Implementation of Minimax with Alpha-Beta Pruning as Computer Player in Congklak

Brian Sumali[#1] , Ivan Michael Siregar[#2], Rosalina[*3]

President University, Jl. Ki Hajar Dewantara, Cikarang Baru – Bekasi 17550

[1]brian.sumali@hotmail.com

[2]ivan.siregar@gmail.com

[3]rosalina@president.ac.id

Abstract — **Congklak is one of the most popular traditional board games in Indonesia. It is said to train mathematical aspect of the player. Interestingly, nowadays more and more people are becoming less interested. Nowadays, people are paying less attention to traditional things and instead focus more to digital and electronic devices, including Indonesian. Traditional things are beginning to be forgotten from the minds of Indonesian. To re-capture Indonesian's heart for traditional things, one of the ways is to digitize them. Hopefully by making a video game of Congklak more people, especially Indonesian, become more interested in Congklak as one of Indonesian traditional. Minimax Algorithm with its improvement, Alpha-Beta pruning is an option to make an intelligent computer player for Congklak. By applying Minimax Algorithm to the computer player, it becomes intelligent enough to play Congklak as a good opponent; making the game more fun.**

Keywords— **congklak, minimax algorithm, alpha-beta pruning, artificial intelligence**

I. INTRODUCTION

Games are entertainment facility which favored by adults and young alike. Traditional games are any kind of games which were inherited from older generations. Most of these kinds of games were passed to younger generation without any documentation to note its existence; any rules were passed by oral, and not textual. Modern games, in other hand, are any kind of games which were presented in a device (i.e. PC, smartphones, consoles).

A board game is a game that involves counters or pieces moved or placed on a pre-marked surface or "board. Games can be based on pure strategy, chance (e.g. rolling dice), or a mixture of the two. Early board games represented a battle between two armies, and most current board games are still based on defeating opposing players in terms of counters, winning position, or accrual of points (often expressed as in-game currency). Sometimes easy-to-remember board games are hard-to-master, even having professional ranking for it. Examples for this type of game are Chess, Shogi, and Congklak.

Congklak is one of the most popular traditional board games in Indonesia. It is said to train mathematical aspect of the player, similar to other two-player-perfect-information family games: Chess, Go, Shogi, Checkers, and others. Congklak is also quite famous internationally, having spawned several games which can be found on the internet. Interestingly, all the games the writer has played have some differences in their rules and also none of them has the exact same rules as what the writer knows.

No less interesting is the decline of interest from modern day's youngsters to Congklak. Congklak is an easy-to-play and easy-to-remember game. In addition, it is a traditional game; at least adults should know how to play it (and know how to introduce them to their children). With observation, it is common sense that modern day children rarely play Congklak or any other traditional games; they prefer to play Computers and gadgets. Hence, the writer decides to make a program for Congklak, with the rules as the writer knows.

AI for games from the family of two-player-perfect-information games is a hot topic for Computer Scientists. Many of Computer Scientists tried to solve games of this family; one of the famous products is the Deep Blue for chess.

One of the simplest ways to detect the best move of a state for a game from this family is to simulate all possible moves, from start to end. The problem of this solution is that the number of possibilities is very huge that even present-day supercomputer is unable to check in a reasonable time.

Congklak, luckily, is not the impossible one. Even though it may seem impossible for humans, it is definitely possible for the computers. It is all because Congklak's game-state is not an expanding one. Take a look at chess: the first turn, one player may move eight of his pawns plus two of his knights – totaling ten pieces. The second turn, it is possible for that player to have a possible move for more than ten pieces. But Congklak in the other hand, the maximum move is fixed: the number of the player's holes; what makes it better is that the possible move may decrease, but it will never increase to more than the number of the player's holes. In conclusion, it is perfectly fine to apply this solution to Congklak.

The objective of this research are:

1) To create game application to applies Minimax theorem with Alpha-Beta Pruning as the Artificial Intelligence (AI)

2) To exploit the AI thus the game can make a decision within a short space of time with several levels of difficulties.

There are many variations of rule for congklak game. The rules followed by this research are:

1) It is played on a block with two rows of seven circular holes and two large holes at both ends called "home", totaling sixteen holes.

2) The game begins with seven game pieces (shells, marbles, pebbles or seeds) in each hole except "homes" which remain empty. Each player controls the seven holes on his side of the board and owns the "home" to his right. The goal is to accumulate as many pieces in your own "home".

3) On a turn, a player removes all pieces from one of the seven holes on his side. He then distributes them counter-clockwise --- one in each hole to the right of this hole --- in a process called sowing. Sowing skips an opponent's "home" but not a player's own "home".

4) If the last piece sown falls into a player's own "home" then the player earns another turn, which can begin at any of the seven holes on his side. This is considered a priority move for most of Congklak player; if they can make it, they will do it.

5) If the last piece sown falls into an empty hole on his side then the player captures all the pieces in the hole directly across from this one, on the opponent's side and put them (plus the last piece sown) in his own "home". This ends the player's turn. This method is called as "shooting".

6) If the last piece falls into an occupied hole then all the pieces are removed from that hole, and are sown in the same way (clockwise from that hole) in another round. This player's (current) turn ends when the last piece falls into an empty hole on the opponent's side.

7) The other player chooses which hole he wishes to start from, removes the pieces and sows them - one in each hole, clockwise from that chosen hole. If a player has no pieces on his side of the board when it is his turn, then he must pass.

8) The game ends when no pieces are left in any hole on both sides of the board. The players now count the number of pieces in their own "home" and see who has won.

