Juan Pablo Vielma
A01203239

Challenge Report - DominAI

AI has been used to solve boardgames in the past decades. Since Von Neumann's game theory was published many attempts to make an intelligent system able to play chess and many other games have been achieved, but nobody has been able to solve an incomplete information game. Incomplete information multiplayer games characterize themselves for being NP-Hard problems and therefore "impossible" to solve, heuristics must be incorporated in order to approach a solution. In the following report we're going to try to reach a solution for a four-player partnership domino game by implementing a combination between decision matrixes and the Minimax algorithm.

Probability decision Matrix

Probability is perhaps the most determining decision making factor for any game strategy, therefore, it is very important to be aware of all possible scenarios and prioritize according to a precise probability. This section may not be very related to Artificial Intelligence, but it will be responsible for providing the foundations in which all implementation will depend.

Since we have incomplete information for all players, we will create a probability profile for each of them. We'll call this profiles Player #1, Player #2 and Player #3 respectively. Three different matrixes will form each profile: the binary matrix that will store values of true or false (1 or 0) for each domino tile, the relative probability that will combine the binary probability with all other player matrixes and finally the absolute probability which is a combination between the relative probability and the current number of tiles in each player's hand. The resulting Absolute probability will be equivalent to the real probability.

Binary Probability

Binary probability will consist in determining if a player can have or can't have a specific tile. A 7x7 matrix will store 1|0 values, each cell (or cells) will represent a specific tile. At the beginning all values will be initialized to 1. Values will be set to 0 if any of these conditions are met:

-The user has that tile in his hand.
-That tile has already been played.
-The player passed to that specific value (all tiles containing that value are set to 0)

Juan Pablo Vielma
A01203239

P1	0	1	2	3	4	5	6
0	1	0	1	0	1	1	0
1	0	0	0	0	0	0	0
2	1	0	1	0	0	0	1
3	0	0	0	1	0	1	1
4	1	0	0	0	0	0	1
5	1	0	0	1	0	0	0
6	0	0	1	1	1	0	0

P2	0	1	2	3	4	5	6
0	0	0	0	0	1	0	0
1	0	1	0	0	0	0	1
2	0	0	1	1	0	0	0
3	0	0	1	1	1	0	0
4	1	0	0	1	0	0	1
5	0	0	0	0	0	0	0
6	0	1	0	0	1	0	0

P3	0	1	2	3	4	5	6
0	1	0	1	0	0	1	0
1	0	0	1	0	0	1	0
2	1	1	0	0	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	1
5	1	1	0	0	0	1	1
6	0	0	0	0	1	1	1

Matriz Acumulada

	0	1	2	3	4	5	6
0	2	0	2	0	2	2	0
1	0	1	1	0	0	1	1
2	2	1	2	1	0	0	1
3	0	0	1	2	1	1	1
4	2	0	0	1	0	0	3
5	2	1	0	1	0	1	1
6	0	1	1	1	3	1	1

Relative Probability

La matriz de probabilidad relativa será la matriz que nos indique la probabilidad que tiene cada jugador de tener cierta ficha al evaluar únicamente las fichas que ya estén jugadas en el juego, las fichas que están en nuestra mano de juego y las ocasiones en las que el jugador no ha tirado ninguna ficha. Para calcular la matriz de probabilidad relativa, la fórmula será:

$$\sum_{n=0}^{6}\sum_{m=0}^{6} R_{nm} = \frac{1}{A_{nm}} P_{nm}$$

en donde R es la nueva matriz de probabilidad relativa, A la matriz acumulada, P la matriz binaria, y n y m los índices para cada fila y columna de la matriz. En otras palabras, aplicaremos una regla de 3 en la que dividiremos 100 entre el valor de la matriz acumulada y posteriormente multiplicaremos por el valor de la matriz binaria para las celdas correspondientes. Todo esto siempre y cuando el valor de la matriz acumulada sea diferente de 0, de ser así, el valor de probabilidad relativa, es 0.

