OUTPERFORMING GAME THEORETIC PLAY WITH OPPONENT MODELING IN TWO PLAYER DOMINOES

THESIS

Michael M. Myers, Captain, USAF

AFIT-ENG-14-M-57

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-14-M-57

OUTPERFORMING GAME THEORETIC PLAY WITH OPPONENT MODELING IN TWO PLAYER DOMINOES

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Electrical Engineering

Michael M. Myers, BS

Captain, USAF

March 2014

AFIT-ENG-14-M-57

OUTPERFORMING GAME THEORETIC PLAY WITH OPPONENT MODELING IN TWO PLAYER DOMINOES

Michael M. Myers, BS
Captain, USAF

Approved:

//signed//	7 March 2014
Brett J. Borghetti, PhD (Chairman)	Date
//signed//	11 March 2014
Maj Brian G. Woolley (Member)	Date
//signed//	6 March 2014
Maj Kennard R. Laviers (Member)	Date
//signed//	7 March 2014
Gilbert L. Peterson, PhD (Member)	Date

Abstract

Dominoes is a partially observable extensive form game with probability. The rules are simple; however, complexity and uncertainty of this game make it difficult to apply standard game theoretic methods to solve. This thesis applies strategy prediction opponent modeling to work with game theoretic search algorithms in the game of two player dominoes. This research also applies methods to compute the upper bound potential that predicting a strategy can provide towards specific strategy types. Furthermore, the actual values are computed according to the accuracy of a trained classifier. Empirical results show that there is a potential value gain over a Nash equilibrium player in score for fully and partially observable environments for specific strategy types. The actual value gained is positive for a fully observable environment for score and total wins and ties. Actual value gained over the Nash equilibrium player from the opponent model only exist for score, while the opponent modeler demonstrates a higher potential to win and/or tie in comparison to a pure game theoretic agent.

Acknowledgments

I would like to thank the members of my committee: Doctor Brett Borghetti, Majors Kennard Laviers and Brian Woolley and Dr. Gilbert Peterson. I have gained a countless amount of knowledge from each of your classes and your advisory sessions. My work would not be complete if it weren't for you. To Lt. Col. Jeffrey D. Clark (honorary committee member), your class and offline teachings helped me to gain great ideas as to how I would build certain aspects of my implementation.

I would also like to give special thanks to Captain Kyle Stewart for spending countless hours helping me with Qt and other C++ related problems that I came across during my research process. I know you had your doctoral research to do but you always took time to help students like me. I would not be finished with this project without your help.

I would also like to thank the members of the AFIT Cyber ANiMaL group for giving me advice on how to present my thesis points as well as where to go in my research.

Lastly, I would like to thank my loving wife for putting up with me for all of these months here at AFIT. I know it wasn't easy on you taking care of the house and family in my absence but I truly appreciate it.

Michael M. Myers

Table of Contents

Page

List of Figures

Page

List of Tables

OUTPERFORMING GAME THEORETIC PLAY WITH OPPONENT MODELING IN TWO PLAYER DOMINOES

I. Introduction

Problem Statement

The game of dominoes is a partially observable extensive form game (or POEFG) with probability. An extensive form game is a game in which each player takes an action and there are payoffs as a result of the actions made [22]. The rules (found in Appendix A) are simple. With its numeric tile structure and straightforward rule set, dominoes is a resizable game which allows researchers to examine simple to very complex versions of POEFG's with probability by only adjusting a few parameters in the game.

Dominoes is also probabilistic nature. The rules state that there is a boneyard (i.e. the pile of faced down dominoes) that players pull from when there is no available play. This adds the element of chance to the game tree. Moreover, players can only view their hand of play and the contents on the board. They cannot see their opponents' hands or the boneyard.

The goal of this research is to produce an opponent modeling algorithm that can predict an opponent's style of play as well as provide an optimal move in a partially observable environment with probability. This research focuses on choosing optimal moves by exploiting the opponent's strategy in place of applying the Nash equilibrium solution. (Nash equilibrium is achieved when a player, given strategies of other players in the same game, has nothing to gain by unilaterally switching his strategy [25].) Common algorithms like MiniMax (explained in Chapter 2) apply a Nash Equilibrium

(NE) solution to select optimal moves. This research evaluates the expected and actual gain in value that a specific opponent modeler provides over a NE solution for the game of two player dominoes. The next section outlines the complexity of dominoes as well as how this game is simplified to perform this research.

Complexity of Dominoes

There are several factors that influence computational complexity in dominoes. The first element of complexity involves the branching factor because the larger the branching factor becomes, the longer a decision engine has to search to find the best action. (Often the growth rate is much worse than linear and often even worse than polynomial). The number of starting states also makes a big impact on the game because unlike many board games such as tic-tac-toe and chess, dominoes bears billions of starting states (as illustrated in Table 1 and Equation 2). Lastly, dominoes is a game of partial observability where many aspects of the game are hidden. Furthermore, dominoes adds probability with boneyard pulls. All of these issues make solving dominoes a hard problem.

Branching Factor.

The branching factor of dominoes is calculated by examining the number of possible actions or moves per turn. This number involves the number of dominoes in a player's hand and the number of places that a domino can be positioned on the board. Additionally, a domino can be placed in more than one location depending on the domino and the player's strategy. Figure 1 illustrates how dominoes can be placed on the left or right side of the spinner, (the first domino placed on the board.)

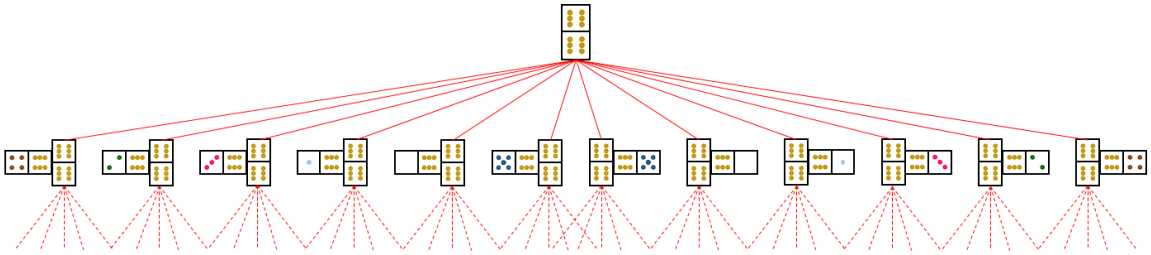

Figure 1: Example instance of double-6 dominoes game tree where the opponent has all possible responses to spinner.

This factor multiplies the branching factor by up to four depending on legal plays available.

Starting State.

The starting state of dominoes is important in comparing this game with fully observable board games such as chess because board games tend to only have a single starting position. Dominoes is interesting because the starting state possibilities change depending on the number of players in the game and the size (or number of dominoes) of the set. Games like poker also have multiple start states; however, unlike poker dominoes has multiple payouts throughout the game instead of one main payout at the end of a hand.

To calculate the number of starting states in two player dominoes, the set size must be found first. The set size depends on the domino with the highest pip count in the set. Set size is calculated by applying

$$S = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} \tag{1}$$

where n represents the highest pip count in a set of dominoes + 1 (i.e., for double-6 dominoes, $n = 7$). One is added to the highest pip count to account for the zero (or blank) dominoes. By applying Equation 1, double-6 dominoes contain 28 dominoes while double-3 dominoes contain 10 dominoes. Now that the set size is calculated, Equation 2 illustrates the number of possible starting states for two player dominos with set size L and a boneyard size B.

$$SS = \binom{L}{B} \times \binom{L-B}{D} \tag{2}$$

Where SS is the number of starting states, L is the size of the set, B is the boneyard size ($B \leq L$-2) and D is the size of a player's hand (both players have equal hand sizes). Table 1 illustrates how the number of starting states grows exponentially as the pip size grows.

Table 1: Table of Starting States for Dominoes.

Pip Size	Dominoes in set	Distribution	Starting States
Double-2	6	2-2-2	20
Double-3	10	3-3-4	4,200
Double-4	15	5-5-5	576,576
Double-5	21	7-7-7	271,591,320
Double-6	28	9-9-10	6.38×10^{11}

While strategies in many board games involve computing a set of plays based on a single board state, 638 billion starting states make it very difficult to compute the perfect play for any starting state at the beginning of a game.

Uncertainty.

Dominoes is also a partially observable game with probability. This adds the element of uncertainty to the complexity of the game by introducing chance nodes and information sets. A chance node is a position on the game tree in which probability is associated with the action taken. For dominoes, chance nodes arise when a player pulls from the boneyard. Information sets are positions on the game tree in which the agent has no knowledge of its position on the tree. Figure 2 illustrates a partially observable game instance (constructed in Gambit software [21]) of double-2 dominoes where there are two player agents, the player (red) and opponent (blue). There is also a chance agent (green) that represents the probabilistic property of drawing dominoes from the boneyard.

The player agent can observe its own two dominoes that are (1|1) (where (1|1) represents the one-one domino) and (1|0); however, it cannot see the target agent's two dominoes or the other two dominoes in the boneyard. From the player's perspective, the target and boneyard share a uniform distribution of the dominoes (2|2), (2|0), (2|1) and (0|0) with each entity having 2 unknown dominoes [21].

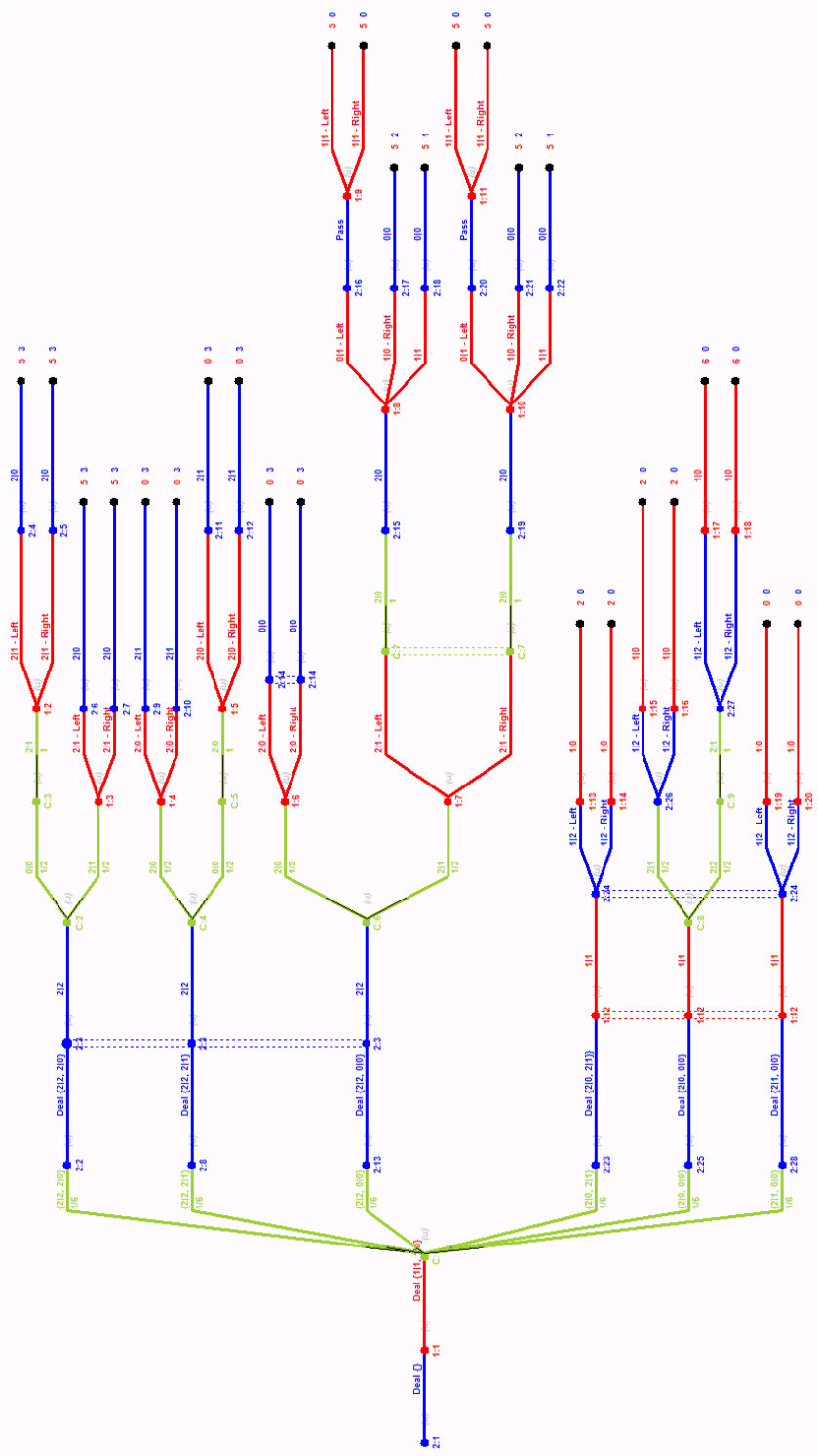

Figure 2: Partially observable game with probability (created with Gambit game theory software [21]).

Information sets (labeled by dashed lines) indicate a set of nodes in which players cannot be sure to which node they belong. For dominoes, information sets represent instances when players cannot differentiate the opponent's current state or future action. Because of this uncertainty, a decision engine must evaluate all possible outcomes and payouts to calculate an accurate expected value of the game from every member of the information set. This means that the engine must account for every possible combination of dominoes between the target and the boneyard. This adds more computational complexity to solving the game. Applying this logic, the game instance in Figure 1 must compute the current complex problem (branching factor of 6) along with every other possible start state (or $\binom{19}{9} = 92{,}378$ starting states, including each state's branching factors) of the opponent, assuming the player hand stays the same.

Simplifying the Game

In order to solve a game of dominoes for this research, the game must be simplified. One way to simplify the game of dominoes is to use subsets of the total set of dominoes, where the total set of dominoes is 28 dominoes ranging from (0|0) to (6|6). A subset of dominoes ranges from (0|0) to a smaller value domino such as (2|2) with six dominoes or (3|3) that contains 10 dominoes. This is depicted in Table 1.

Scope

The scope of this research involves producing an algorithm that has the ability to make predictions of the opponent's strategy as well as producing an optimal move for a player agent.

Limitations

Experimentation for this research employs double-3 dominoes. A set of double-3 dominoes contains enough dominoes to evaluate two player dominoes games with partially observable information and chance nodes. For this research, the starting state of all games includes two players' hands of three dominoes with a boneyard size of four. This subset of dominoes is chosen because it is the smallest subset of dominoes that still provides enough dominoes to have a boneyard size slightly larger than one player's hand size (that is similar to play in a full set). For instance the starting state of double-6 dominoes consists of nine dominoes in both players' hands with a boneyard size of 10 dominoes. In addition, larger boneyard sizes create larger branching factors at chance nodes.

Additionally, all games end after the first round. In other words, after the first domino hand is played, the participant with the highest score wins. Playing only one round is different than establishing a predefined score and playing many rounds until that score is achieved. Since these limitations do not affect the basic rules of the game or how it is played, more computations can be applied to solve this problem.

Assumptions

Research is conducted under the assumption that there is no bluffing involved in dominoes. Bluffing in dominoes entails a player implying he or she lacks an available play by pulling from the boneyard even if he or she has and existing play. All domino rules applied in this research come from Domino Basics and Domino [2, 12]. It is assumed that all domino rules followed from this reference apply to real world domino games.