II. LITERATURE STUDY

Congklak game is popular mancala game in Malaysia, Brunei, Singapore and Indonesia. Many Indonesian believes that the game originated in Malacca Kingdom where it became very popular and spred to the South-East Asia region, this spread due to the many travelers who visited the kingdom because it was a trading city [13].

In Indonesia, Congklak has its roots deep in our culture. Almost every child knows Congklak (even with different name and/or rules). However, nowadays this traditional game is slowly being extinct and this is not only happened in Indonesia but as well as in Malaysia and therefore to make congklak game more interesting for the youth generation, the game build with visual graphics, framework, interactive media and animation [1]. However to make it more interesting this research will not only focus on developing interactive and intelligent congklak game application but it has several level of difficulties feature by implementing Minimax with Alpha-Beta Pruning. Minimax is an algorithm that uses state searching to find the best possible move and Minimax algorithm is known as the most effective algorithm for creating an artificial agent for board game [6]. This research also evaluates the time taken by the AI to perform and calculate the optimal moves for their respective difficulties.

Minimax Algorithm

Minimax is the classic depth-first search techniques for sequential two-player games. These two players are called MAX and MIN. The minimax algorithm is designed for finding the optimal move for MAX, who is the player to move at the root node. The search tree is created by recursively expanding all nodes from the root in a depth-first manner untul either the end of the game or the maximum search depth is reached[5].

Figure 1.1 is a simple example taken from [12], this example is very simple since the tree only has a depth of 2 and a branching factor of 3. If the tree has a maximum depth of m and there are b possible moves in each position, the complexity of the minimax algorithm is $O(b^m)$.

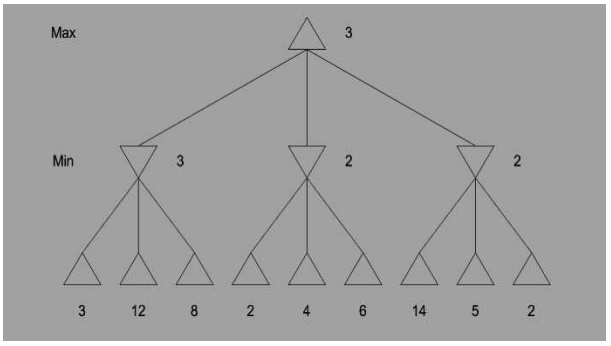

Figure 1.1 Minimax tree

Alpha-Beta Pruning Algorithm

The alpha-beta algorithm is equivalent to the minimax algorithm in that they both find the same best move from position p and both will assign the same value of expected advantage to it. Alpha-beta is faster than minimax because it

does not explore some brances of the tree that will not effect the backed-up value [10], it works by putting a minimum and maximum boundary of the value; alpha as the maximum lower bound of solutions and beta as minimum upper bound of the solution. Whenever any node returns a value to its parent, it may change the parent's alpha-beta boundary. Max-Node parent will change the Alpha boundary and Min-Node parent will change the Beta boundary. If the condition of alpha >= beta has been reached, the search for the children of that node will be stopped (pruned), since valid solution value will always be greater than or equal to alpha and less than or equal to beta (impossible to achieve if alpha > beta; and IS the value if alpha = beta) [2].

An example is also taken from [11], it is the same tree as shown in Figure 1.1, the prunning is implemented by introducing two bounds: the aplha as a lower bound and the beta as an upper bound. After the leftmost 3 leaves were investigated the max player at the root knows it can achieve at least a value of 3. Thus the minimum bound is alpha=3. When the first child of the second min node is investigated the min player can achieve at most 2. Therefore, the beta value is 2. In that case the two remaining leaves are puned, because beta < alpha [7].

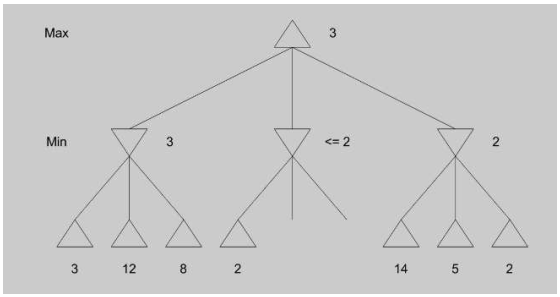

Figure 2.2 Minimax tree after aplha-beta pruning

III. SYSTEM ANALYSIS

This research is intended for two purposes: recreational purpose of a game and for creating an AI for Congklak game, but it is stressed on creating the AI. This research also focuses on making the AI to be fairly quick in making its decision, desirably below five seconds per move. The usecase for the program is denoted in Figure 3.1.

Figure 3.2 illustrates the swimlane diagram for playing with computer. After clicking the "Play with Computer" button, the application closes the main window and opens the game window.

A turn starts by locking all of the player's input buttons. Then, a branching starts. If the current turn is player's turn, then the system will unlock the player's input buttons and waits for the player input. After the input, the application will again lock all of the player's input buttons. If the current turn is the computer's turn, the application will generate the computer's move. The move, whether it is the player's or the computer's, will then be processed. The result of this will start another branch: If the move results in an end-game state, then end the game. If it is not, then it links to the very first action of a turn: Checking whose turn it is.