Juan Pablo Vielma
A01203239

PR1	0	1	2	3	4	5	6
0	0	0.3	0.3	0	0.5	0.3	0
1	0.3	0.3	0.3	0	0.3	0.3	0
2	0.3	0.3	0	0	0	0	0
3	0	0	0	0	0	0	0
4	0.5	0.3	0	0	0	0	1
5	0.3	0.3	0	0	0	0	0
6	0	0	0	0	1	0	0

PR2	0	1	2	3	4	5	6
0	0	0.3	0	0.5	0.5	0.33	0
1	0.3	0.3	0	1	0.3	0.33	0
2	0.3	0.3	0	0	0	0	0
3	0.5	1	0	0	1	0.5	0
4	0.5	0.3	0	1	0	0	0
5	0.3	0.3	0	0.5	0	0	0
6	0	0	0	0	0	0	0

PR3	0	1	2	3	4	5	6
0	0	0.3	0.3	0.5	0	0.3	0
1	0.3	0.3	0.3	0	0.3	0.3	0
2	0.3	0.3	0	0	0	0	0
3	0.5	0	0	0	0	0.5	0
4	0	0.3	0	0	0	0	0
5	0.3	0.3	0	0.5	0	0	0
6	0	0	0	0	0	0	0

Fichas Restantes:	
Jugador 1=	4
Jugador 2=	4
Jugador 3=	5
Fichas indeterminadas:	
Jugador 1=	3
Jugador 2=	2
Jugador 3=	5

Absolute Probability

Una vez calculada la matriz de probabilidad relativa tenemos suficiente información para hacer un prototipo funcional, pero esta información podría ser mucho más precisa. La probabilidad relativa no toma en cuenta el número de fichas con las que cuenta cada jugador. Este factor puede ser de suma importancia para calcular probabilidades más precisas y descartar ciertas hipótesis. A este cálculo de probabilidad tomando en cuenta fichas jugadas, fichas en nuestra mano, turnos pasados y número de fichas en la mano de cada jugador lo llamaremos probabilidad absoluta.

El primer paso para calcular la probabilidad absoluta será calcular cuántas “fichas indeterminadas” tiene cada jugador. Cuando decimos fichas indeterminadas, nos referimos a cuántas fichas de ese jugador no sabemos su valor con exactitud. Para calcular el número de fichas indeterminadas, simplemente calcularemos la diferencia entre fichas de del jugador X y la cantidad de valores de “1” en la matriz de probabilidad relativa para el jugador X.

En caso de que el valor de fichas indeterminadas de cierto jugador sea 0, esto nos brinda nueva información indicando que todas las fichas con una probabilidad menor al 100% en la matriz de probabilidad relativa deberían tener un valor real de 0%. Para realizar dicho cálculo simplemente asignaremos “0” en la matriz binaria a todas las celdas correspondientes en las que el valor sea distinto a 100% en la tabla de probabilidad relativa, siempre y cuando el valor de fichas indeterminadas sea igual a “0”. Este proceso nos ve forzados a repetir el cálculo de matriz acumulada y matriz de probabilidad relativa para cada jugador y en caso de repetirse el escenario de tener que modificar la tabla de probabilidad binaria se repetirá el mismo proceso una y otra vez hasta que no se tengan que realizar modificaciones.

Juan Pablo Vielma
A01203239

Por otro lado, también debemos calcular el número de celdas con valor mayor a 0 en la tabla de probabilidad relativa. En caso de que este valor sea igual al número de fichas de domino restantes en la mano de juego del oponente, debemos asignar “0” al valor de la ficha en la matriz binaria para todos los otros oponentes ya que esto determina que solo este jugador puede tener esas fichas. Por ejemplo, si solo hay 3 fichas que un jugador puede tener, y el jugador tiene exactamente 3 fichas, entonces hay un 100% de probabilidad que tenga una de esas fichas y un 0% de probabilidad que el resto de los oponentes tengan las mismas.
Una vez aplicada la normalización de la tabla de probabilidad binaria y la probabilidad relativa, podemos pasar al cálculo de la probabilidad absoluta. La probabilidad absoluta se calcula de la siguiente manera: divide el 100% entre la suma total de fichas indeterminadas que tiene todos los jugadores que cuentan con una probabilidad de 1 en la matriz binaria para la ficha a calcular. Multiplica el valor obtenido por el número de fichas indeterminadas del jugador X. El valor obtenido es la probabilidad real que tiene el jugador X de tener la ficha Y.

