Documentación del código

Vamos a separar la documentación del código en 4 partes, la primera explicará el esquema de las estructuras de datos utilizada por todos los métodos, la segunda hablará del código sobre la parte que afecta a la interfaz que se muestra por consola y las primeras elecciones sobre el tipo de partida y variante de tablero, la tercera sobre la parte de jugabilidad y como se llevan a cabo la búsqueda y aplicación de movimientos y la última se explica las funciones que usa la IA para buscar movimientos.

Índice

Juan Pablo Cabezas Villalba
Pedro Luis Soto Santos
2021/2022

Estructuras de datos:

Nombre	Esquema	Descripción
Ficha	Integer	Integer, 0 si es un espacio vacío, 1 si es una ficha negra, 2 si es una ficha blanca y 3 si es el rey.
Jugador	Integer	1 que corresponde a jugador negro o 2 el cual corresponde a jugador blanco.
Movimiento	Tupla de 4 Integers: (1,2,3,4)	Los dos primeros muestran la posición actual de la ficha y los otros dos la nueva posición de esta tras aplicar el movimiento. Fila, columna origen a fila columna destino.
Tablero	Tupla de listas con fichas: ([0,0,1,1,1,0,0], [0,0,0,1,0,0,0],...)	En la que cada lista representa una fila del tablero, en ella se muestran Integer dependiendo de la ficha que contenga en esa casilla.
Estado	Tupla tablero y jugador: (Tablero, jugador)	Resume el estado actual de la partida con el tablero (y las fichas) así como el turno del jugador
Nodo	*Clase* *init(estado, nodo, Integer)* +estado: estado actual +movimientos: lista de posibles movimientos desde el estado +n: nivel +q: recompensa +i: Contador de movimiento de movimientos +indMov: índice de la lista de movimientos del padre, del cual se crea este nodo +turno: número de turno +hijos: lista de hijos desde este nodo +padre:nodo padre	Clase usada para construir el árbol de movimientos en la búsqueda del algoritmo

Miscelánea relacionada a la estructura de datos:

Nombre	Entrada	Descripción
Cambio_coordenadas	(len, i): *(int, int)* +len: longitud del tablero +i: índice a cambiar	Esta función transforma las coordenadas de movimiento a los índices de la tabla, ya que si observamos el tablero con una perspectiva cartesiana, la coordenada x no coincide con el índice. Para hacer este cambio se aplica esta función la cual usaremos al acceder a la tupla a partir de un movimiento y viceversa.

Interfaz:

Nombre	Entrada	Descripción
Imprime_tablero	(tablero):	Dado un tablero esta función lo recorre y va imprimiendo en consola su contenido, marcando además las filas y columnas.
Imprime_estado	(estado, numero_de_movimientos, numero_turnos): *(estado, Int, Int)* (variables autoexplicativas)	Esta función es la encargada de llamar a la anterior, además de añadir nueva información por pantalla, se encarga de comprobar que el estado no es final con una función que veremos, si es final muestra el ganador por pantalla, comprobando de quien es el turno actualmente, o si son tablas, si no es el final, se muestra el jugador al que le toca jugar, el número de movimientos válidos que tiene y

Nombre	Entrada	Descripción
		los turnos restantes de la partida antes de tablas.
Elige_variante	():	Esta función se aplica antes de iniciar la partida para elegir entre una de las 6 posibles variantes de tablero mediante input.
Elige_numero_turno	():	Esta función también se aplica antes de empezar para poner un límite de turnos antes de acabar la partida en tablas.
Seleccion_modo	():	Esta función de nuevo se lanza antes de empezar la partida para elegir que tipo de partida desea jugar, jugador vs IA, jugador vs jugador o IA vs IA.
Seleccion_pieza	():	Esta función se lanza si el modo elegido ha sido jugar contra la máquina y sirve para elegir con qué equipo quiere jugar el jugador.
Turno_jugador	(estado, movimientos): *(estado, array)* +movimientos: lista de posibles movimientos	Cuando es el turno de mover ficha del jugador, se imprimen los movimientos posibles de este y se llama a la función de movimiento_valido que veremos más tarde.
Turno_maquina	(estado, movimientos, numero_turnos, tiempo): *(estado,array,Integer,Integer):* +movimientos: lista de posibles movimientos +numero_turnos: número de turnos antes de acabar +tiempo: tiempo para buscar la solución	Si es un modo en el que juega la IA, cuando sea el turno de esta se usa esta función, a la cual le entran las variables que necesita el busca_solución, método que veremos más adelante el cual se encarga de buscar un movimiento óptimo a través de Montecarlo.