Contributions

This research project contributes in the areas of game theory and opponent modeling. The research presents empirical evidence showing that, for three different strategy opponents, there is potential and actual value gained over a game theoretic player by exploiting an opponent's actions. Furthermore, this research confirms the actual value computed by employing an MDA classifier predicting three different playing styles in the game of dominoes.

Methodology

To complete this research, a simple dominoes agent is developed to play games according to the rules of dominoes. The agent is required to facilitate every start state of double-3 dominoes between a player agent and three different opponent types. The player agent uses an opponent modeling algorithm to predict the type of the opponent. The player agent plays incorporating opponent's type into a backwards induction algorithm [25] to maximize score.

Results show the expected and actual value gained by the opponent model as well as how well it predicts the opponent type. The expected value gain of the opponent model is a calculation of the highest gain in value (over a Nash equilibrium player) that an opponent model can provide against a specific opponent strategy. This gain is computed by recording the value of applying an oracle (opponent model with 100% prediction accuracy) to the player subtracting a value computed applying Nash equilibrium decisions with no oracle. The actual value gain is the gain in value over the Nash equilibrium player while applying an opponent classifier in place of the oracle. The prediction accuracy of the opponent classifier is calculated by taking a ratio of the number of correctly predicted targets over the number of attempts to predict a specific target.

Results

Empirical results show that there is a potential value gain over a Nash equilibrium player in score for fully and partially observable environments for specific strategy types. The actual value gained is positive for a fully observable environment for score and total wins and ties. Actual value gained over the Nash equilibrium player from the opponent model only exist for score, while the opponent modeler demonstrates a higher potential to win and/or tie in comparison to a pure game theoretic agent.

Thesis Overview

Chapter II of this thesis discusses previous work in the field of game theory and opponent modeling as well as definitions of many of the concepts discussed in this thesis. Chapter III illustrates how the algorithm is created and implemented, while Chapter IV

discusses and analyses the results of the implementation and testing. Finally, Chapter V concludes this thesis and discusses future work.

II. Literature Review

In Game theory there are many ways to overcome an opponent. The following sections acknowledge concepts from previous work in the areas of game theoretic tree search and opponent modeling. This chapter also provides details of the advantages and limitations to game tree search. Furthermore, this chapter illustrates the benefits of opponent modeling in partially observable strategy games.

Fully Observable Games

Chess is a game of perfect knowledge. Games with perfect knowledge are called fully observable games [26]. Deep Blue applies game tree search in chess. In doing so, this system was able to defeat chess world champion Garry Kasperov[5, 26]. The Deep Blue agent models a chess board and makes predictions based on outcomes (in most cases) 14 game tree layers in advance. Fully observable board games like checkers, chess and go are constantly researched in efforts to find more efficient winning strategies.

With perfect knowledge, programming an agent can be as simple as implementing a game tree and performing a search to find the optimal move for each turn; however, problems (such as increased search times and stack overflows) commence as the game complexity increases. Additional issues arise when choosing the best evaluation function, which is simply the algorithm which returns the best prediction of the value of the game at a specific node. MiniMax is a search algorithm that invokes on an evaluation function and "serves as the basis for mathematical analysis of games" [26].

MiniMax Search

When working with a complex search tree to find the optimal move to make at every level of the tree, a wise decision is to limit the counter move that the opponent is capable of making for that turn. This is done by choosing the move that maximizes the player's chance of winning, while minimizing the opponent's winning chances.

According to Funge and Millington [14], it can be easy to predict chances of winning towards the end of the game based on the minimal count of plays remaining. Conversely, counting remaining plays can be more difficult in the beginning and middle of the game, because of the uncertainty of where the game ends. Therefore, an evaluation function is employed to predict how the end game states appear [14]. This function can also be called a heuristic function on a game tree search.

For this search to work, the player and the opponent or adversary are labeled the terms "MAX" and "MIN," respectively. Furthermore, as explained above, each level of the game tree represents one of these participants.

Figure 3 illustrates how the evaluation function depicts values at every node in the tree in terms of MAX and decides the most advantageous move to make. This algorithm is assessing every move possibility in the game tree with an evaluation function to predict the ending score in the leaf nodes. The algorithm then cycles this final score up through the nodes to the two possible moves that MAX can take. The node with the highest score MIN will allow is chosen [17].

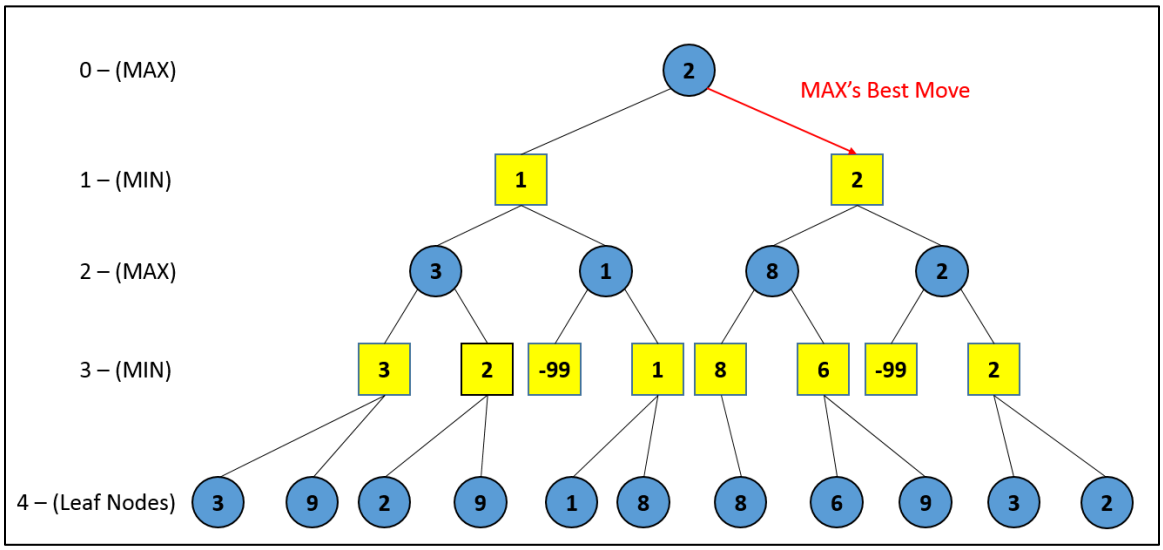

Figure 3: MiniMax game tree.

As shown in Figure 3 MAX chooses the right node at a value of 2. This assessment is completed at every turn throughout the game until a terminal (win, loss or tie) state is achieved.

The Expectimax (EM) algorithm [26] is applied to MiniMax to account for probability (chance nodes) within the game tree. Equation 3 illustrates the value calculated at a chance node within a tree with probabilistic nodes [16]. This equation states

$$Expectimax(s) = \sum_{i}^{n} P(child_i) \times U(child_i) \tag{3}$$

where $P(child_i)$ is the probability of a specific child node of s and $U(child_i)$ is the utility value of $child_i$ and n is the number of chance options (for this research, dominoes in the boneyard).

Stochastic Partially Observable Games

A game is said to be partially observable and stochastic if there is any random and hidden information. Examples of this type of game are card games such as Poker and Bridge where players are dealt random hands hidden to their opponents [26]. Furthermore, as stated in the last chapter, the game of dominoes also shares this trait.

Partially observable stochastic games take on many forms and fashions. Card games like Poker and Bridge are partially observable because participants are unaware of their opponent's cards. This concept also holds true for games like dominoes. The stochastic element is encompassed in random card distribution [26] as well as pulls from the boneyard. Non-stochastic elements of dominoes such as a fixed opponent strategy can be modeled by applying opponent modeling techniques.

Game theory and Opponent Modeling

Game theory can be augmented by modeling an opponent's future actions and/or current state. Ross's [25] definition of game theory outlines how it is applied to various applications:

> Game theory is the study of the ways in which *strategic interactions* among *economic agents* produce *outcomes* with respect to the *preferences* (or *utilities*) of those agents, where the outcomes in question might have been intended by none of the agents [25].

This means that when two agents interact with one another, even though each agent has a desired outcome, the outcome of that interaction could be in the favor of one, all or none of the agents involved. Modeling an opponent allows the player to better decide which action it needs to take to receive the best possible utility (or quantified outcome) in a

game situation. The first part of this section discusses the M* search algorithm that aids a game theoretic approach by providing an opponent model to adversary search (MiniMax). The second part discusses research in applying opponent modeling to poker. Later discussed is how neural networks are employed to learn an approximation of the opponent model that has lower memory requirements than the full opponent model in order to simplify the opponent space and expand the amount of opponents that can be modeled. Lastly is a discussion of how to rank opponent models in order to choose an optimal model.

M Search.*

This section discusses Carmel and Markovitch's M* search algorithm [6]. The goal of this search algorithm is to take advantage of the opponent's weaknesses (tendencies to make plays other than game theoretic equilibrium moves) throughout the game. To accomplish this, M* applies an opponent model to the MiniMax algorithm. Results from running this algorithm demonstrate a higher winning percentage in fully observable extensive form games.

The M* search algorithm employs an opponent model in order to exploit the opponent's weaknesses through adversarial search [6]. Assuming an accurate opponent model, the information obtained provides to the player an advantage of playing a best response to the opponent's modeled strategy instead of the best response to a presumed Nash Equilibrium opponent. This situational awareness offers the player more opportunities to win games that are otherwise unwinnable in equilibrium play. M* works by identifying instances in the game where an opponent underestimates the potential of a

certain action and chooses a move with less value, or times when the opponent overestimates the potential of a move and takes a detrimental action. These occurrences are called swindle and trap positions respectively.

The M* algorithm works similar to MiniMax by employing an evaluation function and search depth to maximize what a player can achieve in a certain play; however, the difference is that M* incorporates an opponent model to provide the specific action of the opponent. This provides the algorithm information on how the opponent may differ from the Nash Equilibrium opponent. The following definition by Carmel and Markovitch, further explains how this opponent model is applied to the algorithm.

1. Given an evaluation function *f*, P = (*f*, NIL) is a player [6] (with a modeling-level 0, or search depth of zero).

2. Given an evaluation function *f* and a player O (with modeling level *n* - 1), P = (*f*, O) is a player (with a modeling level *n*).

The modeling level of the opponent corresponds to the level in which the player models the opponent. For example, a player with a modeling level of zero does not model its opponent. The modeling level of 1 models an opponent with a modeling level of zero and so on [6].

The player P from this definition is used as the opponent model. Using P in the equation, the input of M* becomes <position, depth, (f_{pl}, P)>, where f_{pl} is the player's evaluation function and P is the opponent model [6]. The output of M* is either a value,

a state or both depending on the preference of the software designer. Figure 4 illustrates the algorithm [6].

```
Procedure M*(pos, depth, (f_pl, OPPONENT_MODEL))
  if depth = 0
    return ⟨NIL, f_pl(pos)⟩
  else
    SUCC ← MoveGen(pos)
    for each succ ∈ SUCC
      if depth = 1
        player_value ← f_pl(succ)
    else
      ⟨opp_board, opp_value⟩ ← M*(succ, depth − 1, OPPONENT_MODEL)
      ⟨player_board, player_value⟩ ← M*(opp_board, depth − 2, (f_pl, OPPONENT_MODEL))
    if player_value > max_value
      max_value ← player_value
      max_board ← succ
return ⟨max_board, max_value⟩
```

Figure 4: M* Algorithm [6].

Figure 5 shows an example game tree that illustrates the recursive calls by the M* algorithm [6].

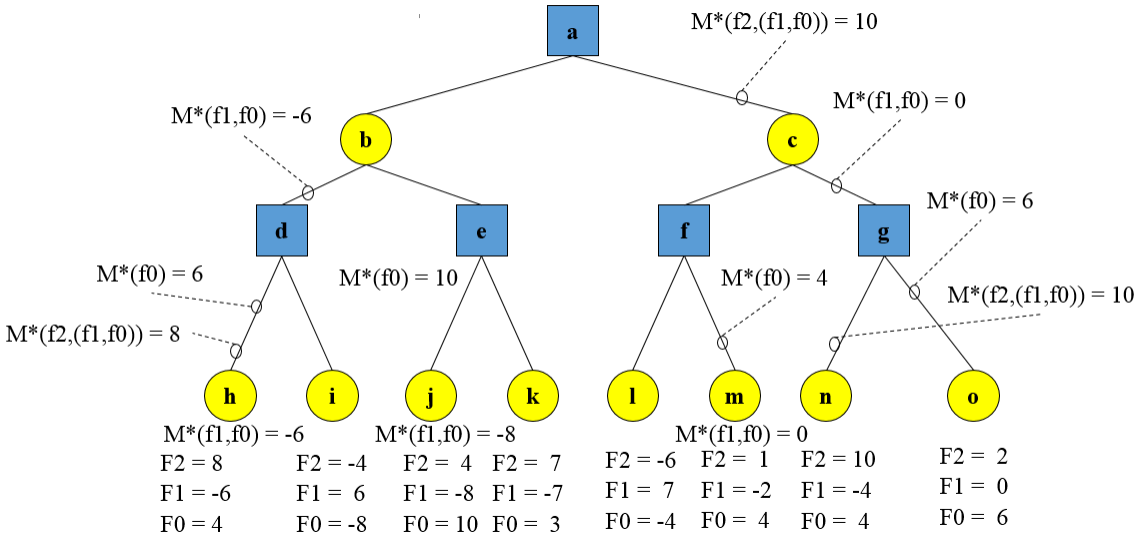

Figure 5: Game tree recursive search of M* with M*(a,3,f_2(f_1,f_0)) [6].

The example in Figure 5 illustrates how the player "swindles" the opponent by convincing the opponent that the player will select leaf node-o; however, the player actually chooses node-n. This occurs as a result of the opponent's model of the player presuming the player is using the f_0 evaluation function. Since this is the case, the opponent's model propagates a value of -6 to node-b and a 0 to node-c. Nonetheless, the player is really using the f_2 evaluation function that returns a value of 10 from node-n. This value is strong enough to propagate back to the root (through backward induction) and become the max value for the player. This value is also the highest value of all f_2 leaf node evaluations. One thing to note here is that a standard MiniMax evaluation of this game using the f_2 evaluation function yields 7. Therefore, this evaluation demonstrates that M* potentially results in higher scores than MiniMax.

Carmel and Markovitch's results from this study show that M* is more effective than a Nash Equilibrium agent at winning at checkers extensive form games. Their tests result in a higher number of wins and higher scores achieved in total against opponents in the fully observable games of tic-tac-toe and checkers.

The M* algorithm is an interesting culmination of logic in that it not only attempts to acquire the best value for the player, but it also works to exploit the weakness of the opponent. M* shares many concepts of MiniMax; however, it differs from MiniMax by employing an opponent model, that predicts moves based on the opponent's strategy. This type of logic can be used to expose higher scoring games and/or wins that Nash Equilibrium does not allow; however, this opponent modeling research example only accounts for fully observable games. The next two sections provide examples of opponent modeling use for partially observable games.

Opponent Modeling in Poker.

When researching opponent modeling for partially observable games, Poker is one of the most popular games researched. This section describes two agents that model opponents in the area of opponent modeling in poker. The first is an agent called Loki [3, 11]. This agent applies weights to possible opponent states (or hands) based on the opponent's actions on previous rounds. The second agent called Poki [10], builds on Loki by predicting potential actions the opponent will select. Both methods are shown to be effective against opponents in open forums and closed environments.

