Creación y entrenamiento de una I.A. para jugar al juego Tetris

por
Marc Pérez de Tudela Aznar

Desarrollo de aplicaciones
multiplataforma
IES Puig Castellar
31 de mayo de 2019

Tutores:
Gerard Falcó Pérez
Daniel Martínez Cruz

Índice

Introducción

En este trabajo se tratarán una serie de conceptos relacionados con la inteligencia artificial, como son: las redes neuronales, algoritmos genéticos o como abordar problemas difíciles de computar.

Para mostrar y analizar todos esos conceptos se llevará a cabo una demostración práctica de como enfrentar un problema complejo utilizando la inteligencia artificial. Más concretamente, en este trabajo se prepará el planteamiento de un problema (en este caso el conocido juego Tetris) y posteriormente se procederá a intentar crear una inteligencia artificial capaz de afrontar el problema de la forma más eficiente y eficaz posible.

En primer lugar se analizará de forma minuciosa el juego Tetris para poder replicarlo correctamente a la hora de plantear el sistema. Una vez estudiado y desplegado el ambiente para el problema se procederá a crear funciones para la recogida de datos de la partida en juego y las partidas pasadas, datos necesarios para el posterior entrenamiento y perfeccionamiento de la IA. A continuación se empezará a crear la inteligencia artificial y todos los métodos y mecanismos que permitirán no solo entrenar la I.A con los datos recogidos, sino también recoger datos del proceso para poder ver la evolución de la I.A., tanto de su nivel de juego como de su curva de aprendizaje y cambios con el paso del tiempo.

Para desarrollar este proyecto se utilizará el lenguaje de programación $Python$ junto con algunas librerías de dicho lenguaje. Los motivos de la elección de dicho lenguaje son diversos, en primer lugar $Python$ es el lenguaje más utilizado actualmente para temas relacionados con la inteligencia artificial y machine learning, por lo tanto será más fácil recavar información de proyectos similares para poder aplicarlo en este, además existen multitud de librerías que simplificarán el trabajo y permitirán centrarse en la parte importante a tratar, finalmente $Python$ es muy parecido al lenguaje humano (Inglés) y fácilmente entendible por lo tanto su curva de aprendizaje es menos pronunciada que otros lenguajes.

Finalmente hay que aclarar que en este trabajo se realizará una demostración de como abordar un problema computacionalmente complejo con técnicas de IA y machine learning, en este caso concreto el juego Tetris, pero dichas técnicas son extrapolables para abordar cualquier problema de una naturaleza similar. Por lo tanto se podrían extraer las conclusiones y procedimientos de este trabajo para realizar proyectos similares pero aplicados a otros ámbitos.

1. Análisis del problema: El juego Tetris

En esta sección se llevará a cabo un análisis tan profundo como sea posible del problema a afrontar, en este caso el conocido juego geométrico Tetris. Se recorrerán todas sus características para obtener tantos datos sobre él como sea posible, por dos razones: la primera para poder programar correctamente su implementación en $Python$, y la segunda para conocer de forma precisa y correcta los pormenores del problema a abordar para así poder crear una IA que lo afronte de la forma más correcta y eficiente posible.

1.1. Los origenes

Tetris fue creado en 1985 por 2 ingenieros rusos llamados Alexey Pajitnov y Dimitry Pavlovsky los cuales trabajaban en el centro de computación de la academia de las ciencias rusas. A su vez tomaron la idea de un juego practicado en la antigua Grecia llamado Pentaminos. En 1985 Alexey Pajitnov y Dimitry Pavlovsky programaron la primera versión de este famoso juego.

1.2. Funcionamiento

El funcionamiento de Tetris es el siguiente:

Se parte de un tablero de juego inicial, el cual consiste en una matriz de $10x20$ celdas, es decir un total de 200 celdas.

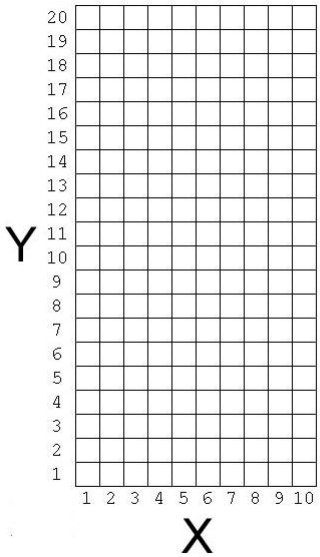

Figura 1: Diagrama del tablero donde se desarrolla el juego

Desde el momento que el juego se inicia, se empiezan a generar tetrominos, generalmente conocidos como piezas. Dichas piezas se generan en la parte superior del tablero y caen hacia la parte inferior, también pueden moverse de izquierda a derecha y rotar sobre si mismas. El jugador obtiene puntos en función del tiempo que consiga seguir jugando y el objetivo del juego es conseguir la mayor cantidad de puntos posibles.

La partida termina cuando el tablero está tan lleno que no puede aparecer una nueva pieza, entonces el jugador pierde y se registra su puntuación. Para poder vaciar el tablero el jugador debe de rellenar las filas con cuadrados, los cuadrados son las partes que conforman las piezas, cada vez que se consigue rellenar una fila entera de cuadrados dicha fila desaparece. Con el paso del tiempo el jugador avanza de nivel, lo que ocasiona que las piezas caigan cada vez más rápido, por lo tanto la dificultad se incrementara con el paso de los niveles, hay que aclarar que este incremento no es perpetuo y a partir del nivel 10 no se incrementa más la velocidad de caída.

La fórmula para calcular la puntuación por segundo es:

$$P_{segundo} = k \cdot N$$

Donde k es la constante de puntos por nivel y N el multiplicador asociado al nivel actual.

La formula para calcular el nivel actual es simplemente:

$$N = \frac{P}{10} + 1$$

Donde P es la puntuación actual. El valor máximo que puede alcanzar N será 30.

También existe una formula para calcular el incremento de la velocidad de las piezas en función del nivel, en este caso :

$$V = 0,27 - (N \cdot 0,01)$$

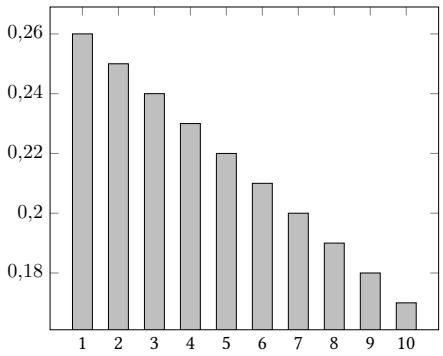

Figura 2: La velocidad de caída en relación con el nivel.

1.3. Las piezas y su creación

Las piezas en Tetris están conformadas por un total de 4 cuadrados organizados de diferentes formas, en total existen un total de 7 piezas las cuales dadas sus formas se pueden representar con letras (O, I, S, Z, L, J, T) y pueden tomar distintas orientaciones, además estas piezas poseen un punto central, el cual deberá coincidir en el momento de su aparición en el tablero con el punto $(6, 20)$ de dicho tablero.

