II PROYECTO PROGRAMADO IA

Connect 4

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computación
Curso: Inteligencia Artificial
Semestre 2 - 2018
Proyecto II
Estudiantes:

- Jonathan Martínez Camacho
- Mariana Rojas Semeraro
- Katerine Molina Sánchez

Repositorio: https://github.com/ProyectoII-IA/Connect4

Estructura general del proyecto

Clases de configuración

- Fit_Agents_Controller: clase que toma los parámetros que se envían por consola, y que se encarga de instanciar y ejecutar el algoritmo genético, para posterior retornar la configuración del mejor agente.
- Game_Controller: clase que toma los parámetros que se envían por consola, y que se encarga de instanciar los dos jugadores, para posterior instanciar el juego con los jugadores creadores y permitir que se realize el proceso natural del juego.

Clases del juego

- Board: es la clase base del juego, sobre ella se define el arreglo básico para el juego, y todas las funciones referentes a validaciones de las jugadas y la detección de los casos cuando hay ganador.
- Player: es una interface la cual define el comportamiento básico de los tipos de jugadores, es implementada por las clases Agent, y Human.
- Agent: define la estructura básica de un agente, contiene las probabibilidades de las diferentes estrategías que cada agente utiliza.
- Human: define la estructura básica de un jugador humano.
- Game: define la estructura del juego, y es el encargado de controlar el flujo del mismo. Posee definidos dos jugadores y utiliza una instancia de la clase Board para administrar el estado del juego.
- Strategy: es la clase que define la base para las estrategías a implementar por parte del agente. Es una generalización que debe ser adoptada por cada nueva estrategía que se desee agregar o implementar.
- Blocking2_vs_1: define la estructura y comportamiento de la primera estrategía, su objetivo es comparar si es mejor un movimiento de bloquear una secuencia de dos fichas del oponente contra bloquear una sola ficha del mismo.

- Center_vs_Extremity: define la estructura y comportamiento de la segunda estrategía, su objetivo es comparar si es mejor un movimiento de inserción al centro del tablero, en contraposición a una inserción a los extremos del tablero.
- Sequential_vs_Space: define la estructura y comportamiento de la tercera estrategía, su objetivo es comparar si es mejor un movimiento de inserción en secuencia o una inserción dejando un espacio en blanco en medio, con el objetivo de confundir al oponente.
- Horizontal_vs_Vertical: define la estructura y comportamiento de la cuarta estrategía, su objetivo es comparar y determinar si un movimiento de inserción en horizontal es mejor que un movimiento de inseción en vertical.
- Position: clase que permite llevar el control de los diferentes filtros de estrategías que se aplican a un estado en particular del juego (board) con el único objetivo de determinar la o las columnas que ofrecen una mejor jugada. Esta clase es solo utilizada por las instancias de Agent.

Clases del algoritmo genético

- Game_Genetics: Clase genérica adaptada para el algortimo genético, con la cual se logra la competencia entre dos agentes dados. Esta clase es de poco interes debido a que solo es una adaptación de la clase "Game" en game.
- Genetics_Algorithm: Clase encargada de ejecutar el algoritmo genetico inicializando los parámetros esenciales del algoritmo además de la manipulación de la población de una generación a otra y calculo general del fitness de toda una población de individuos.
- Individual: Cada individuo de la población está definido por una instancia de esta clase, la cual, brinda los métodos de crossover entre dos individuos además de la mutación entre un individuo y como obtener el fitness.

Estrategia de solución del juego

- Configuración: el inicio del juego está determinado por la configuración que se le envie por consola al juego, en este sentido, el primer paso de la configuración es el parseo de los parámetros enviados al programa (ver el manual de usuario para comprender la forma en que los parámetros deben ser ingresados), según el valor de los mismos se determina el segundo paso de la configuración, en este caso, la instanciación de los jugadores, y por consecuente, el tercer y último paso de la configuración, la instanciación e inicio del juego como tal.

- Juego: una vez activado el flujo del juego desde la configuración, se inicia el ciclo del juego, básicamente este ciclo está determinado por dos posibles estados del tablero, el primero, es que se encuentre un ganador, para lo cual, con cada movimiento realizado se realiza la verificación de si existe o no un ganador, y el segundo estado posible, es que el tablero se encuentre lleno, por lo que, la condición del bucle es la verificación de que aún hayan casillas vacías.