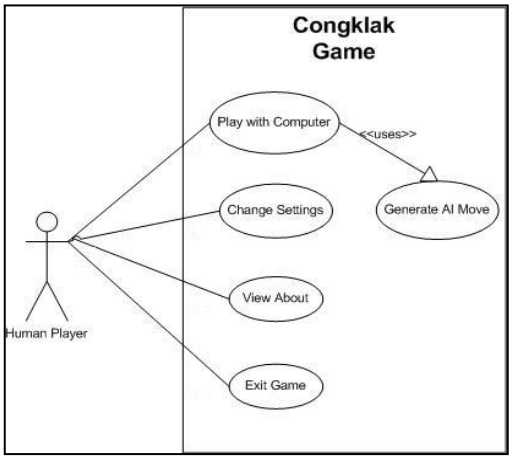

Figure 3.1 Use case diagram

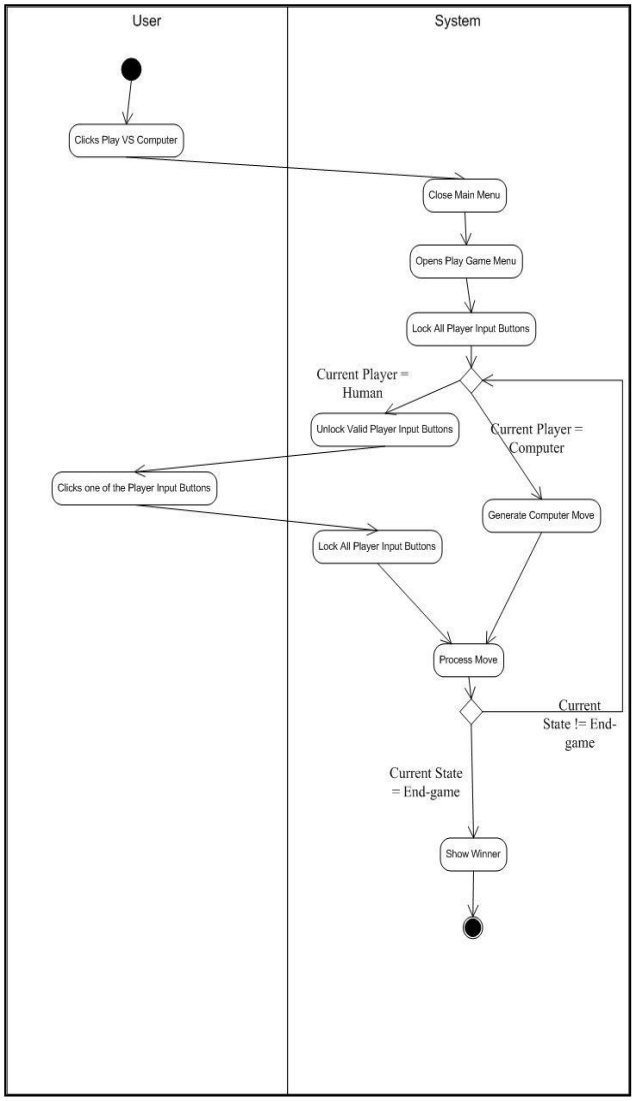

Figure 3.2 Swimlane Diagram for Play with Computer

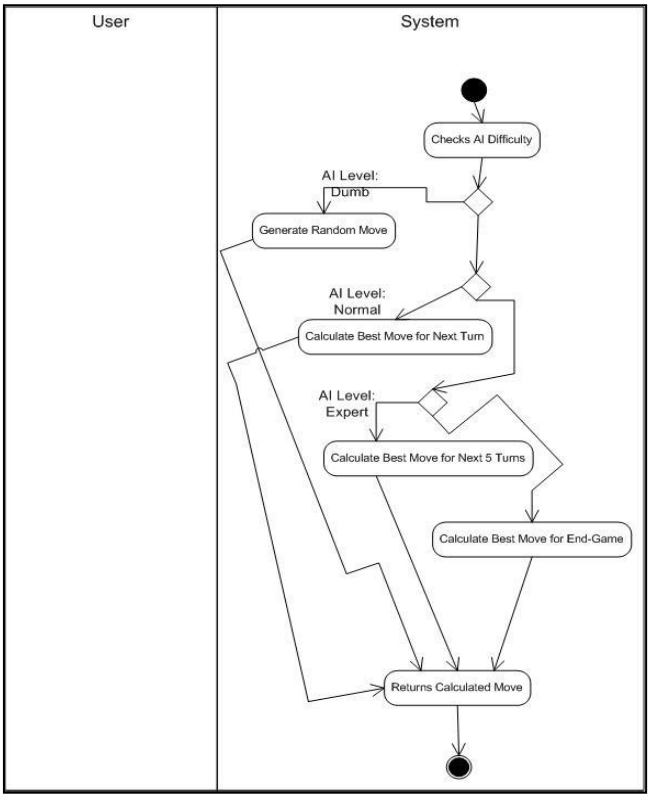

Figure 3.3 Swimlane Diagram for Generate AI Move

Figure 3.3 illustrates the swimlane diagram for generating AI move. The system generates AI move appropriates to the AI Level from the game settings.

A. Minimax

Implementation of Minimax Algorithm usually involves usage of three methods:

- GetMove (State S) : Move ➔ get the best move for current state
- GetMax (State S, Depth D) : Value ➔ get the best move value for evaluated state
- GetMin (State S, Depth D) : Value ➔ get the worst move value for evaluated state

GetMax and GetMin implementations have their current evaluation-depth as their parameter. This parameter will be compared to a global depth-limit variable to limit the maximum depth to be evaluated.

Figure 3.4, 3.5, and 3.6 shows the pseudocode for these three methods. GetMove, GetMax, and GetMin are all similar. They all compares and retrieves value from Minimax Search Tree's leaf nodes; their difference lies in: GetMove retrieves a move instead of an evaluation value, GetMax retrieves the best evaluation value, and GetMin retrieves the worst evaluation value.