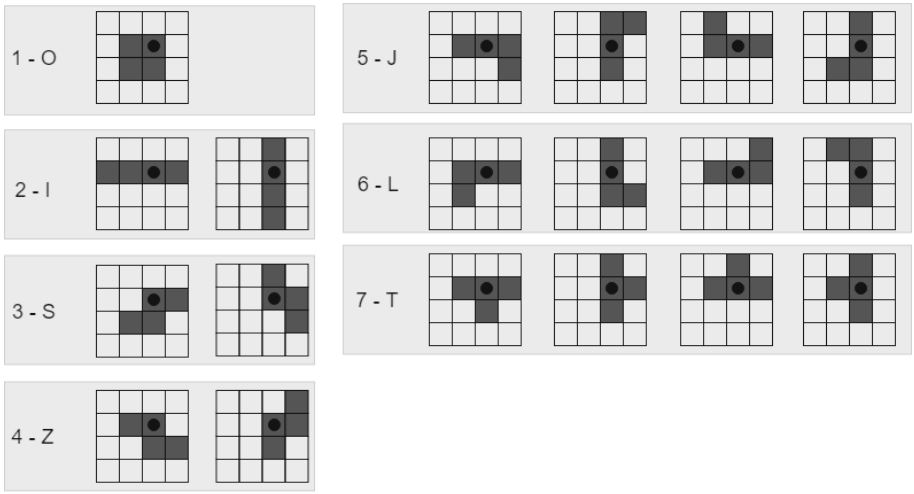

Figura 3: Las piezas de Tetris y sus orientaciones junto a su punto central.

Cada vez que se genera una nueva pieza se hace utilizando la siguiente formula:

$$P_{Indice} = (n \mod 7) + 1$$

Donde n es un numero entero aleatorio.

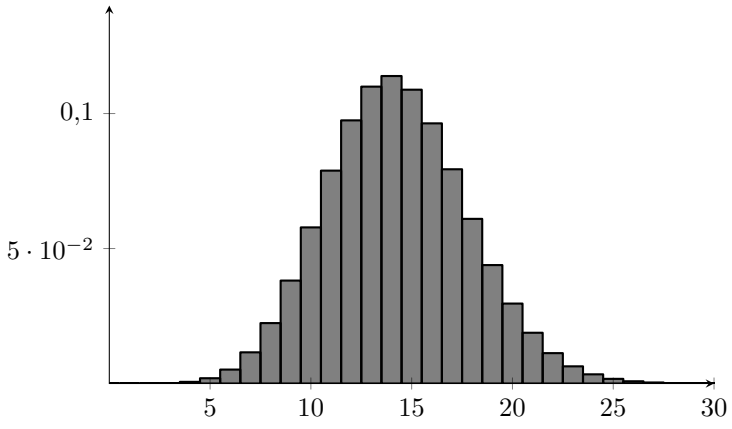

Figura 4: Distribución binominal para n=100, p=(1/7)

En algunas versiones de este juego la generación de las piezas es ligeramente distinta, para evitar la desviación estadística estándar. En estas versiones al empezar el juego se crea un array de 7 posiciones el cual contiene 7 piezas distintas, entonces cada vez que se genera una pieza se selecciona aleatoriamente de ese array, al vaciarse se rellena de nuevo con 7 piezas distintas, este método garantiza una distribución uniforme de los tipos de piezas a lo largo del tiempo.

Para ejemplificar este concepto realizaremos una pequeña simulación con el código siguiente:

Listing 1: Generador de piezas

```
import random

def prob_simulation(rolls):
    counter = 0
    for i in range(rolls):
        number = random.randint(1,7)
        if number == 1:
            counter += 1
    return counter
```

Después de numerosas ejecuciones del programa obtenemos los siguientes resultados:

Piezas creadas	Esperado	Resultado	Varianza	Desviación estándar
7	1	2	1	1
70	10	11	1	1
700	100	98	2	1,4142
7000	1000	986	14	3,7416
70000	10000	10046	46	6,7823

Al observar los resultados podemos comprobar que a mayor numero de piezas generadas mayor variabilidad y desviación estándar obtenemos, lo que podría generar a la larga sucesiones de determinadas piezas que podrían dificultar o incluso imposibilitar a la I.A. seguir jugando. Por ejemplo en una publicación creada por Heidi Burgiel llamada '*How to Lose at Tetris*' demuestra que es imposible mantenerse jugando después de una secuencia de 70.000 'S' y 'Z' alternadas. La referencia [5] de la bibliografía enlaza con un resumen con imágenes y gráficos del articulo de Burgiel.

Pero dado que este fenómeno será raramente encontrado durante nuestro trabajo y solo tendría relevancia en partidas extremadamente largas, además de ser altamente improbable, en nuestra aplicación optaremos por el método clásico de generación de piezas.

Por último trataremos el tema de la aparición de las piezas en el tablero.

Respecto a su orientación inicial, cada pieza tiene como orientación por defecto al aparecer la que se muestra en la primera columna de cada pieza en la Figura 3.

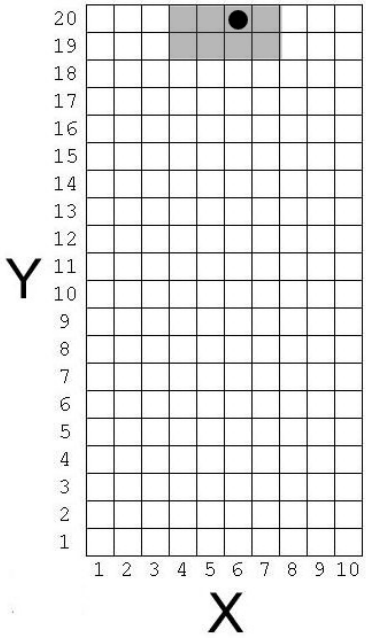

Figura 5: Diagrama de la zona de aparición de las piezas.

En el diagrama superior podemos observar como se ha mencionado anteriormente en esta sección que la pieza aparece en la parte superior del tablero, en la zona ensombrecida de un tamaño de 4x2 haciendo coincidir el punto central de la pieza sobre el punto (6,20) del tablero, dada la orientación inicial de todas las piezas incluso la figura 'I' puede aparecer correctamente en el espacio de 4x2, si la pieza generada aleatoriamente no pudiese aparecer en dicha zona el juego se daría por terminado y el jugador perdería.

1.4. Interacción con el jugador

Tetris es un juego bastante simple en este aspecto, las opciones que tiene el jugador para comunicarse con el programa para ejecutar movimientos son las siguientes:

- **Mover derecha**: mueve la pieza una columna hacia la derecha.
- **Mover izquierda**: mueve la pieza una columna hacia la izquierda.
- **Rotar derecha**: rota la pieza en el sentido de las agujas del reloj.