Nombre	Entrada	Descripción
Movimiento_valido	(estado, movimientos):	Una vez llamado por turno_jugador se pide un movimiento y esta función comprueba si es válido o no, si lo es se aplica con una función que veremos más adelante y si no es válido se vuelve a pedir de nuevo un movimiento válido.
Interfaz_usuario	():	Esta es la función más importante, es la encargada de llamar a las funciones anteriores en su debido momento e ir iterando en los turnos entre jugadores actualizando las variables en cada turno e ir comprobando que la partida no ha llegado a un final para seguir avanzando.

Jugabilidad:

Nombre	Entrada	Descripción
Estado_inicial	(variante): *(Integer):* Variante: 1 - Hnefatafl 2 - Tablut 3 - Ard Ri 4 - Brandubh 5 - Tawlbwrdd 6 - Alea Evangelii	Entrado el número de la variante que ha elegido el jugador, se crea el estado inicial del tablero y se añade el turno al jugador que tenga fichas negras, en todas las partidas empiezan moviendo las negras.
Obtiene_movimientos_peon	(tablero, jugador): *(tablero, jugador):*	Obtiene todos los movimientos posibles de los peones de este jugador.

Nombre	Entrada	Descripción
Obtiene_movimientos_rey	(tablero):	Al igual que la función anterior, solo que busca los movimientos posibles del rey, ya que este puede ocupar posiciones en las cuales no puede estar un peón.
Obtiene_movimientos_negras	(tablero):	Esta función llama a la función obtiene_movimientos_peon pasándole el jugador negro.
Obtiene_movimientos_blancas	(tablero):	Esta función llama a la función obtiene_movimientos_peon pasándole el jugador blanco, a la lista que devuelve esta función se añaden los movimientos del rey los cuales se obtienen con obtiene_movimientos_rey.
Obtiene_movimientos	(estado):	Se encarga de llamar a las funciones de obtiene_movimientos dependiendo del jugador del estado que le entra.
Ganan_negras	(estado, num_movs):	Dado el estado comprueba si ha ganado el jugador negra por que se ha comido al rey o por que el blanco quede sin movimientos válidos.
Ganan_blancas	(estado, num_movs):	Igual que la función anterior, solo que comprueba si el rey está en las esquinas, las cuales son las casillas de escape o que el negro este sin movimientos validos.
Es_estado_final	(estado, numero_de_movimientos, numero_turnos): *(estado, Integer, Integer):* (variables autoexplicativas)	Comprueba con las dos funciones anteriores y el número de turnos para tablas, si el estado de la partida es final, devuelve True si lo es, de lo contrario False.

Nombre	Entrada	Descripción
Aplica_movimiento	(estado,movimiento):	Esta función aplica al tablero del estado que le entra el movimiento que se indica, este movimiento es válido ya que esta función es llamada en movimiento_valido la cual comprueba a través de las listas de posibles movimientos si el movimiento que introduce el usuario es posible en el estado, si es asi llama a esta función para modificar el estado.

Algoritmo de búsqueda:

Nombre	Entrada	Descripción
Busca_solucion	(s0, t, numero_turnos): *(estado, Integer, Integer):* s0: Estado inicial t: tiempo de computación para encontrar solución numero_turnos: Turnos antes de tablas	Función principal que se encarga de aplicar el algoritmo de Montecarlo Upper Tree Confidence con ayuda del resto de funciones dentro de un tiempo y turnos posibles devolviendo el próximo movimiento calculado.
Tree_policy	(v): (*nodo*)	Mientras el estado actual tenga movimientos posibles expande el árbol un nivel, de lo contrario devuelve el mejor hijo.
Expand	(v): (*nodo*)	Aplica un movimiento dentro de la lista de movimientos del nodo y crea un nodo hijo con ese nuevo estado.
Best_child	(v, c): (*nodo*, *Double*)	Devuelve el mejor hijo de todos los hijos del nodo aplicando una

Nombre	Entrada	Descripción
		ponderación respecto a la recompensa y distancia.
Default_policy	(v): (*nodo*)	Comienza a realizar movimientos aleatorios hasta alcanzar un estado final. Devuelve la recompensa de dicho estado, dependiendo de si el resultado es favorable o no para el jugador.
Backup	(v, delta): (*nodo*, *Double*)	Propaga la distancia y recompensas de las hojas a los padres hasta llegar a los hijos del estado inicial para poder calcular el mejor hijo del origen.