Method GetMove(State *S*):

1. Let *S* be the Current State and *ST* be the Stack for saving states.
2. Let *N* be the number of available moves for *S* and M_0 to M_n be the available moves.
3. Let *Best Move Value* = -1 (loss), and *Best Move* = undefined.
4. For every M_i from M_0 to M_n:
   A. Let Current Move be M_i
   B. Push *S* to *ST*
   C. Do Move M_i. Let *NP* be the Next Player to move and *NS* be the resulting state of move M_i.
      a. If *NP* is opponent player, *Current Move Value* = GetMin(*NS*,*0*);
      b. If *NP* is current player, *Current Move Value* = GetMax(*NS*,*0*);
      c. If *Current Move Value* > *Best Move Value*
         i. *Best Move Value* = *Current Move Value*
         ii. *Best Move* = M_i
   D. *S* = Top of *ST*
   E. Pop *ST*
5. If *Best Move* is undefined (Best worst-possible-move value = loss), set *Best Move* = Random Move(from M_0 to M_n);
6. Return *Best Move*;

Figure 3.4 GetMove Pseudocode

Method GetMax(State S, Depth D) : Value

1. Let *S* be the Current State and *ST* be the Stack for saving states.
   A. If *S* is an End-game state OR *D* is greater than or equal to Depth Limit, return *End-game value for calling player*
2. Let *N* be the number of available moves for *S* and M_0 to M_n be the available moves.
3. Let *Best Move Value* = -1 (loss).
4. For every *M* from M_0 to M_n:
   A. Let Current Move be M_i
   B. Push *S* to *ST*
   C. Do Move M_i. Let *NP* be the Next Player to move and *NS* be the resulting state of move M_i.
      a. If *NP* is opponent player, *Current Move Value* = GetMin(*NS*,*D+1*);
      b. If *NP* is current player, *Current Move Value* = GetMax(*NS*,*D+1*);
      c. If *Current Move Value* > *Best Move Value*, set *Best Move Value* = *Current Move Value*
   D. *S* = Top of *ST*
   E. Pop *ST*
5. Return *Best Move Value*;

Figure 3.5 GetMax Pseudocode

Method GetMin(State S, Depth D) : Value

1. Let *S* be the Current State and *ST* be the Stack for saving states.
   A. If *S* is an End-game state OR *D* is greater than or equal to Depth Limit, return *End-game value for calling player*
2. Let *N* be the number of available moves for *S* and M_0 to M_n be the available moves.
3. Let *Worst Move Value* = 1 (win).
4. For every *M* from M_0 to M_n:
   A. Let Current Move be M_i
   B. Push *S* to *ST*
   C. Do Move M_i. Let *NP* be the Next Player to move and *NS* be the resulting state of move M_i.
      a. If *NP* is opponent player, *Current Move Value* = GetMin(*NS*,*D+1*);
      b. If *NP* is current player, *Current Move Value* = GetMax(*NS*,*D+1*);
      c. If *Current Move Value* < *Worst Move Value*, set *Worst Move Value* = *Current Move Value*
   D. *S* = Top of *ST*
   E. Pop *ST*
5. Return *Worst Move Value*;

Figure 3.6 GetMin Pseudocode

As an example for the Minimax Algorithm, we will try to run it to the Congklak game, with a depth-limit of 2. Depth-2 means that the algorithm can only see 2 turns forward from current turn. To evaluate Depth-2 Congklak Evaluation Tree means to evaluate 36 leaf nodes and retrieve the value up to the root. Even after evaluating Depth-2 evaluation tree, there are still some moves considered as best move (the best move candidate usually is only one). This marks either Depth-2 evaluation tree is not enough to be used as a determining factor or there are several best moves (all candidate moves lead to a winning end-game).

By simple observation, humans know that the best move is to pick Node A, because its entire child has 0 as the second player's total seed. But while using a standard minimax, the evaluation value only knows three values: -1 if first player's seed is less than second player's, 1 if first player's seed is greater than second player's, and 0 if first player's seed is the same as second player's, so the only way to know which move is better is to increase the evaluation depth.

An interesting thing to note from this evaluation tree is that the Min and Max nodes are not alternating by parent-child interval. This is because in Congklak a player may get consecutive turns to play; not alternating as some other games.

B. Minimax Algorithm with Alpha-Beta Pruning

Minimax Algorithm with Alpha-beta pruning implementation are almost identical with standard Minimax Algorithm, their only differences are in parameters as the upper and lower value bound and their usage as boundary:

- GetMove (State S) : Move
- GetMax (State S, MinimumBound Alpha, MaximumBound Beta, Depth D)
- GetMin (State S, MinimumBound Alpha, MaximumBound Beta, Depth D)

Alpha-Beta Pruning's methods are almost identical to the normal Minimax counterpart save for Minimum (Alpha) and Maximum (Beta) bounds. These implementations also have a depth-limitation. Figure 3.7, 3.8 and 3.9 shows the pseudocode for the implementations.

```
Method GetMove(State):

1. Let S be the Current State and ST be the Stack for saving states.
2. Let N be the number of available moves for S and M0 to Mn be the available
   moves.
3. Let Alpha = -INFINITE, and Beta = INFINITE; Let Best Move = unidentified
4. For every M from M0 to Mn:
   A. Let Current Move be Mi
   B. Push S to ST
   C. Do Move Mi. Let NP be the Next Player to move and NS be the resulting state
      of move Mi.
      a. If NP is opponent player, Current Move Value = GetMin(NS, Alpha,
         Beta,D+1);
      b. If NP is current player, Current Move Value = GetMax(NS, Alpha,
         Beta,D+1);
      c. If Current Move Value > Alpha
         i.  Alpha = Current Move Value
         ii. Best Move = Mi
      d. If (Alpha >= Beta) break; Prune happens.
   D. S = Top of ST
   E. Pop ST
5. If Best Move is undefined (Best worst-possible-move value = loss), set Best Move
   = Random Move(from M0 to Mn);
   A. Return Best Move;
```

