Universidad de Oriente

Núcleo Anzoátegui

Escuela de Ingeniería y Ciencias Aplicadas

Departamento de Ingeniería de Computación y Sistema

Taller de inteligencia artificial (0724371)



Asignación 2 (Juego Bridg-it)

Profesor: Claudio Cortinez Bachiller:

Sección: 01 -Leonel Araujo. C.I: 28.462.041

- Robson Gutiérrez. C.I: 24.979.483

Barcelona, 2021

1. Elabore un los modelos correspondientes (UML), para al desarrollo del Agente y su respectiva documentación. Estos saber estos son: Caso-Usos, Diagrama de Clases, Diagrama de secuencias y el Diagrama de Estados.

|  |
| --- |
| <<interface>>  Tablero |
| - pantalla: Objeto display de pygame |
| - puntos: Lista |
| - agente: Objeto Agente |
| - turn: String |
| - jugador: Objeto Jugador |
| + inicializarPuntos(): None |
| + iniciarJuego(): None |

Diagrama de clases

|  |
| --- |
| Puntos |
| - valor: int |
| - presionada: boolean |
| - x: int  \* |
| - y: int |
| - pantalla: Objeto display de pygame |
| - color: tupla |
| - circulo: Objeto rect de pygame |
| + presionar(): None |
| + pintarPunto(): None |

1

1

1

1

1

|  |
| --- |
| Jugador |
| - color: String |
| + conectar(Objeto\_ display\_pygame pantalla, int x1, int y1, int x2, int y2): None |
| + realizarMov(Lista encola, Diccionario tablero, Objeto\_ display\_pygame pantalla): String |

|  |
| --- |
| Agente |
| - color: String |
| - valor: int |
| - profundidad: int |
| - siguienteEstado: Objeto Agente |
| - alpha: int |
| - beta: int |
| - mundo: diccionario |
| - mejorEstado: tupla |
| + mover(Lista mov): None o String  + búsqueda(int mov, Lista conexiones, String jugador): String  + fin\_del\_juego(): Lista  + movValidos(String jugador): Lista  + escogerMov(): None o Lista  + alfa\_beta(Objeto Agente, String jugador, diccionario oponente, int maxProf): Objeto Agente |

\* (-): Representa que el acceso al atributo o función es privado.

\* (+): Representa que el acceso al atributo o función es público.

Diagrama de casos de usos

<<include>>

mover

<<extends>>

<<extends>>

movValidoss

<<include>>

escogerMov

<<extends>>

<<extends>>

busqueda

<<extends>>

inicializarPuntos

<<extends>>

<<extends>>

pintarPuntos

<<extends>>

<<include>>

iniciarTablero

Tablero

presionar

Puntos

conectar

<<extends>>

realizarMov

Jugador

<<include>>

<<include>>

alfa\_beta

Fin\_del\_juego

Agente

Diagrama de estados

Conectar Puntos

[Fin == False]

[Fin == True]

Cambiar Puntos

Dibujar Línea

Cambiar puntos

Cambiar color

Iniciar Juego

iniciarJuego()

Inicializar Tablero

Dibujar Tablero

Tablero

Agente

[Fin == Si]

[Fin == No]

[turn == Jugador]

Pintar Mov

mover()

Escoger Mov

[turn == Agente]

escogerMov()

Esperar turno

iniciarJuego()

Fin del juego

Sin cambiar color

Regresar a color inicial

Presionar Puntos

Cambiar color

Fin juego

Dibujar Puntos

Punto

pintarPuntos()

[Fin juego]

[turn == Agente]

[turn == Jugador]

Esperar Turno

realizarMov()

presionar()

Realizar Mov

Presionar Punto

iniciarJuego()

Jugador

Diagrama de secuencia

mov,, result = escogerMov()

[turn == “Azul”]

agente: Agente

presionar()

presionar()

puntos: Punto

jugador: Jugador

result = realizarMov(encola, tabla, pantalla)

presionar()

[turn == “Rojo”]

tablero: Tablero

iniciarJuego()

inicializarPuntos()

Usuario

agente2 = alfa\_beta(agente, jugador, oponente, maxProf=3)

verificar = fin\_del\_juego()

[event.type ¡= QUIT and ganador ¡= “Azul” and ganador ¡= “Rojo”]

[puntos ¡= None]

pintarPunto()

1. Diseño del algoritmo del agente (solo los elementos funcionales que resuelven el problema)

escogerMov(A)

1. agente2 <- Agente(color)

2. agente2.mundo <- mundo

3. agente2 <- alfa\_beta(A, agente2, color, {“Azul”: “Rojo”, “Rojo”: “Azul”})

4. movVal <- movValidos(A, color)

5. verificar <- Verdadero

6. mientras agente2.mejorEstado ¡= Vacio

7. si agente2.color == color

8. i <- 0

9. mientras i < longitud(movVal)

10. si movVal[i] == agente2.mejorEstado

11. verificar <- Falso

12. i <- i + 1

13. si verificar == Falso

14. mover(A, agente2.mejorEstado)

