sim 5	0.3	4.6	2.4	4.2	0.2	0.7	4.8	100.0

Tabla 4. Ponderaciones de los mejores individuos de las simulaciones de 50 generaciones.

De la misma forma, en la Tabla 5 se muestran las ponderaciones de los mejores individuos del algoritmo para las simulaciones de 200 generaciones.

	Supports	**Holes**	**Height**	**Rows**	**Wells**	**Smooth**	**Right**	**Tetris**
Sim 1	0.0	6.9	3.5	1.3	2.4	0.1	4.2	100.0
Sim 2	0.8	7.9	2.8	1.6	1.1	0.7	3.9	100.0

Tabla 5. Ponderaciones de los mejores individuos de las simulaciones de 200 generaciones.

Comparando los resultados de estas dos tablas con las ponderaciones máximas obtenidas mediante el entrenamiento por competición y por gradiente, se puede ver que la solución obtenida tiene la misma idea: aumentar considerablemente la ponderación para evitar dejar agujeros y dejar muy bajas algunas como la de soportes y suavidad.

Nótese que no se puede realizar una comparación directa con los algoritmos por entrenamiento de competición y por gradiente (*Figuras 8, 9 y 10*), debido a que en estos últimos se eleva sucesivamente el límite máximo de piezas por ronda.

Para poder realizar esta comparación, en la Figura 12, a continuación, se muestran los valores de la puntuación máxima por pieza del algoritmo genético. En la figura se ve como este valor aumenta muy rápidamente al principio, alcanzando al final de 200 generaciones un valor de 24.7 puntos por pieza.

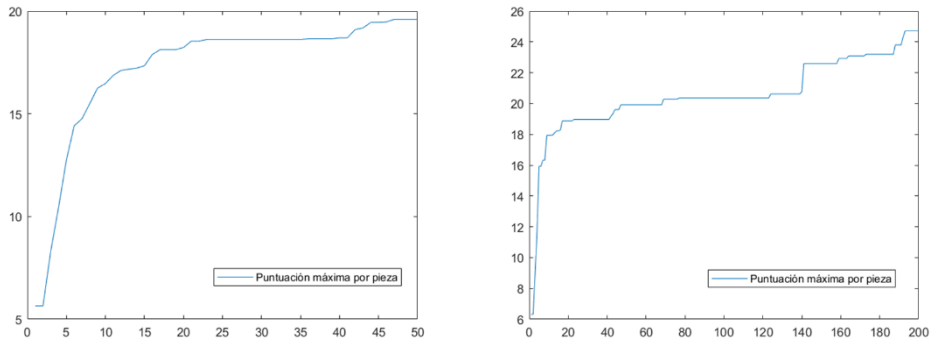


Figura 12. Evolución de la puntuación máxima por pieza del algoritmo genético a) para 50 generaciones y b) para 200

7. Posibles Mejoras

7.1. Cambios en el método de entrenamiento

Hasta el momento el entrenamiento por competiciones busca dejar a cada partida jugar hasta que hayan caído un número determinado de piezas, cuyo número irá incrementado con el número de generaciones lentamente.

Una de las posibles mejoras a la hora de entrenar la inteligencia artificial podría ser que el aumento de las piezas permitidas se realice cada menos generaciones, lo que agilizaría la evolución del entrenamiento hacia un caso más real, pudiendo ver cómo evolucionan los parámetros al buscar poco a poco la supervivencia. Por otro lado, en contraposición a este, se podría implementar otro nuevo modelo en el que cada nueva generación no tuviese límite de piezas. Este método permitirá entrenar la supervivencia del juego totalmente en vez de maximizar la puntuación en periodos cortos, lo que abriría un nuevo dilema en cuanto a qué es lo que debe buscar realmente como mejor método de juego (maximización de puntos con muerte relativamente rápida o máxima supervivencia con ganancia de puntos moderada).

7.2. Adición y eliminación de parámetros

Además de los parámetros ponderados en el algoritmo evolutivo, a esta lista se le podrían añadir un mayor número de parámetros que ayuden a mejorar las puntuaciones o el método de juego y buscar otro tipo de características en el juego para incentivar acciones que se consideren que puedan mejorar el rendimiento del método de juego. Por otro lado, sería necesario la eliminación de ciertos parámetros, los cuales, tras haber sido entrenados, su valor tiende a cero y se autoconsideran innecesarios. Esto además permitiría aumentar el tiempo de reacción del algoritmo.

7.3. Dotación de previsión al algoritmo

Una de las mejoras más interesantes para este proyecto es la dotación de previsión al algoritmo, mediante el cual las decisiones tomadas por el juego no se tomarían solamente en cuantos a qué movimiento conviene en el presente, si no que se buscaría siempre realizar un movimiento en el presente que me facilite y optimice la puntuación en el próximo movimiento. Para ello, se trató de realizar una mejora en el algoritmo de decisión para implementar esto, pero debido al tiempo restante, este algoritmo se tuvo que descartar del código en el último momento (*punto 7.3.1*).