Figure 3.7 Alpha-Beta Pruning GetMove Pseudocode

Figure 3.10 shows the Minimax with Alpha-Beta Pruning Evaluation Tree. Compared to Minimax's evaluation tree, alpha-beta pruning's evaluation tree produces much simpler tree. Alpha-Beta only explores three nodes and Minimax explores 36 nodes; on top of that, Minimax still has several candidates as the best move, meaning it still has to further explore to know which candidate are the best – and Alpha-Beta has one definite candidate, which candidate also included in Minimax's candidate.

```
Method GetMax(State S, Alpha, Beta,D):

1. Let S be the Current State and ST be the Stack for saving states.
   A. If S is an End-game state OR D is greater than or equal to Depth limit, return
      End-game value for calling player
2. Let N be the number of available moves for S and M0 to Mn be the available
   moves.
3. For every M from M0 to Mn:
   A. Let Current Move be Mi
   B. Push S to ST
   C. Do Move Mi.
   D. Let NP be the Next Player to move and NS be the resulting state of move Mi.
      a. If NP is opponent player, Current Move Value = GetMin(NS, Alpha,
         Beta,D+1);
      b. If NP is current player, Current Move Value = GetMax(NS, Alpha,
         Beta,D+1);
      c. If Current Move Value > Alpha, set Alpha = Current Move Value
      d. If (Alpha>= Beta) break; Prune happens.
   E. S = Top of ST
   F. Pop ST
4. Return Alpha;
```

Figure 3.8Alpha-Beta Pruning GetMax Pseudocode

```
Method GetMin(State S, Alpha, Beta):

1. Let S be the Current State and ST be the Stack for saving states.
   A. If S is an End-game state OR D is greater than or equal to Depth limit ,
      return End-game value for calling player
2. Let N be the number of available moves for S and M0 to Mn be the available
   moves.
3. For every M from M0 to Mn:
   A. Let Current Move be Mi
   B. Push S to ST
   C. Do Move Mi.
   D. Let NP be the Next Player to move and NS be the resulting state of move
      Mi.
      a. If NP is opponent player, Current Move Value = GetMin(NS,
         Alpha, Beta, D+1);
      b. If NP is current player, Current Move Value = GetMax(NS, Alpha,
         Beta, D+1);
      c. If Current Move Value < Beta, set Beta = Current Move Value
      d. If (Alpha >= Beta) break; Prune happens.
   E. S = Top of ST
   F. Pop ST
   G. Return Current Min;
```

Figure 3.9 Alpha-Beta Pruning GetMove Pseudocode

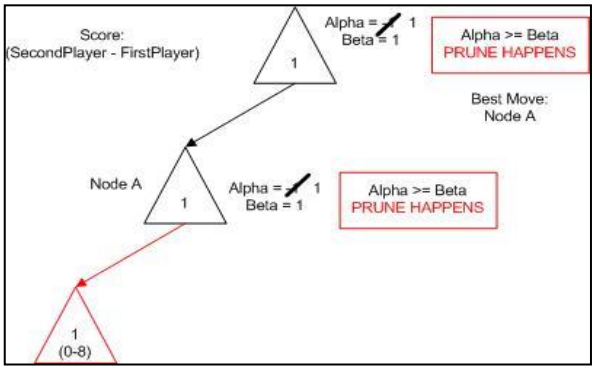

Figure 3.10 Depth-2 Alpha-Beta Pruning Evaluation Tree

IV. USER INTERFACE DEVELOPMENT

User Interface Development is a divided into several sections: Main Menu (Figure 4.1), Settings Window (Figure 4.2), About Window, and Game Window (Figure 4.3). All of the User Interface in this application are handled by QtCreator and therefore stored in .ui XML file. The only exception for this is the UI for Game Window; it is dynamic in terms of button and label placement, and because of that is written in a cpp code.

A. *Main Menu*

This is where the user will be directed to every time the application runs. Figure 4.1 shows in QtCreator how the design is. Figure 4.2 shows the screen-shot when the application runs in reality.

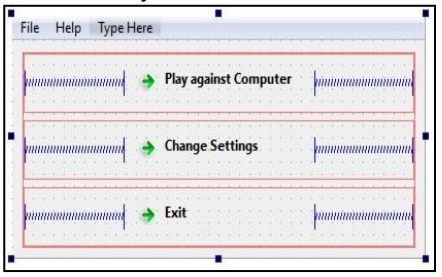

Figure 4.1 Main Menu UI Design

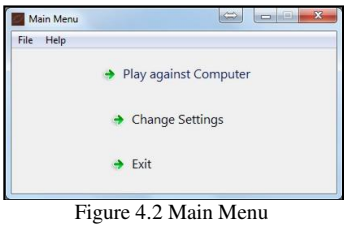

Figure 4.2 Main Menu

B. *Change Setting*

This is the window for the user to change the settings. Figure 4.3 shows in QtCreator how the design and on the right side shows the Change Settings interface in the application.

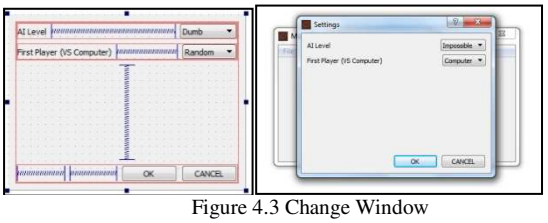

Figure 4.3 Change Window

C. *Game Window*

This is the window for the user to play the game. Figure 4.4 shows the in-game Play with Computer UI.