- **Rotar izquierda**: rota la pieza en el sentido contrario al de las agujas del reloj.
- **Soltar**: deja caer la pieza de forma inmediata.

Estos son todos los controles necesarios para jugar a Tetris, normalmente los 4 primeros van relacionados con las flechas direccionales del teclado, mientras que 'soltar' suele relacionarse con la tecla 'espacio'.

1.5. Estrategia de juego

En Tetris al igual que cualquier otro juego se puede definir una estrategia a seguir para tratar de optimizar el rendimiento, esta estrategia estará destinada a optimizar algún o algunos parámetros del juego que deseemos, en el caso que nos ocupa, la puntuación. Existen modalidades de Tetris donde al eliminar varias filas con una sola pieza obtenemos más puntos de lo habitual, también existen modalidades donde si eliminamos varias filas seguidas con piezas consecutivas obtenemos más puntos, pero en esta versión estándar de Tetris solo obtenemos puntos al pasar el tiempo, por lo tanto nuestra estrategia se orientará a mantenerse jugando el mayor tiempo posible para acumular todos los puntos posibles, ya que dada la formula de obtención de puntos que analizamos anteriormente, la cantidad de puntos que obtenemos por unidad de tiempo consiste en la constante del nivel en el que nos encontramos y la forma de aumentar de nivel consiste en acumular más puntos.

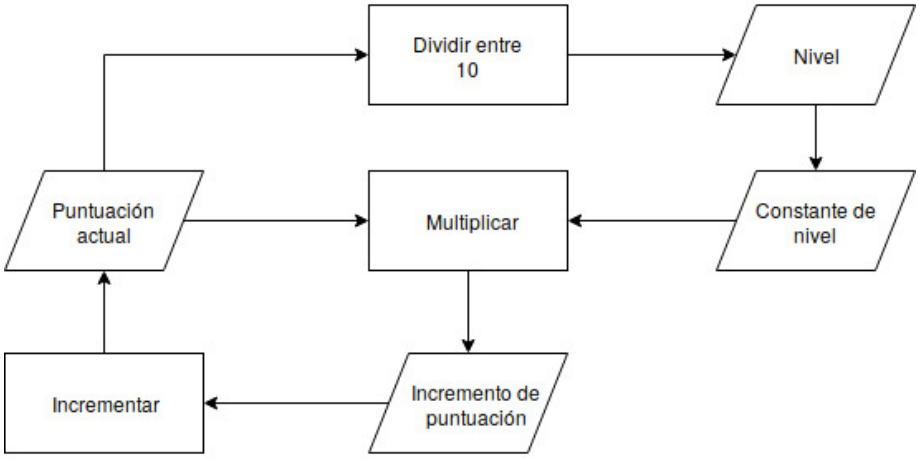

Figura 6: Diagrama del progreso de la puntuación y el nivel.

Por lo tanto podemos deducir fácilmente que nuestra estrategia para maximizar la puntuación consistente en mantenerse jugando será también la más eficaz, ya que no existe otra forma de acumular puntos.

Para analizar Tetris correctamente, al igual que en muchos juegos similares debemos aplica la teoría de juegos combinatorios (no confundir con la teoría de juegos), esta teoría es extensa y no se aplicará en profundidad, pero hay tres propiedades importantes que se deben considerar:

- **Complejidad de estado**: esta propiedad hace referencia a todos los posibles estados del juego (en este caso de su tablero), para calcular esto debemos hacerlo de la siguiente forma:
  - Tetris consiste en un tablero de 10x20, es decir un tablero de 10 columnas y 20 filas, por lo tanto el número máximo de posibles estados es de 2^{200}
  - Cada fila puede tomar 2^{10} valores, si se elimina el estado donde la fila está totalmente vacía o totalmente llena y por lo tanto desaparece obtenemos un total de 1022 posibles estados por cada fila.
  - Si una fila está totalmente vacía las superiores también lo están.
  - La estimación final de los posibles estados del tablero es de:
    $$(1/2) \cdot 9(1022^{19} \cdot 91023) = 0{,}9625 \cdot 92^{199}$$
- **El arbol de juego**: esta propiedad se define como la cantidad de posibles jugadas diferentes que se pueden realizar en cada situación. Para crear dicho arbol en Tetris debemos considerar que:
  - En Tetris existen un total de 7 piezas con sus correspondientes rotaciones, si consideramos cada pieza junto con su rotación como una pieza diferente obtenemos un total de 19 piezas distintas.
  - El tablero tiene un total de 10 columnas donde se puede ubicar cada pieza.
  - La pieza 'O' no tiene ninguna rotación posible y como ocupa 2 columnas solo podrá ser ubicado en 9 columnas distintas, la 'I' tiene solo una rotación por lo cual en vertical podrá ser puesta en 10 columnas distintas pero en horizontal solo en 7, el resto de las 16 piezas en orientación vertical solo se podrán poner en 9 columnas, y en horizontal solo en 8.
  - Por lo tanto obtenemos que el árbol de juego de Tetris se define como:
    $$n = (9 + (10 + 7) + 8 \cdot 9 + 8 \cdot 8)^N = 162^N$$
    Donde n se define como el numero de nodos por nivel y N el nivel donde nos encontramos.

Nivel (N)	nª nodos (n)
1	162
2	26244
3	4251528

Figura 7: Progresión aritmética de los nodos

Como resultado de aplicar la formula anterior obtenemos el siguiente árbol de juego:

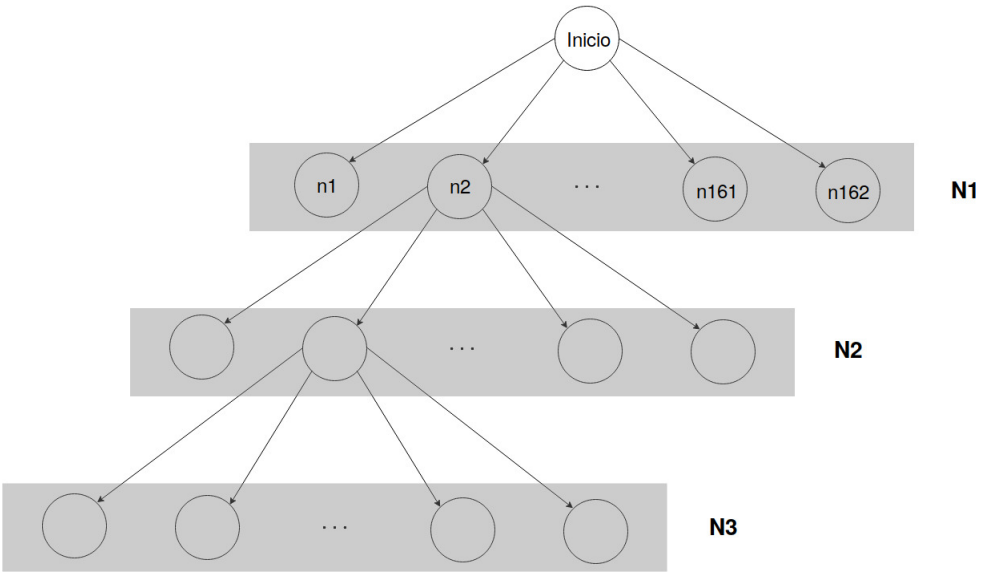

Figura 8: Árbol de juego de Tetris, expandiendo un nodo a cada nivel.

