PROJECT 2: KALAH GAME WITH MINMAX – AB AND ALPHABETA SEARCH ALGORITHMS

Submitted By:
Satish Mallikarjuna Padaganur

Student ID: A05006271

Under the Guidance Of:

Dr. Moonis Ali

Department of Computer Science Texas State University
San Marcos, Texas 78666

CS5346

SPRING 2020

TABLE OF CONTENTS

PROBLEM DESCRIPTION

The objective of this project is to design a kalah game using Minimax search and Alpha-Beta pruning algorithms with the help of an efficient evaluation function, so that the moves made in the game are intelligent.

Develop a Kalah game by using MINMAX-A-B, and ALPHA-BETA algorithms in C or C++ or Java programming language. Write DEEP-ENOUGH (with reasonable depth) and MOVE-GEN functions. An efficient evaluation function must be written in the code so that the stones in the game make intelligent moves. Execute programs with the data and analyze the performance of each algorithm, each evaluation function and each depth by tabulating the total length of the game path, total number of nodes generated and expanded, execution, the size of memory used by the program, and winning/losing statistics. For a team of two students, two evaluation functions, two algorithms and two depths, you will be executing programs eight times. Analyze the results and determine which algorithm and which evaluation function performs better. Program should play with a human to computer as well as computer to computer.

TEAM MEMBERS CONTRIBUTION

This project was brought to life with a team effort. The team members involved are:

* Nagendra Kammari
* Rohit Goud Lode
* Satish Mallikarjuna Padaganur

The contributions made to the project by each team member are as follows:

* All of us together spent time on understanding the project requirement. We shared our ideas about understanding of the algorithms and Kalah game.
* We came up with a draft version of project structure.
* I worked on writing code for MINMAX-AB algorithm.
* Rohit worked on writing code for ALPHBETA SEARCH algorithm.
* Nagendra worked on writing code for board function (Board.java).
* Rohit worked on implementing evaluation function 1.
* I worked on implementing evaluation function 2.
* Nagendra worked on implementing evaluation function 3.
* We all combinedly worked on writing code for GameEngine functions.
* Nagendra developed a GUI which includes writing code for GameGUI(), StaticGUI() functions (GameGUI.java, StaticGUI.java files).
* We all worked on integrating the program files.
* We all individually ran the programs for Kalah game to verify the correctness of the implementations.

KALAH GAME DESCRIPTION

History of Kalah game:

The Kalah game also called as Mancala game is one of the oldest known games to still be widely played today. Mancala is a generic name for a family of two-player turn-based strategy board games played with small stones, beans, or seeds and rows of holes or pits in the earth, a board or other playing surface. The objective is usually to capture all, or some set of the opponent's pieces. Versions of the game date back to the 7th century and evidence suggests the game existed in ancient Egypt.

Evidence of the game was uncovered in Israel in the city of Gedera in an excavated Roman bathhouse where pottery boards and rock cuts were unearthed dating back to between the 2nd and 3rd century AD. Among other early evidence of the game are fragments of a pottery board and several rock cuts found in Aksumite areas in Matara (in Eritrea) and Yeha (in Ethiopia), which are dated by archaeologists to between the 6th and 7th century AD; the game may have been mentioned by Giyorgis of Segla in his 14th century Ge'ez text Mysteries of Heaven and Earth, where he refers to a game called qarqis, a term used in Ge'ez to refer to both Gebet'a (mancala) and Sant'araz (modern sent'erazh, Ethiopian chess).

Kalah Game Basics:

The Kalah game is played between two players. Kalah is played on a board of two rows, each consisting of six pits that have a large store at either end called kalah. A player owns the six pits closest to him and the kalah on his right side. Beginners may start with three seeds in each pit, but the game becomes more and more challenging by starting with 4, 5 or up to 6 pieces in each pit. Today, four seeds per hole has become the most common variant, but Champion recommended the expert game which has 6 pieces in each pit.

Kalah Game Rules step by step:

Step 1:

The objective of the game is to collect as many playing pieces as possible before one of the players clears their side of all the playing pieces. The row of six cups in front and closest to each player are theirs.

Step 2:

Start by placing four stones in each small cup. You have 72 stones total, and 12 cups, which means there should be four stones in each cup. Each player starts off with a total of 36 stones or beads.