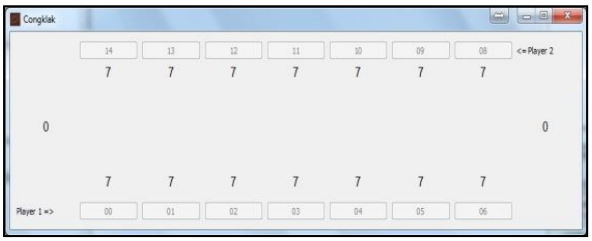

Figure 4.4 Game Window

V. ALGORITHM IMPLEMENTATION

1. Minimax Algorithm

Implementation of MiniMax Algorithm usually involves usage of three methods:

- GetMove (State S) : Move ➔ get the best move for current state
- GetMax (State S, Depth D) : Value ➔ get the best move value for evaluated state
- GetMin (State S, Depth D) : Value ➔ get the worst move value for evaluated state

GetMax and GetMin implementations have their current evaluation-depth as their parameter. This parameter will be compared to a global depth-limit variable to limit the maximum depth to be evaluated.

Figure 5.1, 5.2, and 5.3 shows the pseudo-code for these three methods. GetMove, GetMax, and GetMin are all similar. They all compares and retrieves value from Minimax Search Tree's leaf nodes; their difference lies in: GetMove retrieves a move instead of an evaluation value, GetMax retrieves the best evaluation value, and GetMin retrieves the worst evaluation value.

```
Method GetMove(State S):

1. Let S be the Current State and ST be the Stack for saving states.
2. Let N be the number of available moves for S and M0 to Mn be the available moves.
3. Let Best Move Value = -1 (loss), and Best Move = undefined.
4. For every Mi from M0 to Mn:
   A. Let Current Move be Mi
   B. Push S to ST
   C. Do Move Mi. Let NP be the Next Player to move and NS be the resulting state
      of move Mi.
      a. If NP is opponent player, Current Move Value = GetMin(NS,0);
      b. If NP is current player, Current Move Value = GetMax(NS,0);
      c. If Current Move Value > Best Move Value
         i.  Best Move Value = Current Move Value
         ii. Best Move = Mi
   D. S = Top of ST
   E. Pop ST
5. If Best Move is undefined (Best worst-possible-move value = loss), set Best Move
   = Random Move(from M0 to Mn);
      d. Return Best Move;
```
Figure 5.1 GetMove Pseudocode

```
Method GetMax(State S, Depth D) : Value

1. Let S be the Current State and ST be the Stack for saving states.
   A. If S is an End-game state OR D is greater than or equal to Depth Limit, return
      End-game value for calling player
2. Let N be the number of available moves for S and M0 to Mn be the available moves.
3. Let Best Move Value = -1 (loss).
4. For every M from M0 to Mn:
   A. Let Current Move be Mi
   B. Push S to ST
   C. Do Move Mi. Let NP be the Next Player to move and NS be the resulting state
      of move Mi.
      a. If NP is opponent player, Current Move Value = GetMin(NS,D+1);
      b. If NP is current player, Current Move Value = GetMax(NS,D+1);
      c. If Current Move Value > Best Move Value, set Best Move Value =
         Current Move Value
   D. S = Top of ST
   E. Pop ST
Return Best Move Value;
```
Figure 5.2 GetMax Pseudocode

```
Method GetMin(State S, Depth D) : Value

1. Let S be the Current State and ST be the Stack for saving states.
   A. If S is an End-game state OR D is greater than or equal to
      Depth Limit, return End-game value for calling player
2. Let N be the number of available moves for S and M0 to Mn be
   the available moves.
3. Let Worst Move Value = 1 (win).
4. For every M from M0 to Mn:
   A. Let Current Move be Mi
   B. Push S to ST
   C. Do Move Mi. Let NP be the Next Player to move and NS
      be the resulting state of move Mi.
      a. If NP is opponent player, Current Move Value =
         GetMin(NS,D+1);
      b. If NP is current player, Current Move Value =
         GetMax(NS,D+1);
      c. If Current Move Value < Worst Move Value, set Worst
         Move Value = Current Move Value
   D. S = Top of ST
   E. Pop ST
5. Return Worst Move Value;
```
Figure 5.3 GetMin Pseudocode

As an example for the MiniMax Algorithm, we will try to run it to the Congklak game, with a depth-limit of 2. Figure 5.4 shows the state which the MiniMax algorithm ran at, A to G respectively becomes the Node A to Node G.

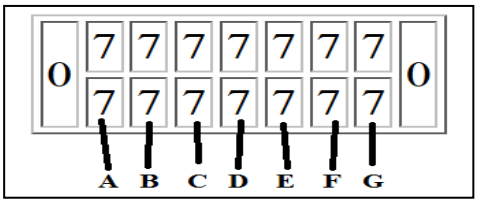

Figure 5.4 Evaluated Congklak Game State

Depth-2 means that the algorithm can only see 2 turns forward from current turn. To evaluate Depth-2 Congklak Evaluation Tree means to evaluate 36 leaf nodes and retrieve the value up to the root. Even after evaluating Depth-2 evaluation tree, there are still some moves considered as best move (the best move candidate usually is only one). This marks either Depth-2 evaluation tree is not enough to be used as a determining factor or there are several best moves (all candidate moves lead to a winning end-game).

By simple observation, humans know that the best move is to pick Node A, because its entire child has 0 as the second player's total seed. But while using a standard minimax, the evaluation value only knows three values: -1 if first player's seed is less than second player's, 1 if first player's seed is greater than second player's, and 0 if first player's seed is the same as second player's, so the only way to know which move is better is to increase the evaluation depth.