Loki is an approach to opponent modeling in poker that is "capable of observing its opponents, constructing opponent models and dynamically adapts its play to best

exploit patterns in the opponent's play" [3]. This agent applies a probability distribution to the opponent's potential cards by assigning and updating weights to those possibilities as the opponent bets, raises or calls. Statistics are then reevaluated every time the opponent takes an action updating the opponent model.

Table 2 illustrates a subset of every possible opponent hand in a hand of Texas Hold'em [3].

Table 2: Re-Weighting of Various Hands after the Flop [3].

Hand	Weight	HR	HS	~PP	EHS	Rwt	Nwt	Comments
JH, 4H	0.01	0.993	0.99	0.04	0.99	1	0.01	very strong, not likely
AC, JC	1	0.956	0.931	0.09	0.94	1	1	strong, very likely
5H, 2H	0.6	0.004	0.001	0.35	0.91	1	0.2	weak, good potential
6S, 5S	0.7	0.026	0.006	0.21	0.76	0.9	0.54	moderate, low potential
5S, 3S	0.4	0.816	0.736	0.04	0.74	0.85	0.6	mediocre, moderate potential

were, the numbers in the first column represent the number on a playing card and the capital letters represent the first letter in each suit (i.e, S is for Spades, H is for hearts). The purpose of the table is to show how each hand is re-weighted after the flop. For each possible hand the opponent modeler's algorithm calculates the initial weight (Weight), un-weighted hand rank (HR), hand strength (HS), and other factors leading to a new overall weight (Nwt) [3]. To explain how this table works, Billings gives details on QS-TS. He states that though this is a strong hand at first, when accompanied with a flop of 3H-4H-JH, the hand strength is then calculated low at 0.189 (where 1 is high and 0 is

low). This is because there are other potential hands available that can better maximize on this flop. After further calculations the algorithm yields an effective hand strength (EHS) of 0.22 for this hand. Consequently, per statistical observations of the opponent, this is lower than the hand strength they usually bet on. The potential hand is then assigned a new weight of 0.01. Furthermore, the new overall weight along with the opponent's next action is the key predictor of whether the opponent has that hand or not.

When played 100,000 games against controlled setup models, Loki was ahead by approximately $5,000 while the other models were down $550. Loki also does well in online poker play against humans; however, not enough information is present to show that it can outperform previous best programs [3].

Poki is an AI poker agent that also plays on an online poker server with human players. It is used to store game data for poker research [11]. This application is a successor to Loki.

One improvement made in this application simplifies the opponent modeling process by eliminating the re-weighting table and an artificial neural network (ANN). By eliminating the re-weighting approach, this framework alleviates the burden of accounting for a number of active opponents and betting positions. This reduces the systems labor factor [11]. Furthermore, the Poki models specific opponents which allow it to make more informed decisions.

An ANN is a machine learning algorithm that loosely represents a biological neural structure, or brain [11]. Davidson also states that neural networks "typically

provide reasonable accuracy, especially in noisy domains. However they rarely can produce better results than a more formal system built with specific domain knowledge."

Figure 6 employs the inputs from Table 3 [10]. The ANN in Poki is trained by applying back-propagation [10] (or supervised learning process for ANNs).

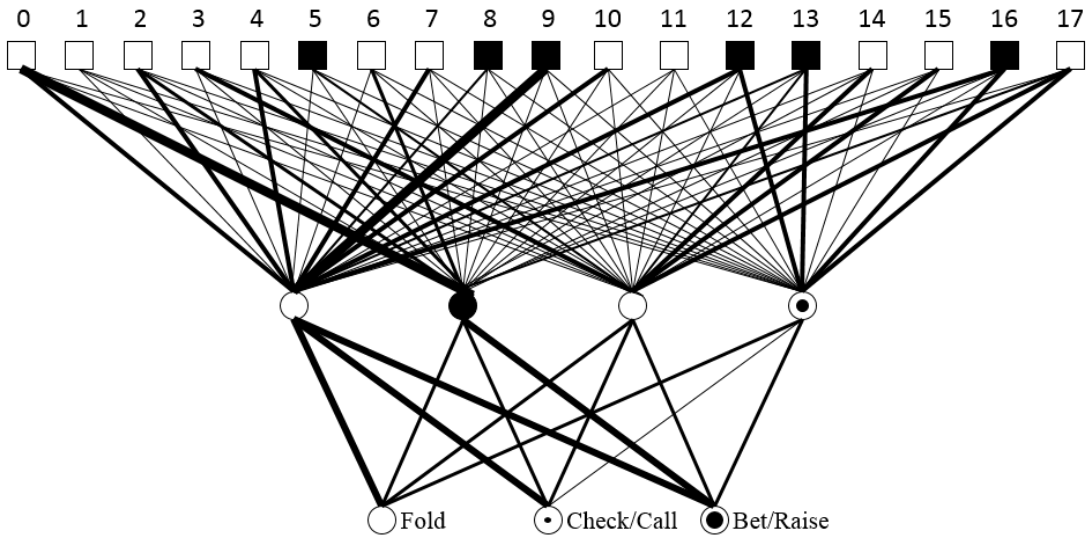

Figure 6: A neural network predicting on opponent's future action [10].

The figure illustrates how the neural network makes use of the 17 inputs in Table 3 and makes a decision on whether an opponent will Fold, Check/Call or Bet/Raise [10].

Table 3: Neural Network Inputs [10].

#	Type	Description
0	real	Immediate pot odds
1	real	bet ratio
2	bool	Committed
3	bool	one bet to call
4	bool	two or more bets to call
5	bool	betting round = turn
6	bool	betting round = river
7	bool	last bets called by player > 0
8	bool	players's last action ws a bet or raise
9	real	0.1 X num Players
10	bool	active players is 2
11	bool	player is first to act
12	bool	player is last to act
13	real	estimated hand strength for opponent
14	real	estimated potential strength for opponent
15	bool	expert predictor says they would call
16	bool	expert predictor says they would raise
17	bool	Poki is in the hand

Poki was able to routinely predict opponent actions with an 80% accuracy, while sometimes achieving accuracies of 90% [11]. While Poki makes a significant improvement to state only predictions, it requires a significant amount of data to function properly against a diverse set of opponents [20]. The next section describes how opponent models can be generalized creating a more robust agent.

Generalizing Opponents to Simplify the Opponent Space.

This section describes how opponent modeling can be generalized to use in game theoretic solutions of a partially observable extensive form game. The partially observable extensive form game covered in this section is called Guess It [19, 20]. Because of hidden game state information in partially observable games, game theoretic

solutions such as MiniMax become difficult or non-useful to apply. To make this information visible, predictions must be made using techniques such as opponent modeling. "Opponent models are necessary in games where the game state is only partially known to the player" [20]. At the same time, many opponent models are used to train against specific opponents deeming it time consuming to train an agent against multiple different opponents. Locket, et al solve this problem by generalizing opponent types in Guess It allowing for an opponent model to play against many different opponents without training against opponent-specific models [20].

Guess It is a game where each player is dealt an equal number of cards (usually 6), and one card is faced down on the table for players to guess. Participants cannot see each other's cards making this a partially observable game. The goal is for participants to figure out the identity of the card faced down by taking turns obtaining and giving information about each other's hand. In each turn, players have three possible actions. The first is to identify the faced down card. The second possible action is to "ask" the opponent for a card that player does not have. Players can also "bluff" by asking for cards that they already have. The participant who identifies the hidden card wins, while any false identification of the hidden card is an automatic loss.

Game theoretic solutions such as MiniMax and other variants of MiniMax make it simple to find the best move in many extensive form games; however, hidden information in Guess It complicates play because the game state is not as obvious. Other game theoretic solutions include partially observable Markov decision processes (POMDP's) though, for games with more than just a few states, POMDP solutions

become intractable [24]. This makes opponent modeling a great resource for solving games with hidden information.

Opponent modeling inputs an opponent's previous actions to a learning engine to help predict the opponent's state or next action. In many cases opponent models learn through machine learning tools such as classification algorithms or neural networks. These tools are used to predict opponent characteristics by using game state features as inputs. Opponent characteristics are associated to classes identified by features entered into the tool. Each model is opponent-specific, meaning that for each opponent, a new model must be developed. This becomes a time management challenge for training for several opponents.

Lockett solves the specific opponent problem by generalizing opponents in the game of "Guess It" and employing mixture models. A mixture model in this research is a probability distribution over a "cardinal set" of opponents [20]. In this research, the cardinal set contains four different opponent-type categories. Each one of these types has high potential to defeat another specific type as long as that type is identified. A mixture model identifier employs neural networks to predict player-types through a supervised learning process. From here, the agent can make a decision based on which opponent it is playing. Experimentation was conducted using a mixture model based player and a controlled setup player.

The four opponent-type categories of the cardinal set consist of Always-Ask, Always-Bluff, Call-then-Ask and Call-then-Bluff. Always-Ask never calls or bluffs, while Always-Bluff never asks or calls. The Call-then-Ask agent attempts to name the

center card if the other player asks for a card the Call-then-Ask agent lacks, otherwise it asks, while Call-then-Bluff calls every potential bluff; otherwise it bluffs [20].

In the cardinal set, each player-type has an effective counter measure for another specific player type. For example Table 4 illustrates all the hierarchy of all strategies. Each strategy is able to defeat one of the other strategies.

Table 4: Guess It Strategy Hierarchy Matrix [20].

Strategy	Defeats
Always-Bluff	Call-Then-Bluff
Call-Then-Bluff	Call-Then-Ask
Call-Then-Ask	Always-Ask
Always-Ask	Always-Bluff

As long as the agent correctly models which opponent-type it is playing, it can determine which strategy to use against it. This serves as the basis of generalizing player types.

Figure 7 illustrates the block diagram of mixture based architecture developed by Locket, et al. [20]. The model includes two modules of integration. The mixture identification module accepts the game state as an input and identifies a mixture. The mixture in this diagram is a probability distribution over all possible opponents in the cardinal set. The decision module accepts the mixture as well as a board state and makes a decision on whether to bluff or call. The default move is to ask [20].

The controlled setup player in this experiment employs only the decision module. While there is the absence of the mixture identification module, the controlled setup has an identical training and validation regimen. The absence of a mixture identification

module in the controlled setup allows for testing the performance of the mixture identification module of the mixture based player.

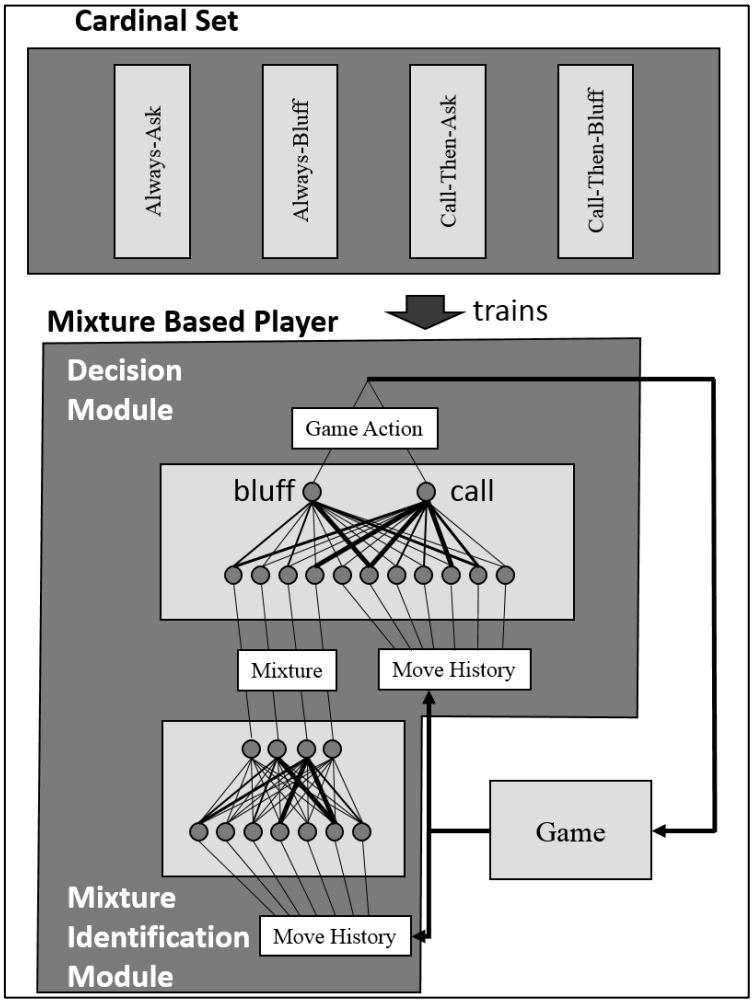

Figure 7: Mixture-Based Architecture for Opponent Modeling in Guess It [20].

By generalizing opponent-types in a mixture model, the agent was able to win an average of 71.3% of 220,000 games played against 11 diverse opponents, while the controlled setup won an average of 57.7% [20]. This diverse set of opponents included the cardinal set opponents as well as unknown opponent types. When playing against all unknown player types, the mixture model based player won an average of 61.5% games,

while the controlled setup player won an average of 54.6% of its games. In addition, when played 20,000 games against the control player, the mixture model won an average of 77.6% of the games. Lockett, et al. also state that all results are statistically significant with to $p < 0.05$ [20]. This shows that generalizing an opponent has the ability to apply opponent modeling on a broader scale without training against every different opponent.

Environment Value.

The discussion in this chapter involves many different methods of opponent modeling. The question to ask is which type is the best to use for an agent in a specific environment. This section bridges the gap into which model will be optimal to use compared with other opponent model types.

Opponent modeling has the potential to improve a player's expected score or winning potential in a game depending on the environment in which the game takes place [4]. This value of a game is improved by utilizing two classes of agent models. The first model type is the opponent's state in the game and the second model is of the opponent's actions. Given an accurate model of both of these features improves the value of the game to the player's favor. The environment value of an opponent model is the game theoretic value improvement that a particular opponent model will provide an agent given the environment. This environment value indicates an "upper bound to the potential performance improvement of an agent" [4]. This section explains how these features (if accurately modeled) help to improve the game theoretic value of a game.

Borghetti states that an agent model is a function or method that "predicts something" about another agent [4]. As stated above, agent models can be employed to

predict information such as the state and/or the action of another agent or opponent. This information is then used to select optimal moves. Optimal moves are selected using game theoretic algorithms such as MiniMax or the probabilistic variant Expectimax (EM) [26].

The state of a game involves all information about the game at a particular instance in time. In computer science parlance, the state of a game can be compared to a node on a tree. The class of state opponent models provides a probability distribution over all possible states [4]. For example, for any number of possible states that an opponent can belong to, the probabilities of all states must sum to one.

This logic is also true for the class of opponent models that predict the actions of an opponent. Actions are classified as the strategy or move that the opponent (or any participant in the game) will make at a given state in the game. As explained in the previous paragraph, this model class provides a probability distribution over all actions.

The environment value (V_Lambda) is the maximum improvement of the game theoretic value given a certain environment. This value is the difference between an environment using a perfect model and environment using no model at all. Equation 4 [4] illustrates this explanation.

$$V_{\Gamma}(M) = U(M \mid \Gamma') - U(M \mid \Gamma) \tag{4}$$

Where:

Γ – The original game with no opponent model

Γ' – The transformed game with a perfect opponent model

$U(\mathrm{M})$ – The utility gained in a game

M – A particular model that provides a probability distribution over actions or states

Borghetti states that given an environment, an upper bound can be developed for an environment to use as a baseline for any opponent model (in its class) to follow [4]. This upper bound baseline is developed by employing a perfect information agent model or oracle to provide an optimal state or action set. The oracle is employed to display the maximum potential that an opponent model can deliver. Assuming MiniMax (or similar algorithm) is employed and all play is rational, if a model cannot do any better, the resulting value is the Nash Equilibrium.