7.3.1. Método de ajuste condicionado

Debido a la ineficiencia del método anteriormente propuesto, se propuso un método similar al anterior que se basa únicamente en evaluar la pieza actual en juego y la siguiente, considerando los valores de ponderación de los parámetros originales. Este proceso busca realizar movimientos más inteligentes que los existentes hasta ahora, ya

que permite analizar cómo beneficia la colocación inicial de una pieza a la siguiente, proporcionando al código una capacidad predictiva.

El proceso de predicción consta de varios pasos:

- Se obtienen las cinco mejores puntuaciones que la primera pieza podría alcanzar en la pantalla actual, considerando sus distintas posiciones y rotaciones.
- Con esos cinco conjuntos de datos, se evalúa la mejor puntuación que la siguiente pieza podría lograr basándose en el estado actualizado proporcionado por cada configuración de la primera pieza. Es decir, se registra la mejor puntuación y posición que la pieza 2 podría obtener para cada una de las mejores posiciones de la pieza 1. Esto genera un nuevo conjunto de datos que contiene cinco configuraciones asociadas a cada movimiento de la figura 1 (sin relación entre sí). De estos nuevos conjuntos, se selecciona el de mayor valor (*Figura 9*).

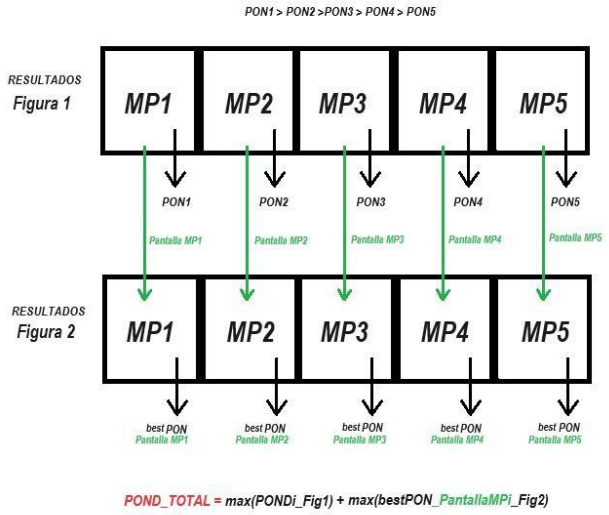


Figura 13.1. Proceso de evaluación de puntuaciones condicionadas [MP = Mejor Puntuación]

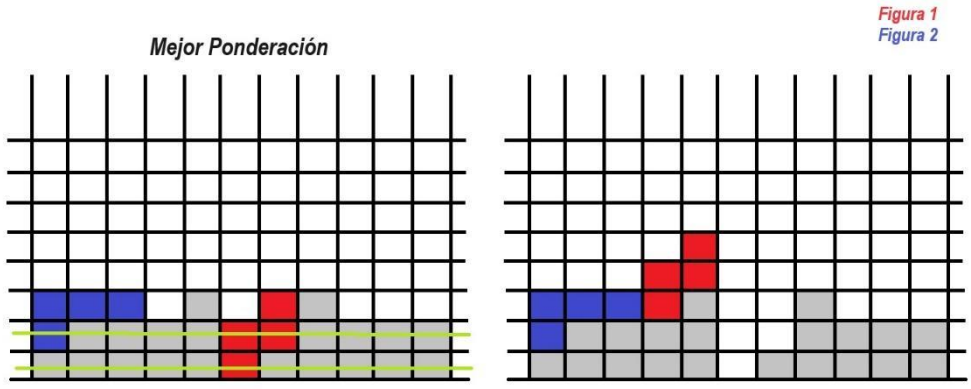


Figura 9.2. Ejemplo posibles ajustes mejores y peores para un mismo mapa

- Se sigue el hilo hacia atrás para determinar la posición y rotación que tenía la primera figura en el momento en que se logró la mejor combinación entre la figura 1 y la figura 2, como se calculó en el paso anterior. Este proceso se repite dos

veces para cada figura potencialmente en juego: la figura actual y la figura guardada (*Figura 10*).

- Se realiza una evaluación para determinar la mejor puntuación considerando las distintas posibilidades para cada figura:

    - En el caso de la figura actual → *¿La Figura 1 es la figura actual y la Figura 2 es la figura guardada, o la Figura 1 es la figura actual y la Figura 2 es la siguiente en la secuencia?*
    - En el caso de la figura guardada → *¿La Figura 1 es la figura guardada y la Figura 2 es la siguiente en la secuencia, o la Figura 1 es la figura guardada y la Figura 2 es la figura actual que fue guardada previamente?*

Figura 14. Posibles soluciones para cada ajuste

- Luego de evaluar estas opciones, se determina cuál es la mejor estrategia a seguir, si es más favorable utilizar la pieza guardada o la pieza actual. Esta decisión implica seleccionar los valores de posición y rotación de la opción que ha obtenido la puntuación total más alta, los cuales serán utilizados para representar gráficamente el movimiento.