2. Minimax Algorithm with Alpha-Beta Pruning

MiniMax Algorithm with Alpha-beta pruning implementation are almost identical with standard MiniMax Algorithm, their only differences are in parameters as the upper and lower value bound and their usage as boundary:

- GetMove (State S) : Move
- GetMax (State S, MinimumBound Alpha, MaximumBound Beta, Depth D)
- GetMin (State S, MinimumBound Alpha, MaximumBound Beta, Depth D)

Alpha-Beta Pruning's methods are almost identical to the normal MiniMax counterpart save for Minimum (Alpha) and Maximum (Beta) bounds. These implementations also have a depth-limitation. Figure 5.5, 5.6 and 5.7 shows the pseudocode for the implementations.

```
Method GetMove(State):

1. Let S be the Current State and ST be the Stack for saving states.
2. Let N be the number of available moves for S and M0 to Mn be the available
   moves.
3. Let Alpha = -INFINITE, and Beta = INFINITE; Let Best Move = unidentified
4. For every M from M0 to Mn:
   A. Let Current Move be Mi
   B. Push S to ST
   C. Do Move Mi. Let NP be the Next Player to move and NS be the resulting
      state of move Mi.
      a. If NP is opponent player, Current Move Value = GetMin(NS, Alpha,
         Beta,D+1);
      b. If NP is current player, Current Move Value = GetMax(NS, Alpha,
         Beta,D+1);
      c. If Current Move Value > Alpha
         i.  Alpha = Current Move Value
         ii. Best Move = Mi
      d. If (Alpha >= Beta) break; Prune happens.
   D. S = Top of ST
   E. Pop ST
5. If Best Move is undefined (Best worst-possible-move value = loss), set Best
   Move = Random Move(from M0 to Mn);
   A. Return Best Move;
```

Figure 5.5 Alpha-Beta Pruning GetMove Pseudocode

```
Method GetMax(State S, Alpha, Beta,D):

1. Let S be the Current State and ST be the Stack for saving states.
   A. If S is an End-game state OR D is greater than or equal to
      Depth limit, return End-game value for calling player
2. Let N be the number of available moves for S and M0 to Mn be
   the available moves.
3. For every M from M0 to Mn:
   A. Let Current Move be Mi
   B. Push S to ST
   C. Do Move Mi.
   D. Let NP be the Next Player to move and NS be the resulting
      state of move Mi.
      a. If NP is opponent player, Current Move Value =
         GetMin(NS, Alpha, Beta,D+1);
      b. If NP is current player, Current Move Value =
         GetMax(NS, Alpha, Beta,D+1);
      c. If Current Move Value > Alpha, set Alpha = Current
         Move Value
      d. If (Alpha>= Beta) break; Prune happens.
   E. S = Top of ST
   F. Pop ST
Return Alpha;
```

Figure 5.6 Alpha-Beta Pruning GetMax Pseudocode

```
Method GetMin(State S, Alpha, Beta):

1. Let S be the Current State and ST be the Stack for saving states.
   B. If S is an End-game state OR D is greater than or equal to Depth limit ,
      return End-game value for calling player
2. Let N be the number of available moves for S and M0 to Mn be the available
   moves.
3. For every M from M0 to Mn:
   A. Let Current Move be Mi
   B. Push S to ST
   C. Do Move Mi.
   D. Let NP be the Next Player to move and NS be the resulting state of move
      Mi.
      a. If NP is opponent player, Current Move Value = GetMin(NS,
         Alpha, Beta, D+1);
      b. If NP is current player, Current Move Value = GetMax(NS, Alpha,
         Beta, D+1);
      c. If Current Move Value < Beta, set Beta = Current Move Value
      d. If (Alpha >= Beta) break; Prune happens.
   E. S = Top of ST
   F. Pop ST
   G. Return Current Min;
```

Figure 5.7 Alpha-Beta Pruning GetMin Pseudocode

Figure 5.8 shows the Minimax with Alpha-Beta Pruning Evaluation Tree. Alpha-Beta only explores three nodes and MiniMax explores 36 nodes; on top of that, Minimax still

has several candidates as the best move, meaning it still has to further explore to know which candidate are the best – and Alpha-Beta has one definite candidate, which candidate also included in Minimax's candidate.

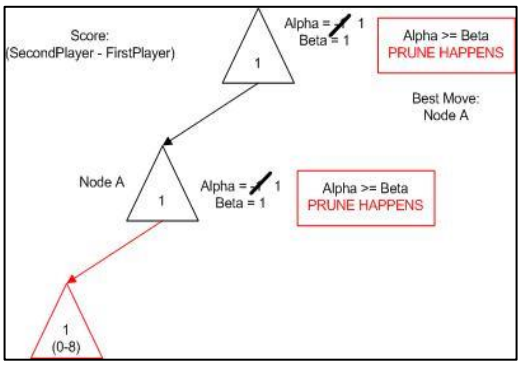

Figure 5.8 Depth-2 Alpha-Beta Pruning Evaluation Tree

VI. SYSTEM TESTING

The test to be run is based on the system's functionalities. Application can run in any Windows 7 PC and Laptop. User can choose the desired option: Play Game, Change Settings, View About, and Exit Game. In menu Play Game, user can play Congklak with the computer, user input signaled by button presses. In Change Settings menu, user can change AI Level and which player moves first for the game.

Testing Scenario is a conducted in several sections: Main Menu Handling Section, Play with Computer Handling Section, Change Settings Handling Section, View About Handling Section, and Exit Handling Section.