The precondition to finding the environment value is that the Nash Equilibrium of a game can be found. For this research, this requires solving the game of dominoes, which is not possible with today's computers. "When this precondition does not hold, we may be better off approximating the environment value using an estimation of the distribution over likely opponents." [4]. This research focuses on finding the expected potential value gained from the information an action opponent model can provide against specific opponents. Therefore the environment value is not calculated.

Game theory has many applications that benefit from opponent modeling today. The goal of applying opponent modeling is to provide information on the current state or future actions of the opponent. Carmel and Markovitch's [6] M* search provides a lot of insight on opponent actions to show where to capitalize on opponent mistakes in a fully observable environment. If there is a way to employ M* to a partially observable environment, this opponent modeling algorithm could add value to dominoes. Billings and Davidson, et al. [3, 11] have made vast improvements to how poker is played online by applying models to predict opponent states and actions; however, this research requires a lot of data processing to be optimal. Locket [20] solves this problem by generalizing opponents into a cardinal set (or two-dimensional space of opponents defined by the probability of calling or bluffing [20]), in which each decision is made by applying the action of the generalized model. If opponents in dominoes can be generalized, applying opponent modeling to dominoes should be a quicker and less data intensive practice. Lastly, Borghetti shows how to choose the best opponent model for specific environments. Any opponent modeler applied in current two player dominoes research should apply Borghetti's technique as a baseline.

Other Methods.

Donkers's [13] probabilistic opponent model search or PrOM search involves computing a probability distribution over several opponent types in order to make a decision. PrOM search applies an approximation over several strategies, whereas this research will apply the exact strategy of the opponent.

Cowling [9] creates an ensemble method for determination of Monte Carlo Tree Search (MCTS). Cowling's research involves applying MCTS and upper bounds on Trees (UCT) to solve probabilistic games with hidden information. The research applies these methods to the game of Magic: The Gathering in order to study many different heuristics that can solve this complex game. While dominoes has probability and hidden information, the focus of this research is not to develop heuristics to solve dominoes as the simplified version of the game allows for a search to the leaf nodes of the game. The main focus of opponent modeling for this research is to predict target's strategy and attempt to find an optimal move on a shallow game.

Dominoes AI Agents.

Although there are many online dominoes games available, academic research has been completed in creating dominoes graphical user interfaces in C++, Java and BASIC. Versions have been found that employ each language respectively. The C++ version is a single game GUI entitled Domino 1.2.0 [23]. This version is created by using the Qt Creator Library. The Java version Badomino [18] is also a single game domino interface. Smiths [28] BASIC version is a game that employs strategy tables and learning techniques to make optimal moves.

Domino 1.2.0 provides a user interface that plays a single hand of double-6 dominoes while keeping score. It also tells the user who wins the game at the end. While the project is completed in a familiar language and library there are some limitations. The first limitation is that most of the code and comments are in Russian. While the code provides a useable GUI, the code is difficult to decipher. There is also no way to scale

the code from double-6 to a smaller (or larger) set of dominoes for experimentation purposes. Lastly, there is no experimental control to run many games in sequence.

The main focus on the Badomino project is to build a well working domino graphical user interface employing AI logic; however, more emphasis is placed on the goodness of the GUI itself. The AI module employs myopic strategies; however, it does apply AI to determine the best myopic strategy to employ. Furthermore, this is also a single game system that is not scalable for smaller or larger sets of dominoes.

Smiths learning algorithm involves strategy learning. This technique applies a model that plays hundreds of games against itself to learn many situations of the game in order to make an optimal move against other opponents. Smith's research applies strategy tables to avoid applying techniques used for fully observable games. This research applies opponent modeling and roll-out techniques to overcome many of the barriers involved with partially observable games.

Conclusion

Game theory is a complex field in which research is conducted implementing search trees and other graphical figures to play games at an optimal level. When traversing a tree in adversarial games, MiniMax search implements an evaluation function that estimates the value of the game at that point in order to optimize every action. The drawback of this search is that its purpose is to work in a fully observable environment. Opponent modeling is used in many in many applications in order to produce a better estimate on imperfect information to make the best prediction. The next

section explains how opponent modeling will be used in two-player dominoes to make the best prediction in a stochastic partially observable environment.

III. Methodology

Introduction

This chapter describes the methodology of how a dominoes artificial intelligence agent employs opponent modeling to achieve improved knowledge of the opponent strategy in order to optimize the decision making process. The chapter starts with the definition of the problem. Next, the experimental setup is explained. Lastly, this chapter explains how each agent of the experimental setup is tested.

Problem Definition

Goals and Hypothesis.

The goal of this research project is to show how well modeling an opponent's strategy can improve a player's decision in two player dominoes. When an opponent's strategy is unknown (assuming all players are rational), the optimal strategy is to achieve a Nash equilibrium through an adversarial search. MiniMax (a type of adversarial search) can be applied to this research by creating a game tree representation for the game of dominoes. Figure 8 illustrates a simple game of dominoes where the player and opponent both pull three random dominoes from a full set of double-6 dominoes. Each board picture represents a node on a game tree. The position on the game tree is called the game (or board) state. This tree represents all possibilities of playing this game instance between the two players.

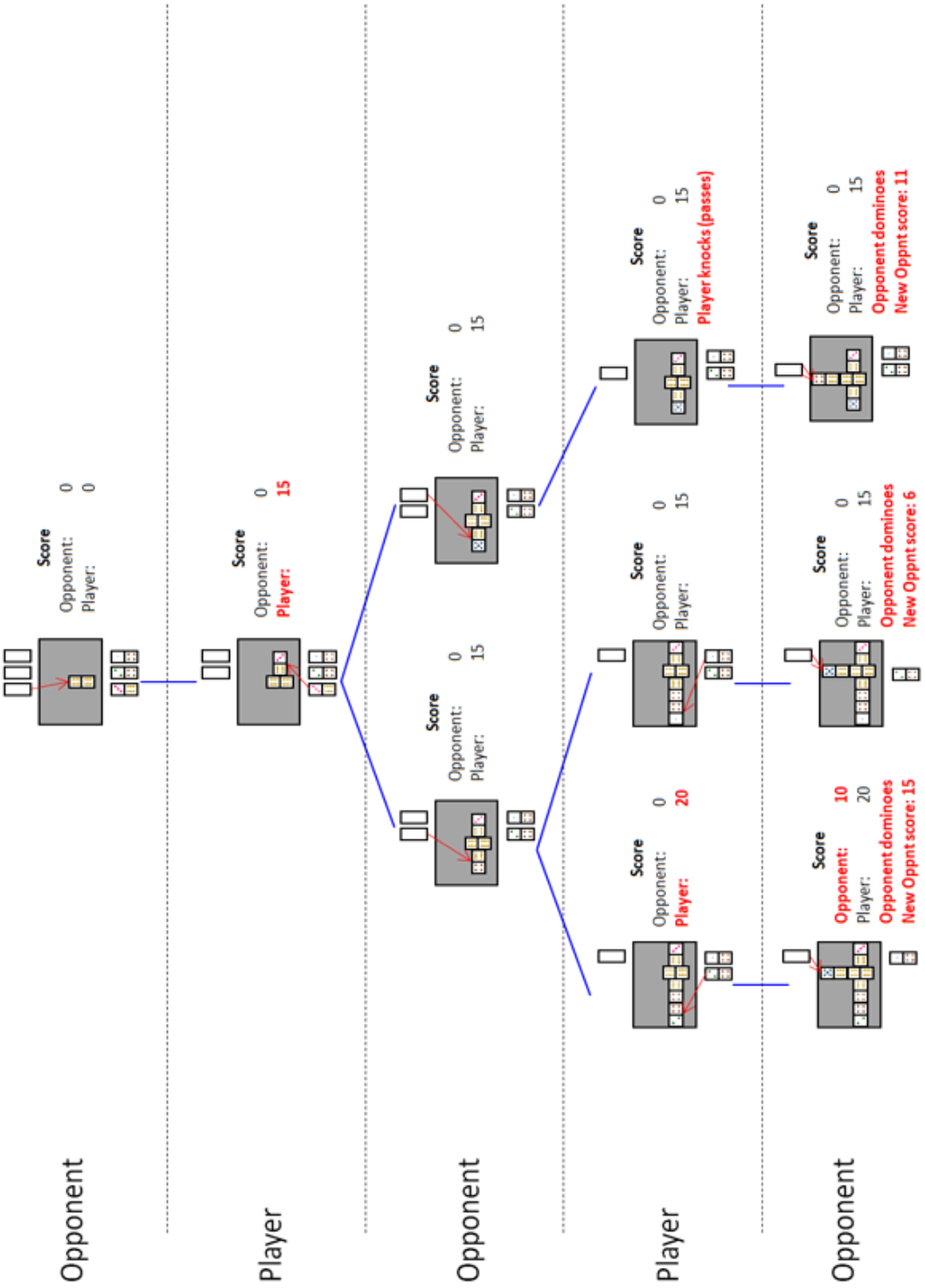

Figure 8: Game tree representation of dominoes.

If the opponent plays with a non-rational strategy, another equilibrium can be achieved that capitalizes on non-rational actions made by the opponent. This is achieved by applying M* search [6], which replaces MIN's strategy with an opponent model. In order to use M* search, an opponent model (or strategy) must be identified. This concept generates the research question "how does M* search perform in the game of dominoes?"

In addition to finding and evaluating the opponent model, another challenge arises in applying M* search to a partially observable game. The issue with applying a search algorithm to a partially observable game is that the level of certainty in the location on a game tree is much worse than in a fully observable environment. Without accurate board state data, potential exists for the search method to traverse a game tree with an erroneous game instance, reducing the possibility of finding an optimal play.

Approach.

To solve these problems, this project centers around a hierarchal approach that focuses on employing an opponent classifier to predict opponent types [27]. Then a decision engine for the player agent uses the estimated opponent's strategy in M* search. To handle the effects of hidden information, Monte Carlo sampling simulates iterations through multiple possible game states according to the actual probability distributions of unknown dominos in the boneyard and the opponent's hand.

The hierarchal approach to this decision system involves first predicting an opponent type, then applying the opponent's strategy to M* search. Opponent types from here on will be defined as "targets." There are three targets identified for this research.

Each target applies a different strategy to play the game. The first target is the random-play agent, which makes legal plays of pseudo-randomly selected dominoes at its turns. The second target agent is the myopic scoring target, which tries to score the most points at each play without considering consequences on future plays. The final target plays myopically defensive by blocking the player from taking an action when there is an opportunity. The myopic defense target agent also keeps the board at a low scoring level to deny the player from achieving high scores during turns.

The random target agent plays legal moves in a pseudo-random fashion. This target selects its plays from a list of legal moves; however the move is selected at random, with equal likelihood for any legal move. The seed for this random number generator can either be generated based on the system clock for individual games or a game iteration number for experimentally controlled games during repeated testing.

The myopic scoring target agent is a greedy scorer that selects a legal move that scores the highest from all legal moves with no concern for the future value of the move later in the game. If a tie arises between possible moves or there is no score available at that play, the myopic scoring player plays domino that produces the highest board count to increase high scoring opportunities for future plays. (The board count is the sum of all of the outside numbers of the domino chain.)

The myopic defense target agent's goal is to reduce the number of plays available to its opponent as well as prevent an opponent from obtaining a high score. This agent selects actions that limit the player's legal plays on the next turn. Figure 9 shows an example of board states from least favorable to most favorable for the defensive target.

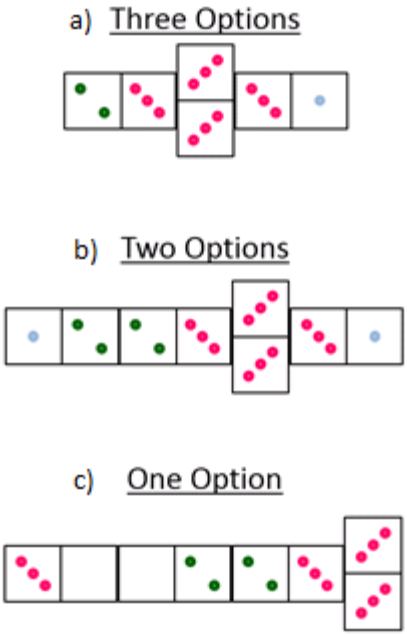

Figure 9: Play options available for the player based on target's defensive strategy.

In this example a)'s playable options (from left to right) are 2, 3 and 1 giving a player three options to choose from. b)'s playable options (from left to right) are 1 and 3 allowing for two playable options, while c)'s only playable option is 3. Limiting playable options restricts the opponent's available actions, therefore creating a possible pull or pass instance. Furthermore, if there are no available defensive plays or a tie in defensive options, the defensive target plays the domino that produces the lowest board count to keep all possible scores low.

The player agent computes the M* search decision before each play. The opponent model used in M* is the same strategy that one of the targets will apply. Since M* is a variant of MiniMax search [6] , it maximizes the player's decision based on

predicted decisions of the target. Monte Carlo rollouts are applied to the M* search to iterate through many possible board state situations. These rollouts provide the decision engine with an expected game state outcome in order for the player to make an optimal decision in an uncertain environment, given that the opponent model matches the true opponent.

Testing.

Testing this approach involves measuring the prediction accuracy of an opponent classifier as well as measuring how well M* search performs in the game of dominoes against specific player types. The accuracy of the opponent classifier is evaluated by applying a confusion matrix of all target predictions. Table 5 illustrates the data representation of this matrix.

Table 5: Confusion matrix of the opponent modeler.

		Predicted Target		
		Random	**Myopic Scoring**	**Myopic Defense**
Actual Target	**Random**	***A***	*B*	*C*
	Myopic Scoring	*D*	***E***	*F*
	Myopic Defense	*G*	*H*	***I***

Capital letters represent the number of predictions computed by the opponent classifier for a specific target. The rows indicate the actual target type (or labels of the sample data) and the columns represent the predicted target type. The letters in the diagonal (A, E and I) represent all accurately predicted data while all other letters

represent the inaccurate predictions [15]. The accuracy of the opponent classifier is computed by applying the table values to

$$(AC) = \frac{A + E + I}{S_{total}} \tag{5}$$

where AC is the accuracy, A, E and I are the numbers of accurately predicted targets and S_{total} is the total number of targets evaluated. Furthermore, when applied to the opponent classifier, the testing prediction accuracy of the classifier is calculated by

$$AC_{OM} = \frac{p}{S_{turns}} \tag{6}$$

where AC_{OM} is the prediction accuracy of the opponent classifier, p is the number of correctly predicted targets and S_{turns} is the total number of turns predicted by the opponent classifier.

To test the potential and actual performance of an opponent classifier with the M* search decision engine, an all-knowing oracle is introduced. The oracle is a method that predicts the target's state and future actions with perfect accuracy. The oracle is applied to the M* decision engine in place of the opponent classifier to predict the opponent's type. A set of games is played and all decisions made by the M* decision engine with the oracle are verified to validate M*'s performance.

The expected value gain (EVG) of M* with an opponent type over the Nash equilibrium player is calculated applying

$$EVG = U(M^*_T \mid T) - U(EM \mid T) \tag{7}$$

where T is the target and M^*_T is M* search with the target's strategy, and EM is Expectimax. The value gained by applying the M* search decision engine is calculated with both a perfect information and imperfect information environment. The perfect information environment calculates the theoretic upper bound potential of the M* search decision engine with an opponent type. The imperfect information environment calculates the actual upper bound value of the engine.