- **El juego perfecto**: esta propiedad se define como la capacidad de jugar a un juego de la forma más eficiente posible para maximizar la puntuación u obtener siempre la victoria. Para conseguir este objetivo es necesario conocer en cada jugada el movimiento a realizar, por ejemplo en el caso de Tetris donde poner cada pieza creada, si se encontrase una forma de poner las piezas creadas de tal forma que se pudiese garantizar un juego de tiempo ilimitado y por lo tanto puntuación ilimitada podría decirse que se ha encontrado el juego perfecto para Tetris o en otras palabras que Tetris ha sido resuelto, hay juegos conocidos para los cuales ya existe juego perfecto conocido, como podrían ser el 3 en raya o las damas, pero sin embargo no existe aún juego perfecto conocido para Tetris.

Si se toman en cuenta todos estos factores se puede llegar a una conclusión: Tetris es un problema complejo, no existe formula conocida para resolverlo y los intentos para resolverlo por fuerza bruta están destinados al fracaso dado el enorme volumen de posibles variantes, por lo tanto ya que ninguna de esas opciones funcionaría deberemos crear un algoritmo que tome algunos parámetros del juego que conocemos por experiencia que nos ayudarán a optimizar nuestro juego, algunos de estos parámetros podrían ser:

- El numero de huecos que dejamos. Entendemos como hueco dejar un cuadrado vacío el cual tiene un cuadrado ocupado justo encima.
- El numero de lineas que hacemos desaparecer con cada jugada.
- La diferencia de altura de las columnas, la diferencia entre la columna más alta y la más baja.
- Vacíos en la fila, este parámetro se define como el numero de celdas vacías adyacentes en la misma fila con 2 celdas llenas.

Estos son algunos ejemplos de parámetros que la I.A podría tener en cuenta a la hora de calcular sus jugadas y asignarles un valor, detallaremos esto en profundidad en apartados posteriores. Como aclaración final mencionar que los parámetros también pueden tomar valores negativos, es decir, si el maximizar algún parámetro es beneficioso para nuestro objetivo el valor que tomará de cara a nuestra I.A será positivo y a más importante sea ese valor mayor valor numérico tomará, de igual manera si el minimizar uno de esos parámetros nos ayudará a conseguir nuestro objetivo este tendrá un valor negativo y a más perjudicial sea mayor valor numérico negativo tendrá. Estos conceptos se detallarán de forma más precisa durante el diseño y construcción de la I.A

1.6. Conclusiones

En esta sección se ha analizado el juego Tetris desde diversos ángulos desde su planteamiento inicial a su funcionamiento interno de forma detallada, incluyendo un análisis superficial de algunos de los aspectos matemáticos de Tetris relacionados con su complejidad y la teoría de juegos y la forma de afrontar Tetris como problema computacional, quedando patente que la simple fuerza de calculo no basta para afrontar el problema de forma que pueda resolverse o al menos obtener resultados que alcancen una puntuación aceptable, por lo tanto se propondrá para afrontar este problema la creación de una I.A que mediante una serie de entradas de datos y operaciones será capaz de encontrar movimientos acertados que le permitan obtener buenos resultados. Además esta I.A sería capaz de seguir jugando Tetris de forma eficiente incluso si algunos parámetros del juego cambiaran como por ejemplo las dimensiones del tablero o la forma de las piezas.

2. Machine Learning: Conceptos y definiciones.

En esta sección se introducirá el concepto de *Machine Learning* en detalle, desde una definición general hasta algunos detalles y los distintos tipos y subvariantes, además se describirán los distintos tipos de problemas y situaciones que pueden ser abordados con estas técnicas. También se analizarán las diferencias entre *Machine Learning* y *Deep Machine Learning*.

2.1. Introducción al Machine Learning

Lo que conocemos como *Machine Learning* es una ciencia que involucra programación y diversas ramas de las matemáticas tales como la estadística y el cálculo. Esta ciencia está destinada a crear programas capaces de aprender a realizar una tarea.

A diferencia de la programación tradicional donde se crea un programa con un comportamiento definitivo en *Machine Learning* se diseñan algoritmos capaces de modificar su propio comportamiento para poder realizar una tarea de forma más rápida y eficiente. Por lo tanto la mayor diferencia entre el enfoque clásico y esta nueva modalidad es la capacidad de **aprendizaje** de esta última, lo que permite la creación de una nueva generación de programas capaces de afrontar problemas de nuevas formas, en cierta parte de forma más parecida a como lo haría un humano.

Pero a pesar de esto seguimos tratando con máquinas, y con las ventajas que estas nos ofrecen, como el no cansarse, o ser capaces de procesar enormes volúmenes de información los cuales un humano sería incapaz de procesar en un tiempo razonable, lo cual permitiría nuevas posibilidades en cuanto a la creación de programas capaces de realizar tareas que no podían realizarse mediante computadoras, por falta de capacidad para crear algoritmos que las rigiesen de forma correcta en tareas complejas.

Este campo se subdivide en 3 campos, los cuales són:

- **Supervised Learning**.
- **Unsupervised Learning**.
- **Reinforcment Learning**.

En los siguientes apartados se seguirá indagando en este campo y profundizando en aspectos más concretos de su funcionamiento, así como los distintos tipos de *Machine Learning* y sus aplicaciones.

2.2. Tipos de Machine Learning

En el apartado anterior se han mencionado los 3 tipos de *Machine Learning* existentes, pero en este se delimitará el alcance de cada uno así como sus ventajas e inconvenientes. A la hora de afrontar y resolver un problema es importante saber que tipo de ellos encajará mejor y así poder elegir los algoritmos de forma óptima.

2.2.1. Supervised Learning

En el aprendizaje supervisado, los algoritmos trabajan con datos "etiquetados", por lo tanto la misión del algoritmo será la de predecir una etiqueta de salida en función de los datos de entrada.

Como ejemplos de esto se pueden encontrar el reconocer dígitos escritos a mano, predicciones meteorológicas, clasificación de correos, o predicción del crecimiento demográfico.