- Búsquedas: dentro del ciclo de juego, cada jugador (agente o humano), tiene definida una función llamada "next_action" que recibe como parámetro el estado actual del juego, está función es la que es invocada en cada turno correspondiente, y se encarga de definir la siguiente acción que el jugador va a realizar. En cuanto al agente se refiere, dicha función lo que realiza es un proceso de descarte de acuerdo a las estrategías de movimiento definidas y al valor de probabilidad que las mismas poseen. Ese proceso de descarte implica, realizar una serie de búsquedas (aplicación de estrategias) para ir eliminando todas aquellas columnas que no cumplen una estrategía, y asi sucesivamente hasta llegar a

la última estrategía y elegir la más optima. En caso de empate, la estrategía columna se elije al azar. En cuanto al humano, la función "next_action", lo único que realiza es solicitar la entrada de la columna a la cual se quiere realizar la inserción.

Estrategias de movimientos de los agentes

- Blocking 2 vs 1: Dada una probabilidad X, se genera un número aleatorio N, si N está entre 0 y X se usa la estrategia bloquear_2, al contrario si N está entre X y 1 se usa la estrategia bloquear_1. La estrategia bloquear_2 consiste en seleccionar las columnas que cumplan el requisito de si echamos una ficha en dicha columna ésta ficha bloquee dos fichas consecutivas del jugador oponente, lo que es decir, caiga en una posición del tablero que sea secuencial a dos fichas del oponente. La estrategia bloquear_1 funciona de manera similar a la estrageia anterior, la única diferencia es que busca bloquear una ficha del jugador oponente.

Pseudocódigo:

```
function get_action(board, array_number) return array of number object
    inputs: board, structure of the game
            array_number, a array of objects position
    random <- a random number between 0 - 1
    if random >= probability_blocking_2 then
        n <- 2
    else
        n <- 1
    array_blockers <- array with all possibilities of block the opponent with n
sequential pieces
    array_blockers <- get_blockers(board, n)
    for col in array_blockers do
        increase the amount and the strategy of the col in array_number
    return array_number
```

```
function get_blockers(board, max_connect) return all possible blockers near to
opponent symbol
    inputs: board, structure of the game
            max_connect, a number according to blocking 2 or 1
    array_blockers <- []
    for col = 0 to 7 do
        if board is not fill in col then
            board <- insert symbol opponent in col
            if board with last_insert has winner with max_connect then
                array_blockers <- insert col
            board <- delete last_insert
    return array_blockers
```

- Center vs Extremity: Dada una probabilidad X, se genera un número aleatorio N, si N está entre 0 y X se usa la estrategia centro, al contrario si N está entre X y 1 se usa la estrategia extremo. La estrategia centro consiste en seleccionar las columnas que cumplan el requisito de estar en el centro del tablero,

lo que es decir, sea la columna 3, 4 ó 5. La estrategia extremo consiste en seleccionar las columnas que cumplan el requisito de estar en el extremo del tablero, lo que es decir, sea la columna 1, 2, 6 ó 7.

Pseudocódigo:

```
function get_action(board, array_number) returns a array of positions
    inputs: board, structure of the game
            array_number, a array of objects position
    random <- a random number between 0 - 1
    center_columns <- an array of the columns center numbers
    extremity_columns <- an array of the columns extremity numbers
    if random >= probability_center then
        columns <- center_columns
    else
        columns <- extremity_columns
    for col in columns do
        if board is not fill in col then
            increase the amount and the strategy of the col in array_number
    return array_number
```

- Sequential vs Space: Dada una probabilidad X, se genera un número aleatorio N, si N está entre 0 y X se usa la estrategia secuencial, al contrario si N está entre X y 1 se usa la estrategia espacial. La estrategia secuencial consiste en seleccionar las columnas que cumplan el requisito de si echamos una ficha en dicha columna ésta ficha sea consecutiva a una o más fichas del jugador actual, lo que es decir, caiga en una posición del tablero que sea secuencial a una o más fichas del jugador actual. La estrategia espacial consiste en seleccionar las columnas que cumplan el requisito de si echamos una ficha en dicha columna ésta ficha no sea consecutiva a una o más fichas del jugador actual, lo que es decir, no caiga en una posición del tablero que sea secuencial a una o más fichas del jugador actual. En ambas estrategias se analizan los vecinos con búsqueda local y según el criterio se selecciona la posibilidad de usar un determinado vecino dado por la búsqueda.