The actual value of the M* decision engine with an opponent model is calculated by applying the opponent model to the engine. This value is a representation of how well the opponent model predicts the target. Similar to the expected value gain calculations, the actual value gain is calculated in perfect and imperfect information environments to show perfect state information and real world conditions respectively.

Table 6 illustrates a four quadrant matrix of desired results for testing the M* algorithm. Rows of the matrix represent the state information given for all games played. Columns represent the given target predictor. With this configuration, quadrant one and two show a perfect state information environment, while quadrants three and four apply Monte Carlo Rollouts (MCR) to account for hidden information. Quadrants one and three are EVG calculations while two and four are actual values.

Table 6: Quadrant chart of experimental results.

		Target Information	
		Target Oracle	**OM**
State Info	**State Oracle**	**1. Expected Value Gain** Upper Bound	**2. Actual Value**
	MCR	**3. Expected Value Gain**	**4. Actual Value** Real World

The perfect information environment yields the upper bound of M*'s potential for environment and actual values. The target oracle predicts a target's strategy with 100% accuracy. This allows the M* engine to apply the correct target type at every play of every game, therefore each game reflects an optimal decision at every play with respect to M*'s capabilities. Likewise, when applying perfect state information to the Expectimax engine, Expectimax plays optimal with respect to its capabilities. Thus, the maximum gain is calculated by taking the difference in value between both engines with respect to score differentials or total games won/tied.

Dominoes Experimental Setup

This section describes the experimental setup that answers the question of whether an opponent model can improve a player's decision in the game of two player dominoes – and by how much is the improvement. The setup consists of an adversarial search decision engine as well as three target agents. The adversarial search decision engine consists of the M* and Expectimax search algorithms. The player agent employs

this engine when playing against each of the three targets in the setup. The next two subsections describe these methods.

Adversarial Search Decision Engine.

The player agent consults the adversarial search engine at every play. This decision engine makes decisions through the application of M* and Expectimax. M* is experimental setup for this research. This algorithm is applied to make optimal decisions against a specific (non-rational) target throughout the game. Expectimax the controlled setup. This engine is applied to make game theoretic plays for a Nash equilibrium solution.

Since dominoes is a partially observable game, play decisions are made by running the adversarial search algorithm through several Monte Carlo rollouts over possible partitions of the unknown states between the boneyard and the target. These simulations allow the agent to make a probabilistically weighted decision based on the expected value of several possible game states. Values of each player action are averaged over the all simulations. This results in an expected optimal action for the player at every play. The decision engine then selects the action with the highest expected value.

Expectimax is a probabilistic variant of the MiniMax adversarial search algorithm. The added benefit to applying Expectimax to dominoes is that it accounts for chance nodes. Chance nodes occur in dominoes when the player or target pulls a domino from the boneyard. The Expectimax algorithm then returns the expected value at that

turn depending on how the game terminates at each chance pull. Equation 8 represents how the expected value is determined at each chance node.

$$v_{chance} = \sum_{i=0}^{n-1} P(move) \times v \tag{8}$$

where *v* represents the value for a specific pull from a set of dominoes in the boneyard, *P(move)* represents the probability of a domino from the boneyard and *n* represents the number of boneyard pulls. The input for Expectimax algorithm is the board state. Figure 10 illustrates how Equation 8 is implemented within the Expectimax algorithm. The output from Expectimax is the play with the highest expected value at a specific chance node.

```
function EXPECTIMAX_DECISION(state) returns an action
        return arg max_{a ∈ Actions(s)} MIN_VALUE(RESULT(state, action))
function MAX_VALUE(state) returns a utitly value
    if no actions then v ← MAX(v, CHANCE_VALUE(state))
    else if TERMINAL_TEST(state) then return UTILITY(state)
    v ← −∞
    for each a in ACTIONS(state) do
        v ← MAX(v, MIN_VALUE(RESULT(state, action)))
    return v
function MIN_VALUE(state) returns a utitly value
    if no actions then v ← MIN(v, CHANCE_VALUE(state))
    else if TERMINAL_TEST(state) then return UTILITY(state)
    v ← ∞
    for each a in ACTIONS(state) do
        v ← MIN(v, MAX_VALUE(RESULT(state, action)))
    return v
function CHANCE_VALUE(state) returns a utitly value
    if Player(state) = MAX
      then return Σ_r P(r)MIN_VALUE(RESULT(state, action)))
    if Player(state) = MIN
      then return Σ_r P(r)MAX_VALUE(RESULT(state, action)))
```

Figure 10: Expectimax algorithm [16, 26].

To minimize computation, the dominoes Expectimax algorithm only employs Expectimax when there is more than one play to select. When there is one action or fewer to choose from, running adversarial search will not yield a different result.

The M* algorithm is also a variant of MiniMax (or Expectimax in games with probability); however, it also takes into account the adversary's (or target's) strategy. For

the game of dominoes, the target's chosen move is the only chosen action for every target play. This differs from Expectimax because Expectimax searches through all moves of the target. Figure 11 illustrates how the MIN-VALUE function runs a TARGET-DECISION method to obtain the target's decision. Each *TARGET_TYPE(state)* method is defined in the section labeled "Targets"

function $M^*\_DECISION(state)$ **returns** *an action*
    return $\arg\max max_{a \in Actions(s)} MIN\_VALUE(RESULT(state, action))$

function $MAX\_VALUE(state)$ **returns** *a utitly value*
    if *no actions* **then** $v \leftarrow \text{MAX}\ (v, CHANCE\_VALUE(state))$
    else if $TERMINAL\_TEST(state)$ **then return** $UTILITY(state)$
    $v \leftarrow -\infty$
    $action \leftarrow TARGET\_TYPE(state)$
    $v \leftarrow \text{MAX}\ (v, MIN\_VALUE(RESULT(state, action)))$
    return v

function $MIN\_VALUE(state)$ **returns** *a utitly value*
    if *no actions* **then** $v \leftarrow \text{MIN}\ (v, CHANCE\_VALUE(state))$
    else if $TERMINAL\_TEST(state)$ **then return** $UTILITY(state)$
    $v \leftarrow \infty$
    $action \leftarrow TARGET\_TYPE(state)$
        $v \leftarrow MIN(v, MAX\_VALUE(RESULT(state, action)))$
    return v

function $CHANCE\_VALUE(state)$ **returns** *a utitly value*
    if $Player(state) = MAX$
      then return $\sum_r P(r) MIN\_VALUE(RESULT(state, action)))$
    if $Player(state) = MIN$
      then return $\sum_r P(r) MAX\_VALUE(RESULT(state, action)))$

Figure 11: M* Algorithm for dominoes [6, 16, 26].

The computational benefit from only applying the target's decision is that the search engine only explores branches of the target's decisions. This pruning method not only

speeds up computation but it also provides a more exact representation of the target's decisions (assuming an accurate opponent model). Figure 12 (created with Gambit game theory software [21]) illustrates a game instance in which Expectimax evaluates all possibilities applying a Nash equilibrium solution to the game instance. Figure 13 (created with Gambit game theory software [21]) illustrates how the M* algorithm only focuses on the actions that the myopic scoring target actually takes. Applying M* to the same game tree reduces the tree from 18 leaf nodes to 3 leaf nodes. The dashed ovals in Figure 12 represent the branches chosen in Figure 13 . All other branches are pruned in this computation.

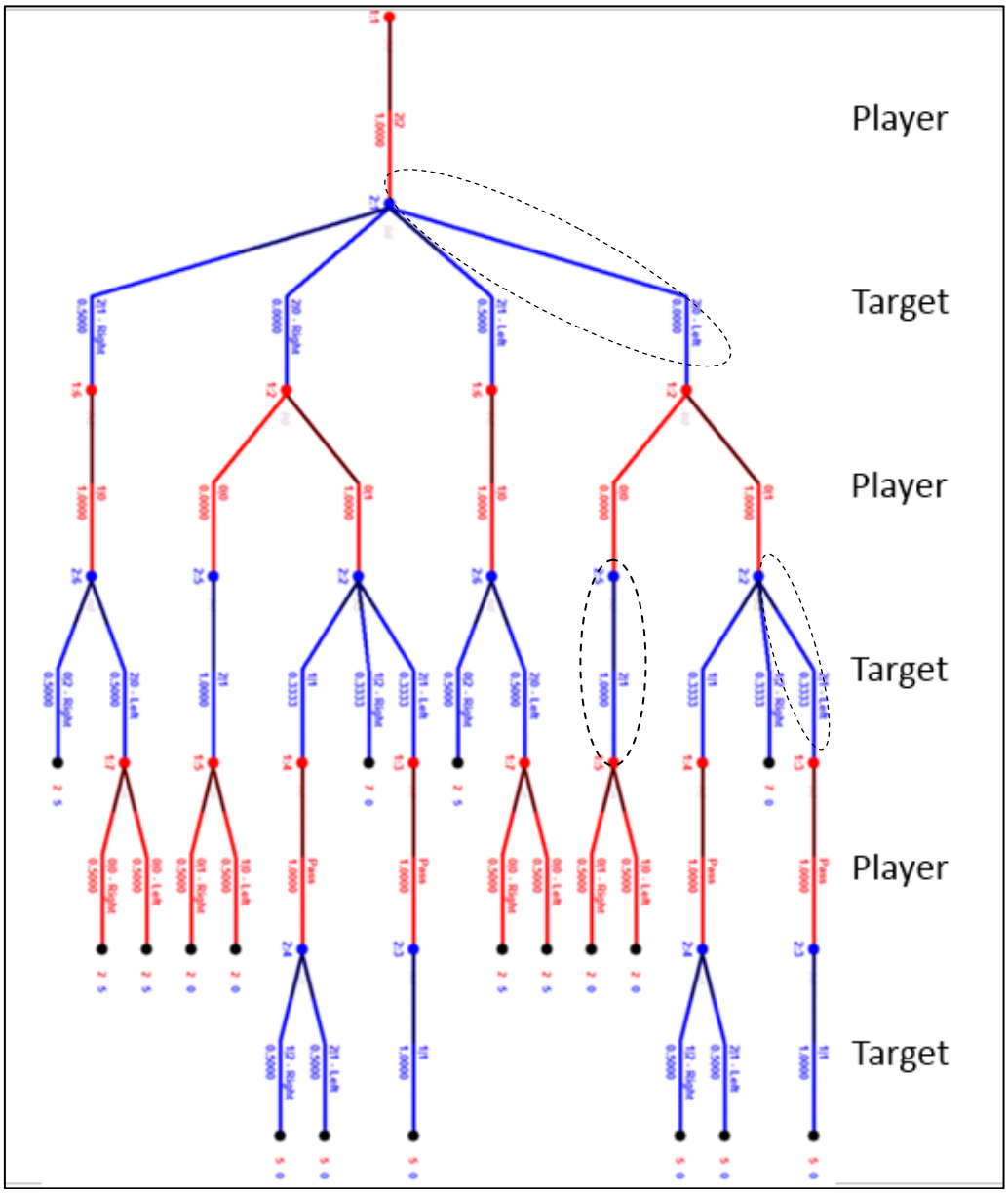

Figure 12: Game tree before M* pruning process.

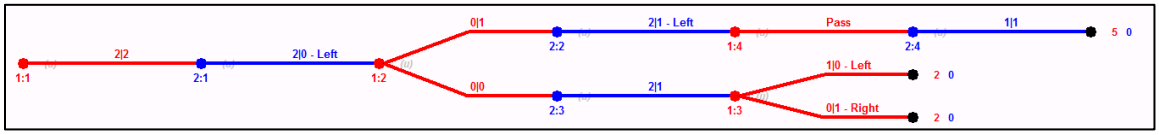

Figure 13: Game tree after M* pruning process.

The adversarial search engine is designed to provide a Nash equilibrium decision with Expectimax computations. This engine is also applies M* search to choose a best response against specific targets as well as reduce the computational complexity of the game (assuming myopic targets). In order for M* to predict the behavior of the target, the target decision algorithm must be established.

Targets.

This sub-section describes the three targets that play against the player agent. Each target starts with a set of legal moves to choose from (if there is a move to make). From here, they make the best decision based on stationary methods. The random-play target agent plays a uniformly randomly chosen domino out of the legal moves that it can play. The myopic scoring target plays the domino in the position that scores the highest. The defensive target plays the domino that hopefully minimizes the opponent's opportunity to score in the future.

The random target agent makes its choice by using a pseudorandom number generator. This seed for the generator is generated at the beginning of the game. This method chooses a random number between 0 and the size of the target's (pre-generated) possible moves list minus 1. It then selects from the list of dominoes, the domino that has an index that matches the random number chosen.

The myopic scoring agent method evaluates every possible move and selects the highest scoring move. In the case of a tie, the myopic scoring agent plays the domino that sets the highest board count. Figure 14 illustrates the algorithm for the myopic scoring target.

```
function MYOPIC_SCORING_TARGET(state) returns action
    for each a in ACTIONS(state) do
        resultsList ← RESULT(state, action)
    scoringAction ← highest scoring action in resultList
    highestBoardStateAction ← action resulting in the highest board state
    if scoringAction = highestBoardStateAction then return highestBoardState
    else return scoringAction
```

Figure 14: Algorithm for myopic scoring target.

The defensive scoring target makes plays that create a high defensive stance for the target. This is completed by computing a unique defensive score or utility value for defense. The play with the highest defensive score is the play that the defensive target makes.

The defensive score is calculated by counting the number of available matching values on the outside dominoes of the board and subtracting this number from four (since there are at most four playable locations on the game board). Figure 16 (lower cell) illustrates the algorithm for calculating this value. This value is an integer between 0 and 3, where 0 indicates four available positions to play on and 3 indicates only one available position to play on.

Having a high defensive score inversely proportional to the number of matches on the board is important because higher values indicate there are fewer possible actions for the player. Fewer actions for the player entail blocking potential for the defensive agent.

Figure 15 illustrates an example of how the defensive score is used in making a defensive decision.

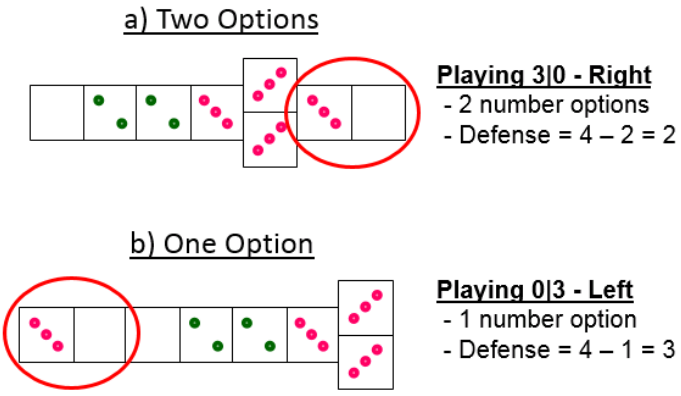

Figure 15: Example of defensive score calculation to make a decision.

In this example the target has an option of playing the (0|3) domino on the left or right side of the domino board. If the domino is played on the right side there are 2 numbers available to play on by the player. The defensive score, in this case, is 2. Playing the (0|3) on the left side leaves only one option and the score calculated this option is 3, which is a higher score than playing on the right. Having less numbers to play on lowers the probability of having a play. Figure 16 illustrates the algorithm for completing this task.

```
function MYOPIC_DEFENSE_TARGET(state) returns action
        for each a in ACTIONS(state) do
                resultsList ← RESULT(state, action)
                defenseScoreList ← DEFENSE(state, action)
        defenseAction ← highest defensive scoring action in resultList
        lowestBoardStateAction ← action resulting in the lowest board state
        if defenseAction = lowestBoardState
                then return lowestBoardStateAction
        else return defenseAction
```

```
function DEFENSE(state, action) returns a value
        for each domino in outSideDominoes do
                outsideValuesList ← values on outer sides of dominoes
        remove duplicate integers from outsideValuesList
        defense ← 4 minus size of outsideValuesList
        return defense
```

Figure 16: Algorithm for defensive scoring target.