Para llevar a cabo este tipo de técnica se trabaja con enormes agrupaciones de datos llamadas *datasets*, estos *datasets* consisten en unos datos de entrada,(En el caso de el reconocimiento de dígitos, podría ser una imagen), con una etiqueta adjunta,(En este caso el dígito que representa la imagen). La mayor parte del *dataset* se utiliza para entrenar al algoritmo en la correcta predicción de la etiqueta de salida, finalmente se reserva una pequeña parte de este, al que se le eliminan las etiquetas de los datos para comprobar si el algoritmo es capaz de predecir el resultado.

En este tipo de casos la predicción no acostumbra a mostrarse como un valor absoluto sino como una tabla de probabilidades de las posibles respuestas, parecida a la siguiente:

Respuesta	Probabilidad(%)	Respuesta	Probabilidad(%)
A	xx.xx	N	xx.xx
B	xx.xx	O	xx.xx
C	xx.xx	P	xx.xx
D	xx.xx	Q	xx.xx
E	xx.xx	R	xx.xx
F	xx.xx	S	xx.xx
G	xx.xx	T	xx.xx
H	xx.xx	U	xx.xx
I	xx.xx	V	xx.xx
J	xx.xx	W	xx.xx
K	xx.xx	X	xx.xx
L	xx.xx	Y	xx.xx
M	xx.xx	Z	xx.xx

Figura 9: Probabilidad estadística de cada respuesta

A continuación se puede ver un diagrama de como funciona el aprendizaje supervisado:

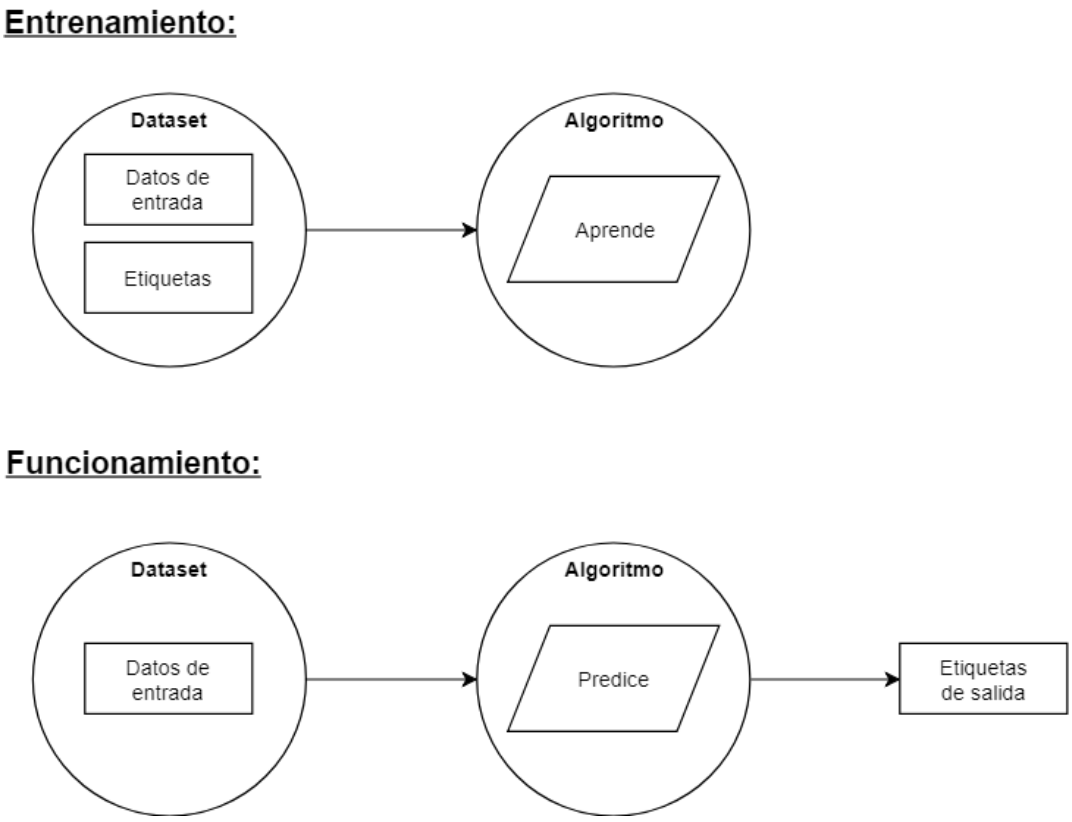

Figura 10: Diagrama del aprendizaje supervisado

2.2.2. Unsupervised Learning

El aprendizaje no supervisado es otro tipo de *Machine Learning*, en esta modalidad los datos de entrada no están asociados a una etiqueta que los identifique, en este caso el sistema intenta buscar patrones en los datos por si mismo. Esta modalidad aún no es muy popular ya que es más dificil de llevar acabo que el *supervised learning* y no tan eficaz, aunque como ventaja se obtiene que no es necesario definir como debe aprender el sistema.

Algunas aplicaciones de este estilo se encuentran en la clasificación de datos no etiquetados en *clusters* según sus características así como encontrar patrones en grandes cantidades de datos, útil en economía o estudios demográficos.

2.2.3. Reinforcment Learning

Esta modalidad de *Machine Learning* será la más analizada en este trabajo, ya que en ella se basa la posterior implementación para realizar la tarea de jugar al juego Tetris.

El aprendizaje por refuerzo consiste en enseñar a un agente a realizar las acciones correctas en un determinado ambiente para obtener los mejores resultados posibles, para eso se utiliza un sistema de recompensas y castigos para premiar al agente en función de su acción.

Este paradigma está inspirado en las técnicas que los humanos han usado a lo largo de la historia para enseñar a animales a realizar acciones o comportarse de cierta manera.

Tiene múltiples campos de aplicación, como podrían ser por ejemplo:

- En entornos industriales para regular el comportamiento de robots y en la automatización.
- *Talk bots*, programas diseñados para mantener una conversación como si de un humano se tratara.
- Enseñar a un agente a jugar a algún juego con unas normas fijas para que obtenga la mayor puntuación o el mayor rendimiento posible.

Este ultimo caso es el objeto de este trabajo y donde se mostrará una aplicación práctica de esta modalidad de *Machine Learning*.

A continuación se puede ver un diagrama del funcionamiento del aprendizaje por refuerzo.