Pseudocódigo:

```
function get_action(board,array_number) return new movement using the board as
reference.
    inputs: board, the state of the game
            array_number, an array of objects position

    var_random -> a random number
    if var_random < probability:
        return get_sequential_action(board, array_number)
    else:
        return get_space_action(board, array_number)
```

```
function get_sequential_action(board, array_number) return new movement using the
board as reference
    inputs: board, the state of the game
```

```
            array_number, an array of objects position

    array_neighbors -> get_neighbors(board)
    for neighbor in array_neighbors:
        if neighbor has neighbors:
            array_number[neighbor[0]] -> increase amount (neighbor[1])
            array_number[neighbor[0]] -> increase strategy
    return array_number
```

```
function get_space_action(board,array_number) return new movement using the board
as reference
    inputs: board, the state of the game
            array_number: an array of objects position

    array_neighbors -> get_neighbors(board)
    for neighbor in array_neighbors:
        if neighbor does not have neighbors
            array_number[neighbor[0]] -> increase amount
            array_number[neighbor[0]] -> increase strategy
    return array_number
```

```
function get_neighbors(board) return an array with tuples of neighbors
    inputs: board, the state of the game

    array_neighbors -> array of pairs for each column [(x,0),(x1,0)]
    array_positions -> array of cells positions that contain a specific symbol
    array_temp_position ->
    for (row,col) in array_positions:
        for index in range(col-1,col+2):
            array_temp_position.append((board.get_empty_element(index),index))
            for (row_p,col_p) in array_temp_position: #row prime and column prime
                if self.validate_sequential_position(row_p,row): # row validation
                    (temp_col,temp_amount) = array_neighbors[col_p]
                    array_neighbors[col_p] = (temp_col,temp_amount+1)
        array_temp_position.clear()
    return array_neighbors
```

- Horizontal vs Vertical: Dada una probabilidad X, se genera un número aleatorio N, si N está entre 0 y X se usa la estrategia horizontal, al contrario si N está entre X y 1 se usa la estrategia vertical. La estrategia horizontal consiste en seleccionar las columnas que cumplan el requisito de dada una posición (fila, columna) de una ficha, la columna a seleccionar no sea la misma que la columna en la que se encuentra la ficha seleccionada. La estrategia vertical consiste en seleccionar las columnas que cumplan el requisito de dada una posición (fila, columna) de una ficha, la columna a seleccionar sea la misma que la columna en la que se encuentra la ficha seleccionada.

Pseudocódigo:

```
function get_action(board, array_number) returns a array of positions
    inputs: board, structure of the game
            array_number, a array of objects position
    random <- a random number between 0 - 1
    positions_symbol <- array of the positions in the board of the symbol
    for row, col in positions_symbol do
        if random >= probability_horizontal then
            for i = 0 to 6 do
                if i != col then
                    if board is not fill in i then
                        increase the amount of the i in array_number
        else
            if board is not fill in col then
                increase the amount of the col in array_number
    return array_number
```

Estrategia de solución de Algortimos Genéticos

- Configuración: el inicio del programa está determinado por la configuración que se le envie por consola al algoritmo genético, en este sentido, el primer paso de la configuración es el parseo de los parámetros enviados al programa (ver el manual de usuario para comprender la forma en que los parámetros deben ser ingresados), según el valor de los mismos se determina el segundo y último paso de la configuración, en este caso, la instanciación del objeto Genetics_Algorithm, con el número de generaciones y la cantidad de individuos que pasan de una generación a otra.

- Creación de Individuos: Para seguir el plantemiento básico se crean dos individuos del tipo agente cuya función es la de trabajar como jugadores. La creación de agentes fue descrita anteriormente. Se crea esta clase en particular debido a la necesidad de crear métodos y llevar contadores que ayudaran al algoritmo a crear generaciones cada vez más óptima. En la creación del individuo se tomarán 3 metodos esenciales los cuales son

Método de fitness para un individuo particular: Este método recibirá una lista de individuos con los cuales jugará. El individuo en instancia jugará n veces donde n será el tamaño de la lista de individuos. Para optimizar la ejecución del algoritmo sólo se toma en cuenta la premisa que un individuo sólo competirá 1 vez con otro individuo en la misma generación, debido a esto en cada jugada se actualizaran los valores de gane o perdida de cada individuo en juego, en la clase se manejaran las variable "won_games" y "total_games", donde una se encargará de guardar el valor de juegos y ganados y el otro la cantidad de juego respectivamente.