15. retorna [agente2.mejorEstado, agente2.valor]

16. agente2 <- agente2.siguienteEstado

alfa\_beta(A, agente, jugador, oponente)

1. alpha <- agente.alpha

2. beta <- agente.beta

3. ganador <- fin\_del\_juego(agente)

4. movVal <- movValidos(agente, agente.color)

5. si agente.profundidad < 3 y (ganador[0] == Vacio o movVal ¡= [])

6. i <- 0

7. mientras i < longitud(movVal)

8. nextAgente <- Agente(agente.color)

9. nextAgente.profundidad <- agente.profundidad + 1

10. nextAgente.mundo <- agente.mundo

11. mover(nextAgente, movVal[i])

12. nextAgente.color <- oponente[agente.color]

13. ganador <- fin\_del\_juego(nextAgente)

14. si ganador[0] == “Azul”

15. nextAgente.valor <- ganador[1]

16. agente.valor <- nextAgente.valor

17. agente.siguienteEstado <- nextAgente

18. agente.mejorEstado <- movVal[i]

19. retorna agente

20. sino

21. nextAgente.valor <- ganador[1]

22. nextAgente.alpha <- alpha

23. nextAgente.beta <- beta

24. alfa\_beta(A, nextAgente, jugador, oponente)

25. si agente.color == “Azul”

26. si ((agente.siguienteEstado == Vacio)

o nextAgente.valor > agente.valor)

27. agente.valor <- nextAgente.valor

28. agente.siguienteEstado <- nextAgente

29. agente.mejorEstado <- movVal [i]

30. alpha <- max(alpha, agente.valor)

31. si agente.valor >= beta

32. retorna agente

33. sino

34. si ((agente.siguienteEstado == Vacio)

o nextAgente.valor < agente.valor)

35. agente.valor <- nextAgente.valor

36. agente.siguienteEstado <- nextEstado

37. agente.mejorEstado <- movVal[i]

38. beta <- min(beta, agente.valor)

39. si agente.valor <= alpha

40. retorna agente

41. i <- i + 1

42. retorna agente

La funcion movValidos lo que hace es buscar cuales son los movimientos que puede realizar cualquier jugador, esto depende del color que se le pase como parámetro que indica si es el agente (“Azul”) o el usuario (“Rojo”), esta función al final retorna un arreglo o lista de posiciones o movimientos que pueden realizarse.

La función fin\_del\_juego() se usa para saber si hay un ganador o no, donde también va colocando puntuaciones dependiendo del ambiente (tablero), ya que lo que hace en si es ver si hay dos esquinas conectadas de un jugador (puede ser tanto el agente como el usuario), esta función retorna un arreglo con dos valores uno donde indica al ganador (este puede ser Vacio) y el otro la puntuación que tiene el agente en ese instante.

3. Describa las características (propiedades) presentes en el ambiente.

1. Totalmente observable: Con la información que se percibe del entorno se puede saber cual movimiento realizar a través de unas series de cálculos (toma de decisiones), además de que no se necesita un estado interno que indique cual es la situación, porque el mismo entorno nos lo da.
2. Estratégico: Es este tipo, ya que el entorno es determinista, porque se puede determinar el siguiente estado a través del estado en el que se está, pero no se puede determinar en si los movimientos que puede realizar el jugador, por esto es estratégico, porque se busca predecir cual movimiento puede realizar el jugador buscando los movimientos más beneficios para el agente.
3. Secuencial: Es este tipo porque las acciones que realiza el agente puede afectar a las acciones futuras.
4. Estático: Ya que el entorno no va cambiando con el tiempo mientras el agente piensa que hacer, se dice que es estático, el agente puede pensar que movimiento hacer sin preocuparse de que haya cambiado el entorno, ya que se bloquea el movimiento del usuario cuando es el turno del agente.
5. Discreto: Porque se tiene un conjunto de estados finitos que son distintos, es decir no tiene un estado continuo, o dicho de otra forma problemas que continuamente tiene que resolver sin parar.
6. Multiagente competitivo: Ya que el usuario al realizar un movimiento ocasiona que se mejore el movimiento que el agente pueda realizar, pero los movimientos del agente está limitado o minimizado por las reglas que tiene el juego (Bridg-it).

4. Describa la estructura del Agente diseñado según el tipo de agente al cual pertenece.

Las percepciones que recibe el agente son el movimiento que realiza el usuario, esto ocasiona que se ejecute la función del agente (escogerMov) para poder obtener la acción que este va a realizar y así sucesivamente hasta que se termina el juego. La estructura del agente es basado en utilidad, porque se usa una función de utilidad (fin\_del\_juego()), que nos indica cual estado o jugada es la más conveniente realizar, para así intentar predecir cuál sería el mejor movimiento, al momento de realizar la búsqueda en el escogerMov(). La función de utilidad busca que el agente escoja un movimiento en el que no pierda, es decir que en el que ninguno de los jugadores haya ganado o en el que el gana (colocando a esta última con mayor ponderación).