PA1	0	1	2	3	4	5	6
0	0	0.3	0.3	0	0.6	0.3	0
1	0.3	0.3	0.3	0	0.3	0.3	0
2	0.3	0.3	0	0	0	0	0
3	0	0	0	0	0	0	0
4	0.6	0.3	0	0	0	0	1
5	0.3	0.3	0	0	0	0	0
6	0	0	0	0	1	0	0

PA2	0	1	2	3	4	5	6	
0	0	0	0.2	0.3	0.4	0.2	0	78%
1	0.2	0	0.2	1	0.2	0.2	0	100%
2	0.2	0	0	0	0	0	0	29%
3	0.3	1	0	0	1	0.3	0	100%
4	0.4	0	0	1	0	0	0	100%
5	0.2	0	0	0.3	0	0	0	49%
6	0	0	0	0	0	0	0	0%

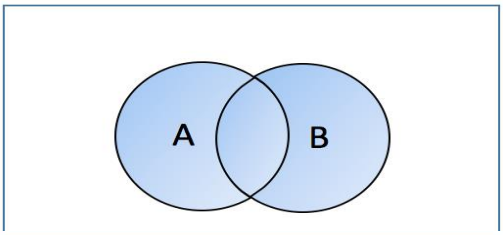
PA3	0	1	2	3	4	5	6
0	0	0.5	0.5	0.7	0	0.5	0
1	0.5	0.5	0.5	0	0.5	0.5	0
2	0.5	0.5	0	0	0	0	0
3	0.7	0	0	0	0	0.7	0
4	0	0.5	0	0	0	0	0
5	0.5	0.5	0	0.7	0	0	0
6	0	0	0	0	0	0	0

Value Probability

En la mayoría de los casos, no nos importa mucho la probabilidad que tiene un jugador de tener una ficha exacta, lo que más nos importa es la probabilidad que tiene de tener una ficha cualquiera con un valor X. Es decir, nos puede no importar la probabilidad de que un jugador tenga la ficha “6|2”, pero nos importa mucho la probabilidad de que un jugador tenga cualquier ficha con un valor de “6” o con un valor de “2”. Ya que tenemos la probabilidad real para cada ficha, podemos calcular la probabilidad que existe para que un jugador tenga una ficha con X valor fácilmente. Una vez calculando la probabilidad exacta para cada ficha, es muy sencillo calcular la probabilidad. Tomando en cuenta la tabla de probabilidad absoluta, tomaremos la probabilidad de que exista cualquier valor de la columna de la tabla de probabilidad.

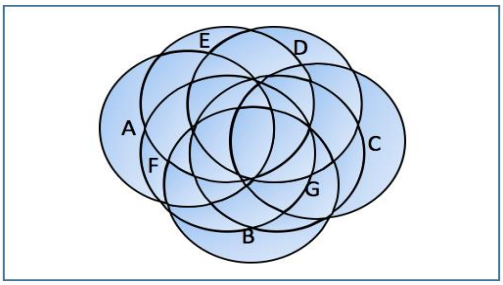
Juan Pablo Vielma
A01203239

Dado a que la probabilidad de cada ficha son eventos no mutuamente excluyentes, usaremos la siguiente formula de probabilidad:



$$\boldsymbol{P}(\boldsymbol{A} \text{ ó } \boldsymbol{B}) = \boldsymbol{P}(\boldsymbol{A}) + \boldsymbol{P}(\boldsymbol{B}) - \boldsymbol{P}(\boldsymbol{A}, \boldsymbol{B}).$$

Considerando que existe más de 2 variables en este cálculo, tenemos que repetir la formula en varias ocasiones por cada una de las 7 variables que representan las fichas con valores de 0 a 6. Nuestra nueva fórmula para calcular la probabilidad se verá de la siguiente manera:



$$\boldsymbol{P}(\boldsymbol{A} \text{ ó } \boldsymbol{B} \text{ ó } \boldsymbol{C} \text{ ó } \boldsymbol{D} \text{ ó } \boldsymbol{E} \text{ ó } \boldsymbol{F} \text{ ó } \boldsymbol{G}) = \boldsymbol{P}(\boldsymbol{P}(\boldsymbol{P}(\boldsymbol{P}(\boldsymbol{P}(\boldsymbol{A} \text{ ó } \boldsymbol{B}) \text{ ó } \boldsymbol{C}) \text{ ó } \boldsymbol{D}) \text{ ó } \boldsymbol{E}) \text{ ó } \boldsymbol{F}) \text{ ó } \boldsymbol{G})$$

Minimax

The minimax algorithm is a decision taking algorithm that builds a decision tree with all possible game combinations and picks a path based on specified heuristics. Each level of the tree will represent a different player and it is assumed they alternate turns equivalently. The convention for naming this players is MIN and MAX in which MAX will always choose a path with the greatest value (maximize profit) and MIN will choose the path with the lowest value (minimize profit). This scenario assumes that MIN is rational and fully aware of the state of the game as possible and will always try to minimize MAX's possibilities of winning. The following image represents how the decision tree is built based on the utility value of each node and how MIN and MAX are distributed.