Table 6.1 Main Menu Handling Section

No.	Scenario	Expected Result	Result
1.	Play game with computer. Click "Play with computer" button.	Start a new congklak game in a separate window. Main menu window is closed.	As expected
2.	Change Settings. Click "Change Settings" button.	Change settings window is loaded.	As expected
3.	Exit menu. Click "Exit" button.	Main window closes and application terminates.	As expected
4.	View About. Click "Help" ➔ "About" from the toolbar.	Opens about window	As expected

Table 6.2 Play with Computer Handling Section

No.	Scenario	Expected	Result
		Result	
1.	Closes Main Menu	The system closes main menu window when this window is opened	As expected
2.	AI's Turn	AI picks and moves the beads. Player input button is locked.	As expected
3.	Player's Turn	Player's input buttons for legal moves are unlocked.	As expected
4.	Player Moves Player clicks on one of the input buttons.	The input buttons are locked and the game starts to distribute beads from the selected hole.	As expected
5.	Detect Soft End-Game The state where a player gains half of maximum beads+1 reached.	System states that the winner is already obvious and asks user to keep on playing or end the game	As expected
6.	Soft End Game Player chooses to end game when the application detects soft end-game	The game ends and displays the winner.	As expected
7.	Persistent Game Player chooses to still play the game even after the application detects soft end-game.	The game continues.	As expected
8.	End Game	The game displays the winner.	As expected

Table 6.3 Change Settings Handling Section

No.	Scenario	Expected Result	Result
1.	Changes the value of AI Level. Pick the AI Level from combo-box	AI Level is changed.	As expected
2.	Changes the value of First Player. Pick the Player Who Moves First from combo-box	First Player is changed.	As expected

JuTISI
Jurnal Teknik Informatika dan Sistem Informasi

3.	Save settings. Click "OK" button.	Window closes and the setting is saved.	As expected
4.	Cancel settings Click "Cancel" button.	Window closes and the setting is NOT saved.	As expected

Time taken by the AI to perform and calculate the optimal moves for their respective difficulties are listed on Table 6.4. Only first five moves are recorded for all difficulties of AI. The lowest AI difficulty is not listed because it utilizes random function, not some algorithm.

Table 6.4 Time Needed for Generating AI Move

No	Description	AI Level Normal - Release (in seconds)	AI Level Expert - Release (in seconds)	AI Level Impossible - Release (in seconds)
1	First Move	0.32	0.93	1.10
2	Second Move	0.29	0.7	0.7
3	Third Move	0.28	0.64	1.29
4	Fourth Move	0.34	0.32	1.50
5	Fifth Move	0.27	0.35	0.3

VII. CONCLUSIONS

There are several conclusions gained from the research:

1) It is possible to implement MiniMax with Alpha-Beta Pruning as Computer Player in Congklak.

2) The computer will always win if its difficulty is set to impossible and is given the first move.

3) Time needed for AI move is not too long, the longest time it took are below two seconds.

REFERENCES

[1] Ahmad Fahmi Hj. Mohamad, Jessifa Joana Mohd Supian, Muhammad Syafiq Md Ribuan "Digital Congklak: The art, play and experience framework", Proceedings of the 10th International Conference on Ubiquitous Information Management and Communication Article No. 88 ACM New York, 2016

[2] Bruce Rosen, CS 161 Recitation Notes - Minimax with Alpha Beta Pruning, http://www.cs.ucla.edu/~rosen/161/notes/alphabeta.html. 2009

[3] Cofer, Adam. MancalaIn Java: Experiment in Artificial Intelligence and Game Playing. Departmental Honor Thesis. University of Tennesse at Chattanooga, 2003

[4] Hasibuan, Z. A., Isal, Y. K., N.C., B. A., Ahmad, M. & Selviandro, N. Preservation of Cultural Heritage and Natural History through Game Based learning. International Journal of Machine Learning and Computing, 1, 460 - 465. 2011

[5] Joseph Antonius Maria Nijssen, "Monte-Carlo Tree Search for Multi-Player Games", Universiteit Maastricht, 2013

[6] Noraziah ChePa, Asmidah Alwi, Aniza Mohamed Din, and Muhammad Safwan, "The Application of Neural Networks And Min-Max Algorithm In Digital Congkak", Proceedings of the 4th International Conference on Computing and Informatics, ICOCI 201328-30, Sarawak, Malaysia. Universiti Utara Malaysia, 2013

[7] Pascal Chorus. Implementing a computer player for abalone using alpha-beta and monte-carlo search. Master thesis, Maastricht University, 2009.

[8] Russell,Stuart and Norvig, Peter, Artificial Intelligence: A Modern Approach (3rd Edition), 2009

[9] Randle, O., Ogunduyile O., Zuva T, Fashola N.A. A Comparison of the Performance of Supervised and Unsupervised Machine Learning Techniques in evolving Awale/Mancala/Ayo Game Player, International Journal of Game Theory and Technology (IJGTT), 1(Jan.2013), 1—10, 2013

[10] Samuel H. Fuller, John G. Gaschnig, Gillogly, "Analysis of the alpa-beta prunning, School of Computer Science, Carnegie Mellon University, 1973.

[11] S. Russel and P. Norvig, Artificial Intelligence – A Modern Approach. Prentice-Hall Inc., Englewood Cliffs, NY, USA, 1995

[12] Sheryl C., Neil M. Retrieved from Predictive Eye Movements in playing Congkak.: DOI = http://www.myriad.com.my/RTAD/documents/RSpdfs/sherylchong.pdf. 2012

[13] Takeo Nakagawa, Hiroyuki Lida "Game Information Dynamics and Its Application to Congklak and Othello", International Journal of Mathematical Modeling and Applied Computing Vol. 1, No. 6, PP:51 -64, ISSN: 332 -3744 (ONLINE), 2013