Step 3:

Your Kalah is the big basin to your right. Also called a "store," it is where captured pieces are placed.

Step 4:

Choose which player is going to go first. Because there's not really an advantage to going first, flip a coin or choose a person at random.

Step 5:

Going counter-clockwise, the beginning player takes all four stones in one cup on their side and places one stone each in any four adjacent cups. Players can only grab stones that are on their side.

Step 6:

Players can put stones in their own Kalah, but not in their opponent's Kalah. If you have enough stones to reach your opponent's Kalah, skip it.

Step 7:

Take turns picking up stones in any cup and placing them.

Step 8:

If your last stone falls into your Kalah, take another turn.

Step 9:

If the last stone you drop is in an empty cup on your side, capture that piece along with any pieces in the hole directly opposite. Captured pieces go into your Kalah store.

Step 10:

When one player's six cups are completely empty, the game ends. The player who still has stones left in their cups captures those stones and puts them in their Kalah. Players compare the number of stones in their Kalah. The player with the most stones wins.

METHODOLOGIES

As per the problem description, we are using MINMAX-A-B and ALPHA-BETA-SEARCH algorithms to develop programs and we are testing out code by playing Kalah game. We have also implemented a function to play the Kalah game between two Random Bots in which random numbers are generated to play the game.
Detailed descriptions of the methodologies are explained below.

MINIMAX

The minimax search procedure is a depth- first, depth-limited search procedure. The idea is to start at the current position and use the plausible-move generator to generate the set of possible successor positions. Now we can apply the static evaluation function to those positions and simply choose the best one.
After doing so, we can back that value up to the starting position to represent our evaluation of it. The starting position is exactly as good for us as the position generated by the best move, we can make next. Here we assume that the static evaluation function returns large values to indicate good situations for us, so our goal is to maximize the value of the static evaluation function of the next board position.
An example of this operation is shown in Fig. 12.1. It assumes a static evaluation function that returns values ranging from - 20 to 20, with 20 indicating a win for us, - 20 a win for the opponent, and 0 an even match. Since our goal is to maximize the value of the heuristic function, we choose to move to B. Backing B's value up to A, we can conclude that A's value is 8, since we know we can move to a position with a value of 8.
But since we know that the Static evaluation function is not completely accurate, we would like to carry the search farther ahead than one ply. This could be very

important, for example, in a chess game in which we are in the middle of a piece exchange. After our move, the situation would appear to be very good, but, if we look one move ahead, we will see that one of our pieces also gets captured and so the situation is not as favorable as it seemed. So, we would like to look ahead to see what will happen to each of the new game positions at the next move which will be made by the opponent. Instead of applying the static evaluation function to each of the positions that we just generated, we apply the plausible-move generator, generating a set of successor positions for each position. If we wanted to stop here, at two-ply look ahead, we could apply the static evaluation function to each of these positions, as shown in Fig. 12.2.