In our case we had to make a couple modifications to the pure version of the algorithm since some issues needed to be taken care of:

- **Exponential Growth:** the possible game distributions for a four-player double-six domino game are (28C7)(21C7)(14C7) which is equivalent to 4.725×10^{14} distributions. Considering all possible combinations may result in a waste of computational resources and time, but this exponential growth is reduced along with the development of the game. The solution to the exponential growth of the decision tree through the beginning of the game is to reduce the depth depending on the current turn. The minimax algorithm will iteratively deepen according to n in which $n = turns\ passed + number\ of\ Players$ (the number of players is added to make sure all players get to have a turn during the first run of the algorithm). The beginning of the game is not a critical stage in which we must take important decision, therefore, we can afford to have a lower depth for the decision tree. This solution will prevent us from exploring millions of options throughout the first turns and still have a complete decision tree prior to the half of the game where decisions start to be crucial and we have much more information of the game.

Juan Pablo Vielma
A01203239

- **Four-player game:** The minimax algorithm is a solution to a two player complete information game. Domino is not only an incomplete information game, it is a four-player game. Even though there are four players, players are playing in partners, which makes up two teams. For this particular scenario, one team will play the MIN role and the other team will play the MAX role. Applying minimax to an incomplete information two-player game would normally result in having incomplete information about MIN, but complete information about MAX. In the four-player scenario previously suggested we will end up having both incomplete information about MIN and MAX. The characteristics of this game make it impossible to soften this scenario and therefore heuristics must be used to tackle the problem.

- **Incomplete Information game:** A pure minimax implementation requires having complete information about the state of the game, that is obviously not the case in partnership dominoes. To compensate the lack of information a heuristic will be incorporated to every node based on a probability function calculated from the previously obtained probability matrix.

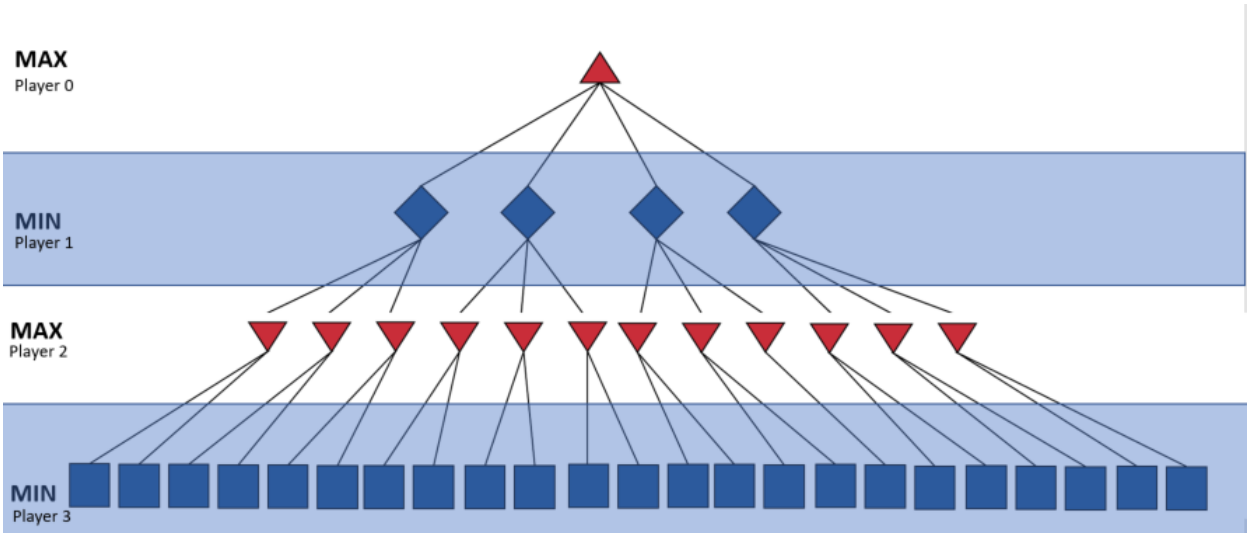
- **Node Utility value:** Since the minimax algorithm is not always going to reach a leaf node (because the tree will not be fully expanded in the first turns), a value to each path must be assigned evaluating the state of all nodes by determining if it's a favorable state or not, we will call this nodes "transit nodes". A favorable state will assign a value of +1, while an unfavorable state will be worth -1. On the other hand, leaf nodes will have values of +10 or -10 (according to winning or loosing). Assigning values ten times higher to "leaf nodes" than the possible value of a "transit node" is done to automatically surpass any other value and give priority to paths that reach a leaf node since leaf nodes determine a desired outcome while transit nodes just represent a desired state.