Each target employs its own strategy in order to test the quality of the opponent modeler with certain types of play. The next section describes how the opponent modeler is fabricated in order to distinguish between each target agent.

The Opponent Classifier.

The opponent classifier employs machine learning to predict targets. Machine learning tools for this research are chosen by matching classifier capabilities to the requirements of this research project. The inputs and outputs are then chosen. Algorithm selection is performed by selecting the best performing classifier after employing 5-fold cross validation across all classifiers. Furthermore, the sample set is a compilation of target versus target competition (i.e., random versus random, random versus myopic

scorer, etc.) where data from each turn are sample rows. The chosen classification tool is applied to the game engine to provide target prediction.

The classifiers considered for this research are naïve Bayes, logistic regression, decision trees, support vector machines (or SVM's) and neural networks. To obtain the best algorithm for this research, the following research requirements are established:

1. Classifier must distinguish between 3 target classes
2. Classifier must be able to handle large data sets (>10,000)
3. Classifier must recognize interactions between features (i.e. if a correlation between two features distinguishes one class from another) to increase the potential of the feature set

From these requirements, the matrix in Table 7 is devised to find the classifiers that best fit this research platform. Information from [1, 7, 29] describe the reasoning for the decisions in Table 7.

Table 7: Opponent Modeling Classifier Requirements Matrix.

	Output Choice - Three Targets	**Large Datasets >10,000 samples)**	**Interactions Between Features**
Naïve Bayes	YES	NO	NO
Logistic Regression	NO	YES	YES
Decision Trees	YES	NO (easily over-fits)	YES
Random Forests	**YES**	**YES**	**YES**
SVM's	NO (without kernel)	YES	YES
Neural Network	NO	YES	YES
LDA	NO	YES	YES
MDA	**YES**	**YES**	**YES**

Based on the requirements above, Table 7 shows that random forests and multiple discriminant analysis (MDA) are the best possible classifiers. The inputs and outputs of the opponent modeling classifier are provided in Table 8.

Table 8: Classifier Inputs.

Classifier Features	
1	Points Scored/Turn (double: $x \geq 0.00$)*
2	Defensive Score (integer: $0 \leq x \leq 3$)*
3	Total Player Pulls/Turn (double: $x \geq 0.00$)*
4	Total Scoring Plays/Turn (double: $x \geq 0.00$)*
5	Board Count (integer: $x \geq 0$)*
Classifier Output	
1	Target Number (integer: $0 \leq x \leq 2$)* 0 – Random Target 1 – Myopic Scoring Target 2 – Myopic Defense Target
*Variable *x* is the range of values that the classifier will accept.	

Inputs 1, 3 and 4 are normalized by the "Turn" number of the game. Points Scored is the total point value earned by the target. The Defensive Score is the same score defined by the defensive target. Total Player Pulls is the total number of times the target has caused the player to pull from the boneyard up to that point in the game. Total Scoring Plays is the total count of scoring plays the target has made up to that point. The Board Count input is the sum of all outside domino pips. (For example, Figure 17 illustrates a game instance with a Board Count of 16. This value is calculated by adding all outside values (clockwise for this example) 1+6+5+4 =16.) Lastly, data is labeled in reference to the target class, where class numbers range in integers between 0 and 2.

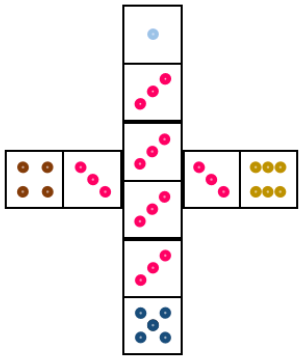

Figure 17: Domino board representation where the board count is 16.

5-fold Cross validation is employed to select the better machine learning algorithm where fitness is decided by the accuracy of each classifier's confusion matrix. Equation 9 illustrates the accuracy calculation per the confusion matrix in Table 5.

$$Accuracy(AC) = \frac{A + E + I}{S_{total}} \tag{9}$$

where S_{total} is the size of the sample set introduced to the classifier and A, E and I are the samples that the classifier accurately classified.

Each classifier is trained by playing games between each target. This training technique is employed to make sure data are provided that illustrate as many different play situations as possible for each target. Table 9 illustrates the order of training per game where each play is a sample of data. The data set size increases as this training order is repeated.

Table 9: Classifier training order per game.

Game	Player	Target
1	Random	Random
2	Myopic Defense	Random
3	Myopic Scoring	Random
4	Random	Myopic Defense
5	Myopic Defense	Myopic Defense
6	Myopic Scoring	Myopic Defense
7	Random	Myopic Scoring
8	Myopic Defense	Myopic Scoring
9	Myopic Scoring	Myopic Scoring

From this data set, 80% of the data is applied to training and testing while the other 20% is employed for validation. Once testing and validation is complete, the chosen classifier is equipped to pair with the M* algorithm and is ready for testing. The next section describes the experimentation process that shows the value M* provides to double of dominoes for a particular target.

Evaluation

This section facilitates the experimentation processes involved to show if M* with an opponent modeler adds value to a player agent in the game of two player dominoes. Testing implements all possible starting conditions in a set of double-3 dominoes with a boneyard of 4 with players receiving 3 dominoes apiece. This research provides evidence to whether the M* search decision process adds value to a player agent competing against three types of domino opponents.

To evaluate this hypothesis, results from all competitions are tabulated and compared with two types of significance tests. The first type of test called the difference of means significance test that evaluates the significance of the expected and actual value

gains. The second test is the binary test that evaluates the number of M* wins. The null hypothesis H_0, of this research states that there is no significant improvement over Expectimax when employing M* search. The alternate hypothesis H_1, states that M* provides an improvement over Expectimax to the player agent's game playing ability in both score differentials and total wins/ties.

The difference of means is a one-tailed test that evaluates the mean scores of all score differentials between the Expectimax decision engine and M*. The null hypothesis for this test is h_0: $\mu_{2-}\mu_1 \leq 0$, stating that there is either no difference in the means or (for this research), there is not enough information to show that M* shows a significant expected gain in value over Expectimax. The alternative hypothesis, h_0: $\mu_{2-}\mu_1 > 0$ states that M* shows significant gain over EM.

Difference of means.

For the difference of means the critical region is found on the Z-table. The Z score is calculated by finding $Z_{critical} = z_{table} \times \sigma_{\mu_2-\mu_1}$, where $\sigma_{\mu_2-\mu_1}$ is the standard deviation of the distribution of the difference of means and $Z_{critical}$ is the critical value.

$$\sigma_{\mu_2-\mu_1} = \sqrt{\frac{{\sigma_2}^2}{n} + \frac{{\sigma_1}^2}{n}} \quad \textbf{(10)}$$

where n is the population size, and ${\sigma_1}^2$and ${\sigma_2}^2$ are the population variances. If $\mu_{2-}\mu_1$ falls within the critical region (which is every value greater than the critical value), the null hypothesis is rejected and employ the alternative hypothesis.

Binary-test.

The binary test is a two tailed test that is applied to data with two outcomes [8]. This test is applied to the win/tie data to statistically show that the win/tie percentages are not the same. This test is conducted by comparing a Z score with critical values from a Z-table. Equation 11 illustrates how to find the Z-score [8].

$$z = \frac{\frac{X}{n} - p}{\sqrt{\frac{pq}{n}}} \tag{11}$$

where p and q represent the prior probabilities that Expectimax will win according to its data set along with its complement respectively. X is M*'s prior probability compared with EM's prior probability of wins plus ties and n represents the size of the sample space.

Illustrating the data.

Data is compared using two types of tables. The first type illustrates score differential data. Table 9 illustrates both decision engines against all three targets as well as the difference between each engine. This difference is known as the expected value gain for score differentials. The actual value gain also applies this table to show the difference in values between M* and Expectimax.

Table 10: Sample score differential results table for the value of the M* decision engine after all levels and factors of experimentation.

Targets	*Random*		*Myopic Scoring*		*Myopic Defense*	
Decision Engine	**M* (Source)**	**Expectimax**	**M* (Source)**	**Expectimax**	**M* (Source)**	**Expectimax**
Fully Observable	*a*	*b*	*c*	*d*	*e*	*F*
$EVG_{score\ \Delta}$	***a-b***		***c-d***		***e-f***	
%Accuracy	***I%***		***J%***		***K%***	

The lower case letters in Table 9 represent the average score differentials between the player and the target from the prospective target dataset. The upper case letters represent the prediction accuracy of the opponent classifier. The second type of table is the win/tie data table. Table 11 is a representation of the wins table that is applied to tabulate the environment and actual values for wins data.

Table 11: Sample win/tie data table for the value of the M* decision engine after all levels and factors of experimentation.

Targets	***Random***		***Myopic Scoring***		***Myopic Defense***	
Decision Engine	**M* (Source)**	**Expectimax**	**M* (Source)**	**Expectimax**	**M* (Source)**	**Expectimax**
Fully Observable	*a*	*b*	*c*	*d*	*e*	*F*
$EVG_{wins/ties}$	***a-b***		***c-d***		***e-f***	
%Accuracy	***I%***		***J%***		***K%***	

The lower case letters represent the total values for the wins and ties against each target with each decision engine. The upper case letters represent the prediction accuracy of the opponent classifier. Each win/tie has a value of 1 and each loss has a value of 0. All four Quadrants from Table 6 are explained applying the and Table 11 structures.

The number of tests to be run correspond to the number of levels and factors of experimentation. Table 12 illustrates the levels and factors for comparing both decision engines.

Table 12: List of levels and factors for experimentation.

<table>
<tr><th>Information Source (I)</th><th>Observability (O)</th><th>Target (T)</th><th>Starting State (S)</th><th>Decision Engine (E)</th><th>Repeat (R)</th></tr>
<tr><td rowspan="2">Oracle - Truth</td><td rowspan="2">Full</td><td>Random</td><td rowspan="3">1-4200</td><td rowspan="2">M* Search</td><td rowspan="3">1-12</td></tr>
<tr><td rowspan="2">Myopic Scoring</td></tr>
<tr><td rowspan="2">Opponent Modeler</td><td rowspan="2">Partial</td><td rowspan="2">Expectimax Search</td></tr>
<tr><td>Myopic Defense</td></tr>
</table>

The four quadrants of data are represented in the two left most columns.

Equation 2 is applied to calculate the number of possible starting states below.

$$Possible\ Starting\ States = \begin{bmatrix} Set\ size \\ boneyard\ size \end{bmatrix} \times \begin{bmatrix} Set\ size - boneyard\ size \\ dominoes\ per\ player \end{bmatrix} \quad (12)$$

$$Possible\ Starting\ States = \begin{bmatrix} 10 \\ 4 \end{bmatrix} \times \begin{bmatrix} 6 \\ 3 \end{bmatrix} = 4{,}200 \quad (13)$$

There are three target types that are played against in all 4,200 starting states by both decision engines. Furthermore, each test is repeated 12 times to account for variations in boneyard pulls.

Equation 14 applies Table 12 and shows of the number of tests needed to complete the dataset.

$$I \times O \times T \times S \times E \times R = \#\ of\ Games \quad (14)$$

where all variables are identified in Table 12. This calculates to

$$2 \times 2 \times 3 \times 4200 \times 2 \times 12 = 1{,}209{,}600\ Games. \quad (15)$$

All experimentation follows the guidelines outlined in this chapter. The next chapter provides an in depth discussion on the results of this experimentation. Chapter 4 also describes which classifier is applied to the opponent model, then demonstrates how the M* decision engine performs against a target with respect to a Nash Equilibrium player agent. Results from all four quadrants are analyzed for statistical significance to show whether the agent gains or loses value over the Nash equilibrium player depending on the information source and the environment.

IV. Analysis and Results

Chapter Overview

This chapter explains the results gathered from evaluating the opponent modeling algorithm with the EM and M* decision engines. The first series of experiments evaluates which classifying algorithm is the best for running the opponent modeler. Next the expected value gain of the opponent modeler is evaluated. Finally, the actual results between the M* and Expectimax decision engines are shown. Results gathered are in the form of score differentials between the player agent and the target.

Classifier Performance Results

Evaluating each Classifier.

There are two classifying algorithms chosen as candidates to run the opponent modeler. The classifiers are evaluated at three different sample sizes to assess how well they perform. The first candidate is the MDA classifier. Next is the random forests classifier. Each classifier is tested and validated using of the 5-folds cross validation with 8k, 50k and 130k samples of data, where 80% of the data samples are applied to training and testing and 20% of the data is applied to validation for each dataset. Classification accuracy (AC) on the validation data from these testes are illustrated via the confusion matrices.

Table 13: Confusion matrix validation results on 1,649 samples from the MDA Classifier with all three targets and 8,242 samples.

		Predicted Target		
		Random	**Myopic Scoring**	**Myopic Defense**
Actual Target	**Random**	**226**	166	151
	Myopic Scoring	147	**279**	133
	Myopic Defense	145	164	**238**

Applying the results from this table to Equation 9 for Table 13, yields $AC = \frac{226+279+238}{1,649} = 0.451$. The time to complete training, testing and validation is 50 ms. These data show that this classifier has the ability to classify three data types with over its prior percentage in a small amount of time. Table 14 illustrates the performance with over 50k samples.

Table 14: Confusion matrix validation results on 10,059 samples from the MDA Classifier with all three targets and 50,291 samples.

		Predicted Target		
		Random	**Myopic Scoring**	**Myopic Defense**
Actual Target	**Random**	**1135**	1132	1083
	Myopic Scoring	769	**1473**	1107
	Myopic Defense	700	1069	**1591**

Applying the results from this table to Equation 9 for Table 14, yields $= \frac{1135+1473+1591}{10,059} = 0.417$. The time to complete training, testing and validation is 330 ms.

These data show that it takes longer to classify the three data types; however, there is no value gained from more samples. This is an indication of over-training the classifier with too much data. Table 15 adds training data to confirm this assumption.

Table 15: Confusion matrix validation results on 25,945 samples from the MDA Classifier with all three targets and 129,724 samples.

		Predicted Target		
		Random	**Myopic Scoring**	**Myopic Defense**
Actual Target	**Random**	**2771**	2777	3038
	Myopic Scoring	2149	**3643**	2895
	Myopic Defense	1921	2564	**4187**

Applying the results from this table to Equation 9 for Table 15, yields $AC = \frac{2771+3643+4187}{29{,}945} = 0.409$. The time to complete training, testing and validation is 800 ms.

These data show that training the classifier with more than 8k samples (with this feature set) over trains the MDA classifier. Therefore, 8k samples is the best sample amount (when compared to the other two data sets) for this set of features.

The next candidate is the random forests classifier. This classifier undergoes the same amount of testing and validation to show its performance. The following confusion matrix analysis illustrates this classifier's performance.

Table 16: Confusion matrix validation results on 1,319 samples from the Random Forests Classifier with all three targets and 8,242 samples.