Pseudocódigo:

```
funcion fitness():
        entradas: lista_oponentes, agente_actual
        for oponente in lista_oponentes:
            jugar(agente_actual,oponte)
            if si_ganador_es_1:
                jugador_1_ +=1 # Aumenta puntaje
```

```
            elif si ganador_es_2:
                jugador_1_ +=1
            judador_1_juegos +=1
            judador_2_juegos+=1
        salidas: lista de oponentes actualizada
```

Método de crossover para un individuo particular: En la función del crossover se especifica el cruce entre dos Individuos particulares. Para realizar el cruce, se realiza un cruce inteligente tomando únicamente un heredero que sea capaz de vencer a los Individuos padres para esto, se obtienen las combinaciones entre las estrategias de los dos Individuos y se realiza una competencia interna para obtener un hijo ganador, por optimización debido a la cantidad de combinaciones, se toma el primer hijo que cumpla con los requisitos.

Pseudocódigo:

```
funcion crossover():
    Entradas: agente_cruce,agente_actual
        strategias_agent1 = agente_actual.strategias
        strategias_agente_cruce= agente_cruce_estrategias
        combinaciones = combinaciones(strategias_agent1,strategias_agente_cruce)
        revolver(combinaciones)
        obtener_mejor_individuo(combinaciones,agente_cruce)
    Salidas: Un individuo nuevo
```

Método de mutación: La mutación se basará en una probabilidad definida junto con el algoritmo. La mutación se define como un cambio aleatorio en algún elemente de probabilidad de estrategias. Se selecciona cuál estrategia va a mutar aleatoriamente y se aumenta según un parametro predefino de aumento, este paramétro se puede utilizar para optimizar la definición del algortimo.

Pseudocódigo:

```
funcion mutation():
        entradas: agente
        estrategias = obtener_estrategias_agente
        nueva_probabilidad = obtener_random_estretegia
        cambiar_estrategia(aumeto_predefinido+nueva_probabilidad)
        salidas: agente mutado
```

- Algortimo génetico: La definición principal de nuestro algoritmo génetico se basa en el siguiente descripción de flujo.

```
    1. Inicialización_de_la_población
    2. Cruce de cada elemento de la población con  los demás elementos de la
población
    3. Aplicar la función fitness
```

```
    4. Ordenar el resultado de mejor a peor individuo
    5. Limitar siguiente generación
```

pseudocodigo:

```
funcion algortimo_genetico()
   entradas: generacion, poblacion,limite
   Repetir generacion:
       nueva_generacion -> cruzar_poblacion(poblacion)
       generacion_fitness -> applicar_fitness(nueva_generacion)
       generacion_ordenada-> ordenar_resultado_por_fitness(generacion_fitness)
       poblacion =  limitar_generacion(generacion_ordenad, limite)
   salidas: primer elemento de la última generación
```

Después de la terminación de este ciclo dado por n generaciones se obtiene el mejor elemento de la última generación y se devuelve como resultado.

Cobertura de las pruebas

- Revisar la carpeta htmlcov (en la raíz del proyecto) lugar donde se encuentra el archivo index.html con el reporte completo de las pruebas realizadas

Generación de cobertura:

```
py.test --cov-report html --cov=model tests/
```

- Nota: Algunas no fueron probadas debido a la naturaleza de la función la cual fue considerada como innecesaria.

Distribución de trabajo realizado y notas

Jonathan Martínez Camacho:

- Diseño e implementación de la solución general
- Desarrollo e implementación del board
- Desarrollo e implementación de la configuración del juego y el algoritmo genético
- Colaboración e implementación del agent
- Documentación

Katherine Molina Sánchez:

- Diseño e implementación de la solución general
- Desarrollo e implementación de las estrategías de movimientos

- Desarrollo e implementación de los algoritmos genéticos
- Colaboración e implementación del agent
- Documentación

Mariana Rojas Semeraro:

- Diseño e implementación de la solución general
- Desarrollo e implementación de las estrategías de movimientos
- Desarrollo e implementación de los jugadores (player, agent, human)
- Desarrollo e implementación del game
- Documentación

Notas:

- Jonathan Martínez Camacho: 100
- Katherine Molina Sánchez: 100
- Mariana Rojas Semeraro: 100