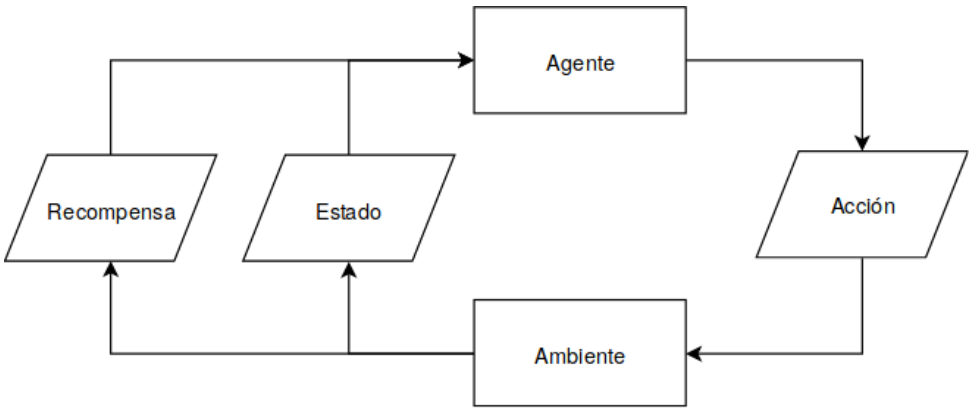

Figura 11: Diagrama del aprendizaje por refuerzo

En el diagrama pueden verse todos los elementos del proceso de *Reinforcment Learning*, los cuales se describirán y detallarán a continuación.

- **Agente**: El encargado de llevar a cabo las acciones para interactuar con el ambiente, tradicionalmente era un humano o animal, pero cuando se habla de *Machine Learning* se tratará de un algoritmo encargado de decidir la siguiente acción a realizar.
- **Acción**: Es la forma del agente para interactuar y modificar el ambiente, en el caso de Tetris las acciones son los posibles movimientos que se pueden realizar,en este caso es lo que se conoce como controles.
- **Ambiente**: Consiste en el "mundo"para el agente. Dicho de otra forma es el entorno que puede controlar o donde sus acciones tienen efecto, el objetivo del agente es modificar el ambiente o valerse de él para maximizar las recompensas. En el caso de este trabajo el ambiente sería el juego Tetris.
- **Estado**: El estado hace referencia a la situación actual del ambiente, después de cada acción realizada por el agente, el ambiente le devuelve el estado para que pueda apreciar como su acción lo ha modificado. En este caso el estado se trataría de la situación actual del tablero del juego.
- **Recompensa**: Después de cada acción realizada por el agente, el ambiente le devuelve además del estado una recompensa, la recompensa es el método que tiene el ambiente de indicarle al agente el valor de su acción, una recompensa positiva se asocia a una acción positiva, por el contrario una acción negativa será recompensada con una recompensa negativa o castigo. En Tetris la recompensa estándar para cada acción será 0 a excepción de aquellas acciones que le hagan sumar puntos, en cuyo caso la recompensa serán la cantidad de puntos ganados.

Utilizando los elementos anteriores se deberá definir un algoritmo que sea capaz de realizar la acción correcta en cada momento para maximizar las recompensas.

2.3. Deep Learning

Finalmente se hará un pequeño resumen de lo que el termino *Deep learning* significa y que lo diferencia del *machine learning*, además de porque se está popularizando este campo en los últimos tiempos.

El *Deep learning* es un subcampo del *machine learning* el cual se caracteriza por utilizar un tipo de algoritmos llamados redes neuronales para cumplir una determinada tarea. El diseño de estas redes está inspirado en el funcionamiento de los cerebros biológicos, una serie de nodos llamados neuronas interconectados entre ellos, con una capada de entrada, una de salida, y una serie de capas ocultas que añaden complejidad a la red por tal que pueda realizar tareas mas complicadas.

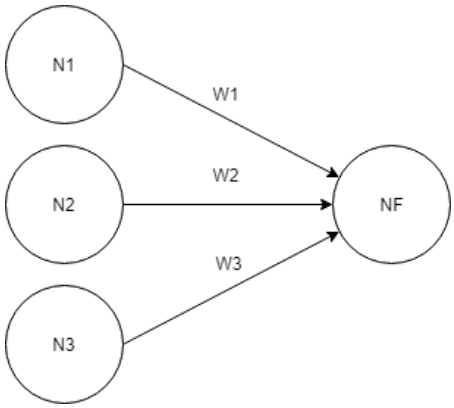

Figura 12: Diagrama de una red neuronal básica

Además cada conexión de la red lleva asociado un *weight* o peso que modificará el valor del input para dotar de distinta importancia a cada una de las relaciones.

Se ha añadido este apartado en este trabajo para tratar de dar un rápido vistazo a este tipo de algoritmos ya que los recientes logros y avances en este campo vienen de la mano de estas redes neuronales, los ejemplos más recientes de esto podrían ser por ejemplo:

- **AlphaZero**: Una red neuronal desarrollada por DeepMind para jugar al ajedrez, a pesar de la poca información que se tiene sobre ella parece un gran avance y probablemente el motor de ajedrez más podereoso que existe.
- **LeelaChess**: La versión de código abierto de AlphaZero, está siendo desarrollada siguiendo el mismo concepto que Alphazero y esta obteniendo grandes resultados contra otros motores.
- **AlphaGo**: En este caso un motor de juego para jugar al juego Go, juego muy popular en Asia. Desarrollado también por DeepMind este motor ha sido el primero de la historia en ser capaz de vencer a un campeón del mundo humano en un match.

Por lo tanto parece ser que el futuro del *machine learning* vendrá de parte de este tipo de algoritmos capaces de dedicarse a un amplio rango de tareas de forma eficaz, desde reconocimiento facial o de lenguaje natural hasta aprender a jugar a distintos tipos de juegos.

3. El agente: La inteligencia artificial

En esta sección se detallará el funcionamiento de la inteligencia artificial que se ha creado para este trabajo, analizando en los siguientes apartados cada uno de los aspectos de la I.A, esta I.A será la encargada de realizar los movimientos y evaluaciones en el ambiente de juego, y cuyo objetivo es obtener la máxima cantidad de puntos.

3.1. Funcionamiento general y tipo de algoritmo

Para este trabajo se creará un algoritmo, cuyo funcionamiento consiste en lo siguiente:

- **Obtener el estado del tablero**: El primer paso de la I.A. consiste en obtener una copia del estado actual del tablero para poder determinar su siguiente movimiento.
- **Probar todas las combinaciones**: Una vez obtenido el estado actual del tablero el algoritmo creará mesas simuladas con cada uno de los posibles movimientos posibles dados la pieza actual y el tablero. Debe entenderse como movimiento no el mover o rotar una pieza sino como ubicarla finalmente en el tablero.
- **Evaluar cada movimiento basándose en los parámetros**: Una vez se tengan listados todos los posibles movimientos el algoritmo pasará a evaluarlos, de esas evaluaciones obtendrá el estado del tablero resultante de su movimiento y la cantidad de puntos obtenidos. Para evaluar el estado de la mesa y poder establecer una estrategia a largo plazo y no basarse solamente en la puntuación obtenida por cada jugada se utilizarán lo que llamamos parámetros.
- **Realizar el movimiento**: Cuando la I.A. haya dado un valor a cada uno de los posibles movimientos elegirá el que tenga el valor más alto y procederá a realizarlo.
- **Actualizar el peso de los parámetros**: Finalmente después de realizar el movimiento se redefinirá el peso de cada parámetro. El peso se define como la importancia del valor de cada parámetro para obtener el valor final de cada movimiento. Para redefinir constantemente el peso de estos parámetros se utilizará un algoritmo común en *machine learning* llamado *gradient descent*. Este algoritmo será el responsable de dotar a la I.A de algo similar a la capacidad de aprender, pues en función del resultado de sus acciones variará los parámetros para optimizar la puntuación obtenida.