		Predicted Target		
		Random	**Myopic Scoring**	**Myopic Defense**
Actual Target	**Random**	**199**	132	113
	Myopic Scoring	82	**259**	109
	Myopic Defense	83	110	**232**

Applying the results from this table to Equation 9 for Table 16, yields $AC = \frac{199+259+232}{1,319} = 0.5231$. These data show that the random forest classifier has a better classifying accuracy than the MDA classifier; however, this classifier spends more time training and validating. The time to complete training, testing and validation is 35.32 s. Table 17 shows the performance of random forests with over 50k samples.

Table 17: Confusion matrix validation results on 8,047 samples from the Random Forests Classifier with all three targets and 50,291 samples.

		Predicted Target		
		Random	**Myopic Scoring**	**Myopic Defense**
Actual Target	**Random**	**908**	906	815
	Myopic Scoring	350	**1627**	742
	Myopic Defense	354	947	**1398**

Applying the results from this table to Equation 9 for Table 17, yields $AC = \frac{908+1627+1398}{8,047} = 0.4898$. This percentage is still larger than each MDA accuracy;

however, this percentage, again is lower than the 8k sample trained MDA classifier. Furthermore, the time to complete training, testing and validation is 413.31 s, which is much longer than MDA and random forests with 8k samples. Table 18 illustrates the random forests classifier performance with ~130k samples.

Table 18: Confusion matrix validation results on 20,756 samples from the MDA Classifier with all three targets and 129,724 samples.

		Predicted Target		
		Random	**Myopic Scoring**	**Myopic Defense**
Actual Target	**Random**	**2731**	2252	1865
	Myopic Scoring	1328	**4035**	1608
	Myopic Defense	1297	2156	**3484**

Applying the results from this table to Equation 9 for Table 18, yields $AC = \frac{2731+4035+3484}{20,756} = 0.4938$. These data show that there is a slight improvement over the 50k sample training set; however, this improvement is not higher than running 8k samples. The time to complete this process is also much higher than >413.31 s, which is the time to run the 50k data set.

Classifier Selection.

Both classifiers have a higher accuracy with a low sample count. Training and validating with more samples takes much longer while running both classifiers; however, the random forests classifier takes much longer than the MDA classifier to run its training/validation process. For 8k data samples, MDA takes 0.50 ms to complete the

validation process while it takes random forests over 35 s. This shows that even though random forests produce a higher classification accuracy with less data, it takes 70 times longer to obtain this result. Furthermore, the random forests classifier only gains a 10% advantage over MDA and MDA's classification accuracy for three classes is still greater than the prior probability of $\frac{1}{3}$. 70 times longer training, testing and validating time is not worth the 10% gain in percentage; therefore, based on these findings and the amount of simulations from the levels and factors of testing, MDA is chosen to complete the opponent model data collection.

Opponent Model Value Assessment

The opponent model value is assessed in two different conditions. The first condition is with perfect information, (as described in quadrants 1 and 2 of Table 6). Perfect information provides data on the upper bound value that the opponent modeler provides. These two quadrants show the possibility of gaining value against the Nash Equilibrium player while competing against three static targets. Next the value of the model is evaluated in an imperfect information environment (for quadrants 3 and 4). This environment demonstrates the actual upper bound value of the game. Each quadrant represents 50,400 games played with EM and M* against each target.

All values in each table are tested for statistical significance in accordance with calculations described in Chapter 3. The two tailed binary test shows significance for the win/tie total data. Its null hypothesis states that there is no difference in win/tie percentage between M* and EM. The alternative hypothesis states that there is a lower or higher difference in win/tie percentage. The one-tailed difference of means test is applied

to the score differential data. The null hypothesis of this test states that the difference of means between M* and EM is less than or equal to zero, while the alternative states that the difference of means is greater than zero. All significant values are computed with a 95% confidence interval. Cells with statistically insignificant values are shaded gray.

Perfect Information.

The data in Table 19 and Table 20 illustrate the upper bound expected value gains of the opponent model. The first three columns represent the expected value gains of the opponent modeler against all three targets. The bottom row shows the classification accuracy of the model M* employs. Since M* receives target predictions from the oracle in quadrant one, the prediction accuracy is 1.00 for all three targets.

Table 19: Mean per-game score differentials between the oracle and Expectimax for each target for 50,400 games in a fully observable environment with a target oracle.

Targets	**Random**		**Myopic Scoring**		**Myopic Defense**	
Decision Engine	**M* (Oracle)**	**Expectimax**	**M* (Oracle)**	**Expectimax**	**M* (Oracle)**	**Expectimax**
Fully Observable	1.869	1.982	1.388	1.269	1.633	1.474
$EVG_{score\ \Delta}$	**-0.113**		**0.120**		**0.159**	
Prediction Accuracy	**1.00**		**1.00**		**1.00**	

Table 19 illustrates the upper bound expected value gain score value for all 50,400 games. Since there is perfect state information and the search reaches the leaf nodes of the tree, Expectimax performs at its best. Furthermore, M* operates at its best with a known state and target strategy. This table represents the environment in which there exists perfect information in state and target strategy; therefore both engines are

functioning at their highest aptitude. With both engines playing at their best, this quadrant displays (given a specific target) if it is possible for M* to outplay the EM engine.

All players except for the random target contribute a gain for the M* engine. Characteristics of the random player involve making unpredictable moves. Furthermore, the M* algorithm's performance depends on making move predictions. These predictions prune parts of the tree in which it assumes that the target will not play. Incorrect information causes M* to make the wrong decision and traverse the wrong nodes. The EM engine makes decisions based on playing a Nash Equilibrium opponent, which means that it examines all plays the opponent can take and expects its opponent to make an optimal move. In short EM is expected to make a better decision than M* assuming M* obtains an incorrect target prediction. Table 20 displays the highest potential gain of win/tie totals between the EM and M* decision engines. The scoring target is positive, which shows that Expectimax loses 466 more games than the M* agent in this quadrant. Expectimax also loses 352 more games playing against the defense target. This means that M* has the potential gain of these values in the other three quadrants. The random target value is insignificant.

Table 20: Total win and tie mean differences between oracle and Expectimax for each target in a fully observable environment with a target oracle.

Targets	Random		Myopic Scoring		Myopic Defense	
Decision Engine	M* (Oracle)	Expectimax	M* (Oracle)	Expectimax	M* (Oracle)	Expectimax
Fully Observable	34,107	34,031	32,209	31,743	33,257	32,905
$EVG_{wins/ties}$	76		466		352	
Prediction Accuracy	1.00		1.00		1.00	

Table 21 and Table 22 illustrate the actual value for double-3 dominoes with perfect state information with the opponent model containing an MDA classifier trained and validated with 8k samples. The one-versus-rest (offline) prediction accuracy for the MDA is 45%. Table 21 shows the actual value in score differentials and Table 22 illustrates the actual value in wins over the Expectimax. Furthermore, these tables show the actual on-line prediction accuracy against all three targets.

Table 21: Mean per-game score differentials between oracle and Expectimax for each target in a fully observable environment with an opponent model.

Targets	Random		Myopic Scoring		Myopic Defense	
Decision Engine	M* (OM)	Expectimax	M* (OM)	Expectimax	M* (OM)	Expectimax
Fully Observable	1.984	1.982	1.143	1.269	1.517	1.474
$EVG_{score\ \Delta}$	0.002		-0.126		0.043	
Prediction Accuracy	0.316		0.465		0.336	

All expected value gains in Table 21 are statistically insignificant. The actual prediction accuracy for the defensive target is 0.336 which is slightly higher than the prior percentage (of a random guess) but much lower than 1.00. The prediction accuracy for the myopic scoring agent is much higher than the prior probability of 1/3; however it

still does not help M* enough to produce a gain against EM. Furthermore, the random target prediction accuracy is lower than the prior probability. This shows that the classifier selects (or classifies) the myopic scoring target over the other two targets. Table 22, illustrates the actual values with total wins and ties.

Table 22: Total win and tie mean differences between oracle and Expectimax for each target in a fully observable environment with an opponent model.

Targets	**Random**		**Myopic Scoring**		**Myopic Defense**	
Decision Engine	**M* (OM)**	**Expectimax**	**M* (OM)**	**Expectimax**	**M* (OM)**	**Expectimax**
Fully Observable	34,212	34,031	31,345	31,743	32,988	32,905
$EVG_{wins/ties}$	**181**		**-398**		**83**	
Prediction Accuracy	**0.316**		**0.465**		**0.336**	

Table 22 illustrates the gain in win/tie totals for M* with perfect state and imperfect target identity (action) information. The imperfect action information affects the myopic scoring win/tie differentials with a loss of 864 more games than quadrant 1. The Myopic defense and random targets produce insignificant gains with the information provided.

Holding the state information constant allows the study of how much affect prediction accuracy has on performance. The M* decision engine makes optimal decisions based on the target information it receives. These data show that with prediction dropping from 1.00 to less than 0.50 the M* decision engine performs with lower gains than with perfect target identity information. Therefore, this depicts that this drop in action information contributes to the loss in score differential and win/loss gains.

Imperfect Information.

Games played with imperfect state information are shown in Table 23 through Table 26. These games are played with an environment identical to regular play (assuming the target type is known). Quadrant three shows games in which state information is imperfect while the target type is known. Quadrant four illustrates the results for imperfect target information and state.

Table 23: Mean per-game score differentials between M*(oracle) and Expectimax for each target under imperfect information in a partially observable environment with a target oracle.

Targets	**Random**		**Myopic Scoring**		**Myopic Defense**	
Decision Engine	**M* (Oracle)**	**Expectimax**	**M* (Oracle)**	**Expectimax**	**M* (Oracle)**	**Expectimax**
Fully Observable	1.556	1.526	0.763	0.726	1.027	0.914
$EVG_{score\ \Delta}$	**0.030**		**0.037**		**0.113**	
Prediction Accuracy	**1.00**		**1.00**		**1.00**	

Table 23 illustrates games with imperfect state but the target type is known. Holding the target oracle constant allows for the comparison of how partial observability affects the score difference EVG. Since the oracle perfectly predicts the target for M* in this quadrant, M* is able to gain an advantage over the Nash Equilibrium agent. The myopic defense provides the largest gain for this table with 0.113. This value is down 0.46 from the potential gain of 0.159 shown in results from Table 23. This shows that the imperfect state has an effect on the amount of gain M* provides over the Nash Equilibrium player. The other two target values provide insignificant gains. Table 24

illustrates the expected value gain for the number of wins/ties with imperfect state information and a target oracle.

Table 24: Win and tie totals between M*(oracle) and Expectimax for each target in a partially observable environment with a target oracle.

Targets	**Random**		**Myopic Scoring**		**Myopic Defense**	
Decision Engine	**M* (Oracle)**	**Expectimax**	**M* (Oracle)**	**Expectimax**	**M* (Oracle)**	**Expectimax**
Fully Observable	32,962	32,887	29,759	29,708	30,988	30,688
$EVG_{wins/ties}$	**75**		**51**		**310**	
Prediction Accuracy	**1.00**		**1.00**		**1.00**	

These data show that there is potential to win and/or tie 310 more games than the Nash Equilibrium agent in a partially observable environment for the myopic defense target. The other target values are insignificant in this quadrant.

Actual Value Evaluation

Table 25 and Table 26 illustrates the opponent model's actual value table for the data taken in all 50,400 starting states using both imperfect state information and an imperfect opponent classifier used with M*. These data show the real world environment in which dominoes games take place. Furthermore, these tables illustrate how the M* algorithm fares against a Nash equilibrium player in the environment when playing against three different target types.

Table 25: Mean per-game score differentials between M*(OM) and Expectimax for each target in a partially observable environment with an opponent model.

Targets	Random		Myopic Scoring		Myopic Defense	
Decision Engine	**M* (OM)**	**Expectimax**	**M* (OM)**	**Expectimax**	**M* (OM)**	**Expectimax**
Fully Observable	1.507	1.526	0.761	0.726	.988	0.914
$AVG_{score\ \Delta}$	**-0.019**		**0.035**		**0.074**	
Predict Acc.	**0.272**		**0.627**		**0.170**	

Table 25 shows that the best performance gain for the M* decision engine when compared with a Nash equilibrium player, occurs against the myopic defense target (with a 0.074 actual value). This shows that in a real world environment the M* decision engine has an actual gain of .074 points than the Nash equilibrium. The other two targets show insignificant gains. Table 26 illustrates how many games M* wins or ties over Expectimax.

Table 26: Win and tie total difference between M*(OM) and Expectimax for each target in a partially observable environment with an opponent model.

Targets	Random		Myopic Scoring		Myopic Defense	
Decision Engine	**M* (OM)**	**Expectimax**	**M* (OM)**	**Expectimax**	**M* (OM)**	**Expectimax**
Fully Observable	32,205	32,887	29,717	29,708	30,846	30,688
$EVG_{wins/ties}$	**-682**		**9**		**158**	
Predict Acc.	**0.272**		**0.627**		**0.170**	

These data show that there is no significant gain in games over the Nash Equilibrium agent when playing against the three targets. The prediction accuracy of this data set is the lowest with the myopic defense target. This shows that (while holding

imperfect information as a constant from quadrant 3 to 4), lowering the prediction accuracy to under 1.00 affects the performance of the M* engine.

The highest prediction accuracy is attributed to the myopic scoring agent. This value of 0.627 shows that the MDA classifier predicts the scoring target more often than the other two targets; however this is not enough to gain value over the Nash equilibrium agent.

Against the random agent, M* produces a significant loss of 682 games in comparison with the Nash Equilibrium agent. The random agent has an unpredictable strategy. Furthermore, M* relies on a known target type and strategy to perform well. The lack of a deterministic strategy to make decisions causes M* to play sub-optimal games. Expectimax is able to make optimal moves without a strategy based on its expectation of the target to play optimally. This explains why the Nash Equilibrium agent is expected to win more games against the random target.

Summary of Results

The total expect value gain in the first quadrant is 0.159. This value drops in the second quadrant due to the lack of perfect target knowledge. The same observation occurs with the myopic scoring expected value gain. Even with an opponent modeler with a 46% prediction accuracy, this gain is insignificant.

The partial observable quadrants show that there is potential to score a gain of 0.113 over the Nash Equilibrium player with the myopic defense target. The other two values are insignificant in the third quadrant. The end results show that in the last quadrant, there is still a statistically significant value gained over the Nash equilibrium

agent of 0.74 by the M* agent with the opponent modeler while playing against the defense agent.

The Defense target demonstrates a 352 game advantage over the Nash equilibrium player in the perfect information environment playing with a target oracle; however, this gain is lost when applying the opponent modeler at a 33.6% prediction accuracy. Similar to these results, the third quadrant of data illustrates a potential gain of 310 games over the NE player. The prediction accuracy is much lower in the fourth quadrant at 17%. This lower prediction shows to affect the M*'s ability to make good decisions and win more games.

In summary, the data show that the M* decision engine with an opponent modeler is able to score higher than the Nash Equilibrium agent. It also has the potential to win more games. The opponent modeler's accuracy has an effect on how many games the M* player wins as well as how many points it scores. Chapter 5 provides conclusions of this thesis research and future work that can add advancement to this research topic.

V. Conclusions and Future Work

Based on the results of characterizing relative performance improvement over the Nash Equilibrium solution for three types of dominoes-playing opponents, applying M* search with an opponent model shows promise as long as the model has a high quality prediction of board state information (to include what is in the other player's hand and what dominos are in the boneyard). This chapter summarizes the findings in the data and explains where research can further explore this topic.