Este método permite anticipar las mejores configuraciones de las piezas consecutivas basándose en las evaluaciones de las piezas actuales, lo que proporciona una estrategia más avanzada para la toma de decisiones durante el juego.

7.4. Algoritmo genético

El algoritmo genético tiene un gran número de posibles mejoras, mencionadas también en el punto 3.2 en el que se describe su desarrollo. En general estos cambios vienen dados por la gran cantidad de parámetros fijos que tiene este tipo de algoritmos, llevando a innumerables posibilidades.

Por ello, una gran mejora sería incorporar una lógica que permita que estos parámetros sean adaptativos, como en [1], donde las probabilidades de crossover y mutación se adaptan a la situación del algoritmo. Esto permite que, tras un número determinado de generaciones, la optimización del problema (la mutación de individuos) cobre mayor importancia que la exploración (crossover), lo que aumentaría la velocidad de convergencia en los puntos a una solución mejor.

[1]

8. Conclusión

El hito principal de este proyecto ha sido al alcanzar una puntuación máxima de 1.8 millones de puntos y 36000 líneas. Este rendimiento supera considerablemente el récord mundial alcanzado por un jugador humano, que se sitúa en 1500 líneas. No obstante, se reconoce la variabilidad inherente al juego debido a la aleatoriedad de las piezas. Para tartar con esto, se trató implementar el método de ajuste condicionado (*punto 7.3.1*), mediante el cual se podría optimizar el algoritmo actual para prever y adaptarse a la aleatoriedad a la hora de las piezas que van surgiendo y conseguir resultados más uniformes.

En el caso de los algoritmos de entrenamiento, por un lado, se comparan el entrenamiento por competición y el entrenamiento por gradiente. El entrenamiento por competición genera mayor número de cambios en los valores de los parámetros, lo que resulta en una evolución más oscilante. Por otro lado, el entrenamiento por gradiente realiza ajustes más precisos, aunque con una evolución más uniforme que requiere más tiempo, ya que cada cambio en los valores de los parámetros necesita de un volumen mayor de pruebas.

En el caso del algoritmo genético, se destaca la capacidad de obtención de un mayor de puntos por pieza colocada (22 puntos/pieza) mientras que, en los demás entrenamientos, este parámetro cae a los 15 puntos/pieza de media (estos últimos destacan por tener una evolución de puntos/pieza decreciente en función del aumento de generaciones en el entrenamiento). Además, el algoritmo genético tiene una alta velocidad evolutiva, obteniendo puntuaciones altas en relativamente pocas generaciones en comparación con los demás algoritmos. Sin embargo, a pesar de esto, existe un cierto momento en el que dicha evolución se estanca, ya que las variaciones en los parámetros son ínfimas.

Otra ventaja del algoritmo genético es que, al ser método estocástico, puede escapar de mínimos locales, no como los demás métodos desarrollados. Los entrenamientos por competición y por gradiente tienen posibilidad de caer en mínimos locales en función de sus parámetros iniciales, aunque gracias al aumento y disminución aleatoria del valor de los parámetros se consigue sortear algunos de estos mínimos locales.

Por otro lado, visualizando la evolución de los parámetros en las simulaciones de los distintos entrenamientos, se observa que algunos resultan innecesarios, como el parámetro de "supporters" y “smoothness”. Estos parámetros tienden a cero en las simulaciones, siendo anulados generativamente por los distintos algoritmos. Además, se ha observado que, aunque la cantidad de parámetros puede hacer conseguir un algoritmo más completo, esto aumenta el coste computacional y el tiempo de ejecución en casos como el entrenamiento por competición y por gradiente. La eliminación de parámetros sería muy beneficiosa sobre todo a la hora en el que el que el algoritmo está a punto de perder, ya que dispondría de mayor tiempo para pensar y ganar en tiempo a la simulación. Sin embargo, esto no es problema en el caso del algoritmo genético ya que su aumento no implica aumento en el tiempo de simulación.

Como punto final, y tras el análisis completo del funcionamiento de los distintos algoritmos, se propone la implementación de un código que mezcla varios algoritmos. Esta propuesta se basa en utilizar inicialmente la velocidad evolutiva del algoritmo genético junto a la buena capacidad de evolución del algoritmo de entrenamiento de gradiente, el cual se activaría en el momento en el que se detectase que el algoritmo genético estuviese llegando a su punto de estancamiento evolutivo.

Asignación de roles

Roles	Asignación
Toma decisiones IA	Jorge y Tomás
Algoritmos Competición y por Gradiente	Jorge
Algoritmo Genético	David
Programación interfaces	Jorge
Documentación	David, Jorge y Tomás

Bibliografía

[1] L. X. S. Han, «An Adaptive Genetic Algorithm,» *SHS Web of Conferences*, vol. 01044, p. 140, 2022.