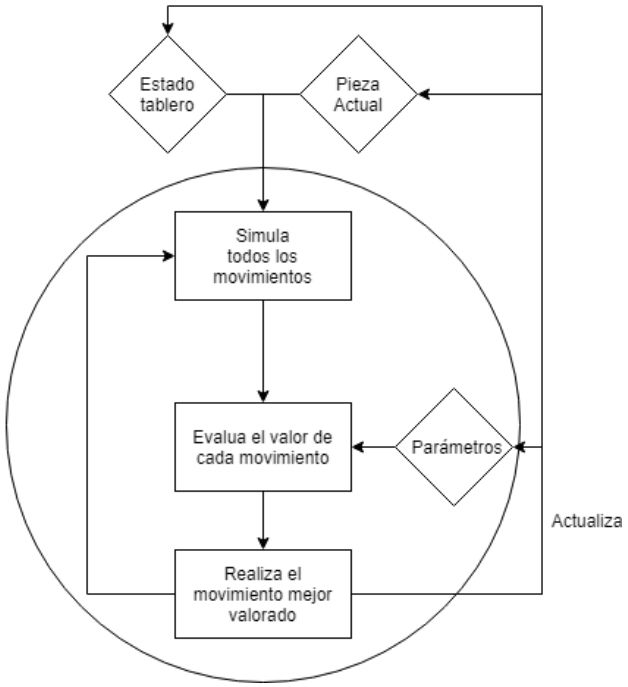

Figura 13: Diagrama del funcionamiento del algoritmo

3.2. Los parámetros

En este apartado se definirá la utilidad de estos parámetros y el porqué de su elección además de especificar que parámetros se tratarán.

Los parámetros serán aquellos atributos del tablero de juego que se tendrán en cuenta por la I.A. para definir si un determinado estado es mejor o peor que otro, otorgándole un valor numérico. Esto es necesario ya que de no ser así el algoritmo solo tendría la cantidad de puntos obtenidos cada jugada para evaluar la calidad de su movimiento, por lo tanto sería muy difícil establecer una estrategia a largo plazo la cual no solo dependiera de la recompensa inmediata sino que también fuese capaz de evitar realizar un movimiento el cual le otorgaría puntos en favor de otro que no, pero sería más beneficioso en un futuro.

Es debido a esa necesidad de no basarse solamente en la recompensa inmediata que son necesarios los parámetros, para esta I.A. se han elegido los siguientes:

- **La suma de la altura de las columnas**: El primero de nuestros parámetros sera la suma total de la altura de todas las columnas del tablero.
- **Suma de las diferencia de altura de una columna y su adyacente**: Se calculará la diferencia de altura de una columna con la siguiente para posteriormente sumar todas esas diferencias y obtener el total.
- **Altura máxima**: La altura máxima que alcanza el tablero.
- **Número de agujeros**: El numero total de agujeros, se define como agujero una casilla no ocupada la cual tiene casillas ocupadas encima suyo.
- **Transiciones en la fila**: Cada vez que una determinada fila pasa de estar ocupado a vacía o viceversa, también se pe podrían denominar irregularidades de la fila.
- **Transiciones en la columna**: Lo mismo que el parámetro anterior pero cambiando las filas por columnas.

Estos han sido los parámetros elegidos para la creación de la I.A., los motivos principales de esta elección han sido experimentos anteriores similares a este que tomaron parámetros similares, y también la experiencia empírica que ha demostrado que esos atributos tienen un impacto importante en el desarrollo del juego. Todos ellos tendrán un *weight* especifico asociado, esto permitirá atribuir a cada valor del parámetro un peso especifico para poder calcular la puntuación final de un determinado movimiento y sus consecuencias.

3.3. Hiperparámetros

Además de los parámetros vistos anteriormente existen de ellos llamados hiperparámteros. Estos son comunes en los algoritmos de *machine learning*, estos definen ciertos aspectos del comportamiento del algoritmo y pueden modificarse para obtener distintos comportamientos. En este caso se utilizarán tres de ellos:

- **Learning rate**: Tambien representado con la letra griega α, este hiperparámetro define como de rápido o lento aprende la I.A., es una forma de determinar cuanto modifica los *weights* de sus parámetros por cada movimiento que realiza.
- **Discount factor**: Tambien representado con la letra griega γ, es un factor de descuento que se aplica a la recompensa inmediata para que el algoritmo no aprenda demasiado de esa recompensa y también tome en cuenta otras acciones que podrían beneficiarle en un futuro.
- **Exploration rate**: Este es el último hiperparámetro implementado en la I.A., de este hiperparámetro dependerá la posibilidad de que el algoritmo elija un movimiento aleatorio pese a no considerarlo el mejor, esto existe para evitar el estancamiento en una acción que proporcione una recompensa mínima cuando podría haber una acción que proporcionase una aún mayor .

4. Creando la aplicación: Diseño y arquitectura

En esta sección se tratará con los aspectos más técnicos del desarrollo, se realizará un recorrido por el proceso de desarrollo, desde el planteamiento inicial y la toma de algunas decisiones iniciales pasando por la estructura del programa y partes de su implementación. También se profundizará en algunos aspectos clave del programa y finalmente se realizarán comparaciones entre diversas opciones. Durante la creación de la aplicación se dará frecuentemente el caso de poder realizar el mismo proceso o implementación de diversas maneras, cuando esto ocurra se intentará entonces elegir la opción óptima. La decisión final se tomará en función de diversos valores tales como:

- **Optimización del código:** Si una solución convierte el código en más óptimo y/o más fácilmente legible tendrá preferencia sobre una ineficiente o que complique el código.
- **Facilidad de implementación:** Se entiende como facilidad de implementación, que una determinada solución sea más simple de programar y sea capaz de ofrecer un rendimiento similar o igual que otra más complicada
- **Utilidad e implantación en el mundo real:** Este apartado hace referencia a la aplicación y utilidad de la solución en el mundo real, esto quiere decir que se optará por usar recursos, técnicas y librerías que se implementen en entornos profesionales en el mundo real.
- **Facilidad de obtener información y recursos relacionados con la tecnología:** Finalmente se priorizarán aquellas soluciones de las cuales sea más sencillo obtener información y documentación sobre su funcionamiento

Para terminar también se tratará el aspecto de como el programa almacena gestiona y organiza la información que obtenga, cosa que será fundamental a la hora de normalizar la información para que el desarrollo de la I.A pueda realizarse correctamente.