Now that the previous issues have been resolved, we can proceed to the implementation.

Determining MIN & MAX

If recursion is applied to build the tree, we must determine if the player in turn is MIN or MAX. This issue is very easily resolved when having 2 players, but it is a little bit more complicated since we are dealing with 4 players, each one with independent profiles, 3 of whom we have incomplete information. Considering the turns are sequential, to determine the player on turn, a value will be stored in each node containing the current tree depth. The player on turn will be determined by the operation $Player\# = TreeDepth\ mod(NumberOfPlayers)$ (it is assumed that player #0 is the player on turn in the root node). Once the player on turn is determined, MIN will be assigned if $PlayerOnTurn = Player1|Player3$ and MAX will be assigned if $PlayerOnTurn = Player0|Player2$.

Juan Pablo Vielma
A01203239



Heuristics and Utility Value

Alpha Beta Pruning

Implementing Minimax requires a big amount of memory to expand a tree with tens of thousands of nodes. Alpha Beta pruning is a technique used to reduce tree expansion in a minimax algorithm. To implement such technique, two variables (alpha & beta) are stored in each node. Alpha will maximize the possible value and beta will minimize value. Both variables will be initialized to their worst possible value (ß=∞, ∂=-∞). A parent node will assign its current value of *alpha* & *beta* to every child node. On the other hand, *alpha* and *beta* values will be modified according to the values of the current node. If the node is MAX*, alpha* will be assigned the highest value that has been found yet. If the node is MIN, *beta* will be assigned the lowest value that has been found. Once that real values have been assigned to *alpha* and *beta*, we must update the alpha and beta values of all nodes that haven't been expanded yet and then we can proceed with pruning the tree. If a node is MAX it will only expand a child node if *beta* is greater than the current value. On the other hand, if the node is MIN, it will only expand a child node if *alpha* is smaller than the current value. The basis of this technique relies in that no matter what value is in the search tree, as long as the pruning condition is true, that value will be useless and not taken into account. This way, we can prevent the minimax implementation from expanding nodes with big branches that are never going to be taken into account and save a lot of memory space and execution time.

Finding the best Move

<u>Results</u>

To evaluate the performance of the AI, three different scenarios are going to be tested. Scenario number 1 will test the AI playing against another "Hard" partnership-domino playing AI. Scenario number 2 will test the AI against a "beginner strategy" that is simply a greddy strategy

that consists on playing the highest available domino tile on your turn. Finally the third test will involve testing the AI against 3 human players with experience playing partnership dominoes.

AI vs ComputerAI

Won	29	71%
Lost	12	29%

AI vs Beginner strategy

Won	643	56.35%
Lost	475	41.63%
Tied	23	2.02%

This results turned out quite unexpected since there was a higher winning ratio expectancy. My hypothesis for this results is that this strategy is totally unpredictable and the minimax algorithm is unable to predict most player's moves since they are not rational at all. After these results, a new question came up: "How many times did the outcome changed by using the minimax strategy instead of the greedy 'beginner strategy'?". To solve such question, I simulated another greedy player and played the exact same game as the Minimax and then compared the results. The following table shows how many games were won because of the change of strategy and how many were lost instead, as well as the ones whose outcome did not change.

Won	119	26.68%
Lost	80	17.94%
No Change	247	55.38%

AI vs Human
[To do…]