This research shows results for the M* decision engine with a 100% prediction accuracy and prediction accuracy less than 100%. Quadrants 1 and 3 contain potential values to be achieved by the opponent modeler. The lower percentages produce lower actual results in quadrants 2 and 4. This means that potential exists for M* win or tie more games than a Nash equilibrium player against specific targets.

The myopic scoring and random targets provide negative or insignificant gains for the opponent modeler. There is potential (from quadrant 1) for the M* search engine to have significant gain over Expectimax; however, data show that prediction accuracy of the opponent modeler is not good enough to predict the correct target and make an optimal decision. The random target's contribution to expected and actual gains are insignificant to negative, showing that M* has no potential to gain an advantage over the Nash Equilibrium player if the target plays a random strategy.

In conclusion, applying an opponent modeler has potential to add value to an agent in double three dominoes for specific opponent types. If the opponent plays a

random strategy, there is no potential. The accuracy of the opponent modeler has a big effect on the way the algorithm makes decisions, even if the opponent classification accuracy is over 50% (as shown by the scoring target in the fourth quadrant). The next section explains future work that can help to further this research path.

Future Work

Future work for this research topic involve areas where improvements can be made with the M* search parameters such as the opponent model and the state provided. These parameters can be tuned in order to close the gap between the oracle results and the opponent classifier results. The first area for improvement involves adding an opponent modeler for the state of the game. The next chapter discusses playing against non-myopic targets. Then it presents a brief discussion on adding a probability distribution over all M* targets decisions and choosing the decision with the highest value. Lastly discussed is applying an online learning module to the classifier in order to learn as the agent plays.

Opponent Modeler for Board State.

The data show that the value of the opponent modeler is higher in a fully observable environment. Revealing information about the state of the game to the decision agent allows it to make better choices. Therefore adding an opponent modeler for state information will provide better results for the opponent modeler if applied to the M* search.

Non-Myopic Targets.

Another area to explore is non-myopic targets. These targets are in line with Carmel and Markovitch's research [6]. Their research explores fully observable games where players assume the opponent's termination condition as well as the evaluation function of the opponent. Applying this concept to a partially observable game means that both players will have to model each other and have some intelligence of the board state.

Applying a Probability Distribution Model to the OM.

The opponent modeling paradigm applied in this thesis performs a three-way classification and picks the most likely opponent from the set of three. In this thesis, M* assumes there is only one possible opponent type with certainty. By altering the output of the opponent-modeling classifier to yield a probability distribution over all possible targets, M* can form a strategy by selecting from a probabilistically-weighted best response to several target strategies instead of just one, whenever their predicted action would differ.

Summary

In conclusion, this research can be applied to applications that involve two or more players that have a desire to gain value in specific environments. For fully observable games, modeling opponent's actions and applying them to an M* style search method provides a player the ability to make better choices as long as the target's strategy is predictable.

Appendix A. Rules of Dominoes

According to Armanino [2], players randomly choose an equal set of two sided tiles of a 28 tile set. Each side of the tiles contains values ranging from blank (or zero) to 6, making each tile in the set unique. Figure 18 shows a full set of 28 dominoes.

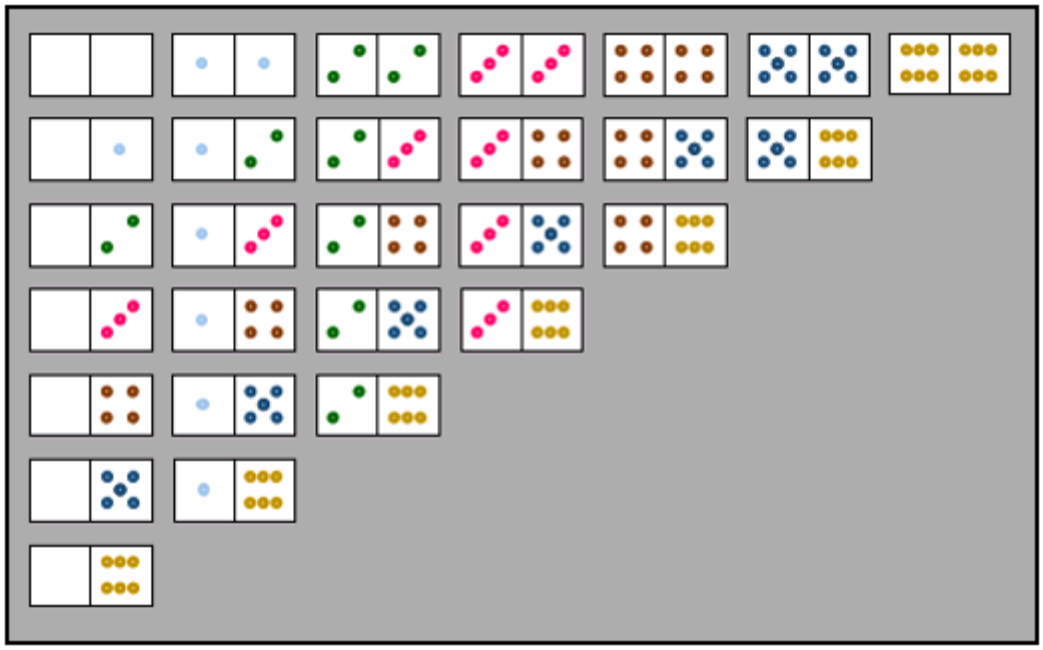

Figure 18: Full set of double-six dominoes (28 pieces).

The object of the game is to match the sides of the dominoes with the dominoes on the board and be the first to achieve a predefined ending score. Scoring is achieved by adding the horizontal and vertical ends of the dominoes on the table. If the total is divisible by 5, that is a scoring play. Figure 19 illustrates how moves are made between

the player and opponent in a simple game with three dominoes per participant

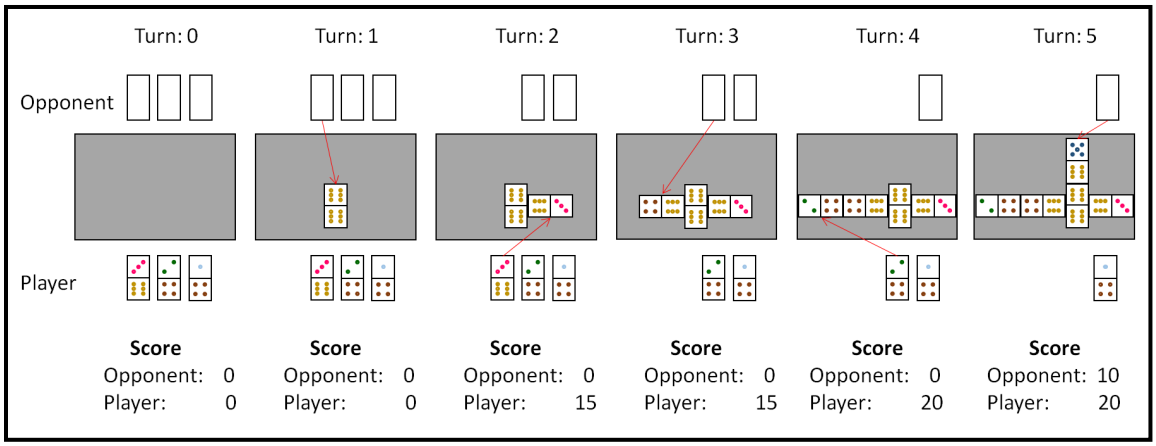

Figure 19: Example 5 turn game

As shown, the player scores twice in this game. The first score of 15 is made by placing the (6|3) domino. This domino produces a value of 3 on the outside right end, which adds to the value of 12 of the (6|6) domino on the left outside end. The second score of 5 is made by the player placing (2|4) with a value of 2 on the left outside end adding to (6|3) with a value of 3 on the right side end. Consequently, the opponent then takes advantage of this opportunity to score 10 points by placing the (6|5) on the top edge with the value of 5 exposed adding with the 2 and 3. By placing the (4|1) instead of the (4|2), the player could have prevented the opponent from scoring. In short, this instance demonstrates how easy scoring can occur as well as how a player can defend by playing the right dominoes at the right time.

Additional rules state that the player whom achieves a "domino" [12], (plays all dominoes in that player's hand first,) obtains an additional score: the remaining pip (domino dot) total in his/her opponent's hands. Defensive play adds to the chance of

"dominoing" by preventing the opponent from playing all of their dominoes. Defense is played by placing dominoes that cause the other player to pull from the boneyard.

Vita

Michael M. Myers was born on September 10, 1980 in Madison, Wisconsin. He graduated from the University of Wisconsin – Milwaukee in 2005 with a Bachelor of Science degree in Electrical Engineering. Michael also received his commission in May of 2005 through the Reserve Officer Training Corps. He entered active duty service in the Air Force in June of 2005 where he spent 4 years at the 453 Electronic Warfare Squadron at Lackland Air Force Base, TX. He then spent 3 years as a program manager and Section Chief at Starfire Optical Range at Kirtland Air Force Base, NM. He began pursuing a Master of Electrical Engineering with an emphasis on software engineering at the Air Force Institute of Technology in Augusts 2012.

References

[1] E. I. Altman, "Financial Ratios, Discriminant Analyisis and the Prediction of Corporate Bankruptcy," *The Journal of Finance,* vol. 23, 1968.

[2] D. Armanino, *Dominoes, Five-Up and Other Games*. Coachwhip Publications, 1959.

[3] D. Billings, D. R. Papp, J. Schaeffer and D. Szafron, "Opponent modeling in poker," in *Association for the Advancement of Artificial Intelligence,* Madison, Wisconsin, 1998, pp. 493-499.

[4] B. J. Borghetti, "The Environment Value of an Opponent Model," *IEEE Transations on Systems, Man, and Cybernetics-Part B: Cybernetics,* vol. 40, pp. 623, 2010.

[5] M. Campbell, A. J. Hoane Jr. and F. Hsu. Deep blue. *Artif. Intell., vol. 134, (1–2),* pp. 57-83. 2002. . DOI: http://dx.doi.org/10.1016/S0004-3702(01)00129-1.

[6] D. Carmel and S. Markovitch. Incorporating opponent models into adversary search. *Proceedings of the National Conference on Artificial Intelligence, vol. 1,* pp. 120-125. 1996.

[7] E. Chen. *Edwin Chen's Blog,* http://blog.echen.me/2011/04/27/choosing-a-machine-learning-classifier/.

[8] A. Clarke. *The Binomial Test,* http://www.elderlab.yorku.ca/~aaron.

[9] P. I. Cowling and Ward, Colin D., Powley, Edward J., "Ensemble Determinization in Monte Carlo Tree Search for the Imperfect Information Card Game Magic: The Gathering," *IEEE Transations on Computational Intelligence and AI in Games,* vol. December, 2012.

[10] A. Davidson, "Opponent Modeling in Poker: Learning and Acting in a Hostile and Uncertain Environment," 2002.

[11] A. Davidson, D. Billings, J. Schaeffer and D. Szafron, "Improved opponent modeling in poker," in *International Conference on Artificial Intelligence,* Las Vegas, NV, 2000, pp. 1467-1473.

[12] Domino-Games.com. *Domino Basics,* http://www.domino-games.com/domino-rules/domino-basics.html.

[13] H. H. L. M. Donkers, J. W. H. M. Uiterwijk and H. J. van den Herik. Probabilistic opponent-model search. *Inf. Sci., vol. 135, (3–4),* pp. 123-149. 2001. . DOI: 10.1016/S0020-0255(01)00133-5.

[14] J. Funge and I. Millington, "Game theory," in *Artificial Intelligence for Games*, 2nd ed.Burlington, MA: Morgan Kaufman, 2009, pp. 668-669-708.

[15] H. Hamilton. *Knowledge Discovery in Databases,* http://www2.cs.uregin a.ca/~dbd/cs831/notes/confusion_matrix/confusion_matrix.html.

[16] T. Hauk, M. Buro and J. Schaeffer, "Rediscovering *-Minimax Search," *Lecture Notes in Computer Science,* vol. 3846, pp. 35-36-50, 2006.

[17] C. Hock-Chuan. 2012, *Java Graphics Tutorial,* http://www.ntu.edu.sg/home/ ehchua /programming/java/JavaGame_TicTacToe_AI.html.

[18] Y. Huang, "The Development and Implementation of A Domino Game," 2005.

[19] R. Isaacs, "A Card Game with Bluffing," *The American Mathematical Monthly,* vol. 62, pp. 99-100-108, 1955.

[20] A. J. Lockett, C. L. Chen and R. Miikkulainen, "Evolving explicit opponent models in game playing," in *Proceedings of the 9th Annual Conference on Genetic and Eevolutionary Computation,* London, England, 2007, pp. 2106-2113.

[21] R. D. McKelvey, A. M. Mclennan and T. L. Turocy. *Gambit: Software Tools for Game Theory,* http://www.gambit-project.org, http://www.gambit-project.org.

[22] H. Peters, "Extensive form games," in *Game Theory, A Multi-Leveled Approach,* Springer Berlin Heidelberg, 2008, pp. 197-212.

[23] A. Polyakov. *Domino 1.2.0,* http://qt-apps.org/content/show.php/Domino?content=105765, http://qt-apps.org/content/show.php/Domino?content=105765.

[24] M. Richards and A. Eyal. Opponent modeling in scrabble. Presented at International Joint Conference on Artificial Intelligence. Hyderabad, India, 2007, pp. 1482-1487.

[25] D. Ross. *Game Theory, The Stanford Encyclopedia of Philosophy (Winter 2012 Edition),* http://plato.stanford.edu/archives/win2012/entries/game-theory.

[26] S. Russel and P. Norvig, "State-of-the-art game programs," in *Artificial Intelligence, A Modern Approach*, 3rd ed., M. Hirsch, Ed. Saddle River, NJ 07458: Prentice Hall, 2010, pp. 185.

[27] F. Schadd, S. Bakkes and P. Spronck, "Opponent modeling in real-time strategy games," in *GAMEON,* Bologna, Italy, 2007, pp. 61-70.

[28] M. A. Smith, "A Learning Program Which Plays Partnership Dominoes," *Communications of the ACM,* vol. 16, pp. 462-463-467, 1973.

[29] I. H. Witten, E. Frank and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann, 2011.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
27 MAR 2014	Master's Thesis	Aug 2012 – Mar 2014

4. TITLE AND SUBTITLE
Outperforming Game Theoretic Play with Opponent Modeling in Two Player Dominoes

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Myers, Michael M., Captain, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way, Building 640
WPAFB OH 45433

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT-ENG-14-M-57

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Intentionally Left Blank

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution statement A. Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES
This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

Dominoes is a partially observable extensive form game with probability. The rules are simple; however, complexity and uncertainty of this game make it difficult to apply standard game theoretic methods to solve. This thesis applies strategy prediction opponent modeling to work with game theoretic search algorithms in the game of two player dominoes. This research also applies methods to compute the upper bound potential that predicting a strategy can provide towards specific strategy types. Furthermore, the actual values are computed according to the accuracy of a trained classifier. Empirical results show that there is a potential value gain over a Nash equilibrium player in score for fully and partially observable environments for specific strategy types. The actual value gained is positive for a fully observable environment for score and total wins and ties. Actual value gained over the Nash equilibrium player from the opponent model only exist for score, while the opponent modeler demonstrates a higher potential to win and/or tie in comparison to a pure game theoretic agent.

15. SUBJECT TERMS
Dominoes, Opponent Modeling, M* search, Expectimax, Myopic Strategy

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Brett J. Borghetti, AFIT/ENG
U	U	U	UU	101	19b. TELEPHONE NUMBER *(Include area code)* (937) 255-6565, x 4612 brett.borghetti@afit.edu