4.1. El lenguaje: Python

En este apartado se ampliará la información que se adelanta en la introducción, el lenguaje elegido para desarrollar el proyecto será Python. A continuación se enumerarán los motivos:

- **Gran comunidad:** Este lenguaje cuenta con una enorme cantidad de seguidores por lo que será más fácil encontrar información sobre los problemas que surjan durante el desarrollo.
- **Lenguaje predominante en machine learning:** Actualmente Python es el lenguaje más utilizado en el campo del machine learning, por lo tanto resultará más fácil encontrar contenido relacionado con el trabajo.
- **Buenas herramientas para crear gráficos:** Para controlar los datos del trabajo el programa creará gráficos para poder ver la evolución de los parámetros y los resultados y ayudar a la comprensión de estos.

4.2. Organización del código

Para este proyecto se ha organizado todo el código fuente en 2 partes diferenciadas, por una parte un archivo donde se concentrará toda la lógica del juego y sus métodos y funcionamiento, y otra donde residirá la implementación de nuestra I.A. Esta organización se lleva a cabo por una cuestión de orden y para una mayor facilidad para identificar errores en el código y en caso de necesidad poder modificar por separado cualquiera de los dos objetos sin que dichas modificaciones afecten al funcionamiento general de la aplicación. Además durante el transcurso de la ejecución de la aplicación se crearán, leerán y modificarán 2 archivos de texto, los cuales se utilizarán para la persistencia de datos en la aplicación, en este caso para almacenar los valores de los *weights* y el valor del *exploartion rate*, de esta forma no será necesario volver a entrenar la I.A. cada vez que se ejecute la aplicación.

4.3. Requisitos para su ejecución

Para ejecutar la aplicación solo será necesario una instalación de *Python 3.6* y algunas librerías, el ambiente de juego *Tetris* utiliza el motor de juegos *pygame* para funcionar. Finalmente aclarar que esta versión de *Tetris* es una copia de la implementación de Al Sweigart de su implementación de *Tetris* en su libro *Invent with Python*. Se tomo esta decisión porque el objetivo del trabajo no era el de desarrollar un ambiente adecuado para trabajar sinó el de crear una I.A capaz de desenvolverse en dicho ambiente, para esto parecía adecuado adaptar un ambiente ya existente con algunas modificaciones.

NOTA: Debido a un problema en el funcionamiento de la librería encargada de realizar las pulsaciones de las teclas necesarias para realizar los movimientos esta aplicación solo funciona en entornos *Windows*, más concretamente en *Windows 10*, por lo tanto no funciona debidamente en distribuciones Linux.

5. Conclusiones: Resultados, mejoras e impresiones

Este es finalmente el último apartado de este trabajo. En el se realizara una análisis de los resultados obtenidos para poder valorarlos, también se sugerirán una serie de posibles mejoras que podrían ser implementadas en un futuro, además de unas impresiones finales que resuman la utilidad de este trabajo.

5.1. Resultados

Antes de plasmar los resultados en este apartado debemos tener en cuenta una serie de consideraciones previas.

En primer lugar los hiperparámetros utilizados, los cuales son:

- **Learning rate**: 0.01
- **Discount factor**: 0.9
- **Exploration rate**: 0.5

Estos son unos hiperparámetros estándar utilizados de forma genérica. Finalmente aclarar respecto al *Exploration rate* que disminuye con el paso de los movimientos realizados.

A continuación se ha entrenado a la I.A durante un total de 100 partidas para que pueda optimizar sus *weights* y poder obtener unos resultados óptimos. Después del entrenamiento se ha procedido a realizar la prueba final, jugar 50 partidas y comprobar su rendimiento promedio.

Finalmente se ha obtenido un rendimiento promedio de **124.49** puntos por partida, por lo tanto la I.A. consigue limpiar un total de 124.49 lineas cada partida antes de perder, llegando a alcanzar picos de hasta 700 lineas limpiadas. Como se puede observar al tratarse de un algoritmo de este tipo obtiene una alta variabilidad respecto de otros algoritmos más tradicionales.

5.2. Mejoras a realizar

Como principales mejoras a este proyecto se podría sugerir el implementar otro tipo de algoritmo más complejo del credo aquí para obtener un mayor rendimiento, así como la elección de otros parámetros o añadirle algunos nuevos para observar los cambios en el comportamiento de la I.A., además de solucionar un problema que ocurre con frecuencia en esta implementación el cual consiste en que la I.A deja demasiados espacios laterales vacíos para introducir una pieza vertical y limpiar diversas filas a la vez, pero finalmente no llega a tiempo de colocar esa pieza y pierde, pese a poder reducir la altura de la mesa de forma inmediata.

5.3. Conclusiones finales

Este trabajo ha tratado de ser una introducción al mundo del *Machine Learning*, más en concreto al campo de *Reinforcment Learning*, por lo tanto después de una serie de explicaciones teóricas se ha tratado de aplicar los conceptos aprendidos en un pequeño proyecto consistente en crear una I.A. capaz de realizar una tarea medianamente compleja. Para esto se han creado un ambiente y un agente encargados de realizar ese trabajo, se plantearon enfoques más complejos que los relatados en esta memoria pero por falta de tiempo y fuentes de información fiables se ha simplificado para obtener un resultado que pese a ser menor fuese aceptable y más fácil de explicar y exponer que algo más complejo. Evidentemente los conceptos y técnicas expuestos en este trabajo pueden extrapolarse a otro tipo de tareas y problemas para crear algoritmos basados en *Machine Learning*, pues la idea detrás de estos métodos es la de crear programas capaces de aprender de su experiencia para mejorar su rendimiento en un futuro. Sin más aquí termina la exposición de este proyecto y su desarrollo.

Bibliografía

[1] *Colin Fahey. Review and history of Tetris.*
colinfahey.com/tetris/tetris.html

[2] *Wikipedia. Article about pentominos.*
en.wikipedia.org/wiki/Pentomino

[3] *Al Sweigart. Tetris made with Python.*
inventwithpython.com/pygame/chapter7.html

[4] *Heidi Burgiel. How to Lose at Tetris.*
citeseerx.ist.psu.edu

[5] *A deadly piece sequence.*
harddrop.com/wiki/A_deadly_piece_sequence

[6] *Onas Sjöstrand. Combinatorial game theory.*
www.math.kth.se/matstat/gru/sf2972/2015/gametheory.pdf

[7] *Juan Ignacio Bagnato ¿Qué es Machine Learning? Una definición*
www.aprendemachinelearning.com/que-es-machine-learning/

[8] *Definitions about machine learning*
skymind.ai/wiki/deep-reinforcement-learning

[9] *Explicación sobre el algoritmo gradient descent*
towardsdatascience.com/gradient-descent-in-a-nutshell

[10] *Proyecto similar que sirvió como ejemplo*
github.com/scuriosity/machine-learning-tetris