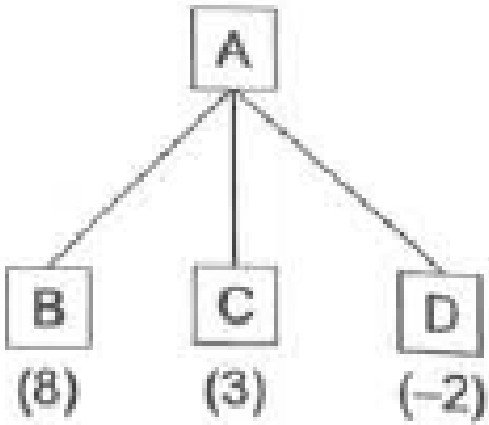


Fig. 12.1 *One-Ply Search*

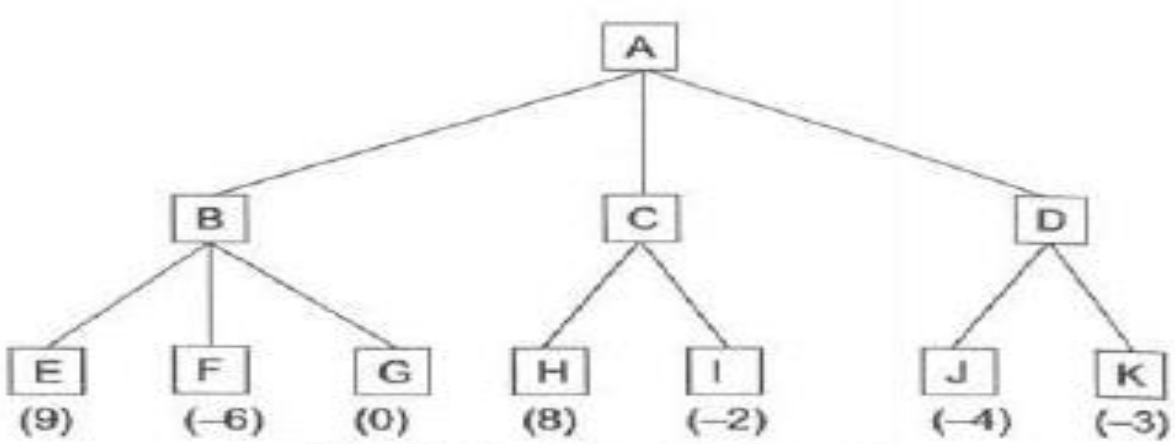


Fig. 12.2 *Two-Ply Search*

But now we must consider that the opponent gets to choose which successor moves to make and thus which terminal value should be backed up to the next level. Suppose we made move B. Then the opponent must choose among moves E, F, and G. The opponent's goal is to minimize the value of the evaluation function, so he or she can be expected to choose move F. This means that if we make move B, the actual position in which we will end up one move later is very bad for us. This is true even though a possible configuration is that represented by node E, which is very good for us. But since at this level we are not the ones to move, we will not get to choose it. Figure 12.3 shows the result of propagating the new values up the tree. At the level representing the opponent's choice, the minimum value was chosen and backed up. At the level representing our choice, the maximum value was chosen.

Once the values from the second ply are backed up, it becomes clear that the correct move for us to make at the first level, given the information we have available, is C, since there is nothing the opponent can do from there to produce a value worse than - 2. This process can be repeated for as many ply as time allows, and the more accurate evaluations that are produced can be used to choose the

correct move at the top level. The alternation of maximizing and minimizing an alternate ply when evaluations are being pushed back up corresponds to the opposing strategies of the two players and gives this method the name minimax. Having described informally the operation of the minimax procedure, we now describe it precisely. It is a straightforward recursive procedure that relies on two auxiliary procedures that are specific to the game being played:

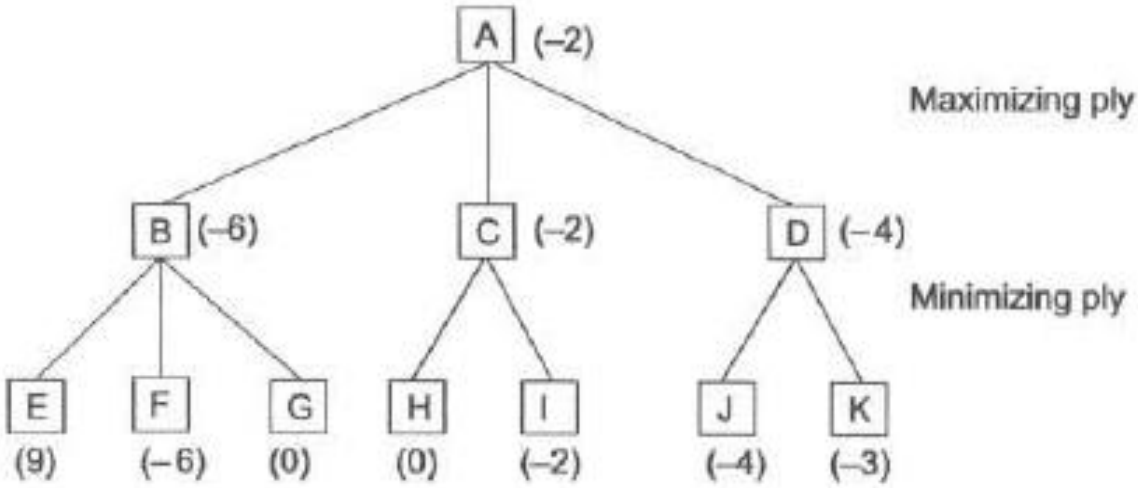


Fig. 12.3 *Backing Up the Values of a Two-Ply Search*

MINIMAX ALGORITHM

1. MOVEGEN(Position , Player)-The plausible-move generator, which returns a list of nodes representing the moves that can be made by Player in Position. We call the two players PLAYER-ONE and PLAYER TWO; in a chess program, we might use the names BLACK and WHITE instead.
2. STATIC(Position , Player) -The static evaluation function, which returns a number representing the goodness of Position from the standpoint of Player. As with any recursive program, a critical issue in the design of the MINIMAX procedure is when to stop the recursion and simply call the static evaluation function. There are a variety of factors that may influence this decision. They include:
    - Has one side won?
    - How many plies have we already explored?
    - How promising is this path?
    - How much time is left?
    - How stable is the configuration?

    For the general MINIMAX procedure discussed here, we appeal to a function, DEEP-ENOUGH, which is assumed to evaluate all of these factors and to return TRUE if the search should be stopped at the current level and FALSE otherwise. Our simple implementation of DEEP-ENOUGH will take two parameters, Position and Depth. It will ignore its Position parameter and simply return TRUE if its Depth parameter exceeds a constant cutoff value. One problem that arises in defining MINIMAX as a recursive procedure is that it needs to return not one but two results:

- The backed-up value of the path it chooses.
- The path itself. We return the entire path even though probably only the first element, representing the best move from the current position, is needed.

Algorithm: MINIMAX(Position, Depth, Player)

1. If DEEP-ENOUGH*(Position, Depth),* then return the structure
   VALUE = STATIC*(Position, Player);*
   PATH = nil
This indicates that there is no path from this node and that its value is that determined by the static evaluation function.

2. Otherwise, generate one more ply of the tree by calling the function MOVEGEN(Position Player) and setting SUCCESSORS to the list it returns.

3. If SUCCESSORS is empty, then there are no moves to be made, so return the same structure that would have been returned if DEEP-ENOUGH had returned true.
4. If SUCCESSORS is not empty, then examine each element in turn and keep track of the best one. This is done as follows.
   Initialize BEST-SCORE to the minimum value that STATIC can return. It will be updated to reflect the best score that can be achieved by an element of SUCCESSORS.

For each element SUCC of SUCCESSORS, do the following:
(a) Set RESULT-SUCC to
   MINIMAX(SUCC, *Depth* + 1, OPPOSITE(*Player))*
This recursive call to MINIMAX will actually carry out the exploration of SUCCC.

(b) Set NEW-VALUE to - VALUE(RESULT-SUCC). This will cause it to reflect the merits of the position from the opposite perspective from that of the next lower Ievel.

(c) If NEW-VALUE> BEST-SCORE, then we have found a successor that is better than any that

have been examined so far. Record this by doing the following:

(i) Set BEST-SCORE to NEW-VALUE.

(ii) The best-known path is now from CURRENT to SUCC and then on to the appropriate path down from SUCC as determined by the recursive call to MINIMAX. So set BEST-PATH to the result of attaching SUCC to the front of PATH(RESULT-SUCC).

5. Now that all the successors have been examined, we know the value of Position as well as which path to take from it. So, return the structure

VALUE = BEST-SCORE

PATH = BEST-PATH

ALPHA-BETA PRUNING

The problem with minimax search is that the number of game states it has to examine is exponential in the depth of the tree. Unfortunately, we can't eliminate the exponent, but it turns out we can effectively cut it in half. The trick is that it is possible to compute the correct minimax decision without looking at every node in the game tree. That is, we can borrow the idea of pruning from Chapter 3 to eliminate large parts of the tree from consideration. The particular technique we examine is called alpha–beta pruning. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.

Consider again the two-ply game tree from Figure 5.2. Let's go through the calculation of the optimal decision once more, this time paying careful attention to what we know at each point in the process. The steps are explained in Figure 5.5. The outcome is that we can identify the minimax decision without ever evaluating two of the leaf nodes.

Another way to look at this is as a simplification of the formula for MINIMAX. Let the two unevaluated successors of node C in Figure 5.5 have values x and y. Then the value of the root node is given by

$$MINIMAX(root) = max\begin{cases} min(3,12,8) \\ min(2,x,y) \\ min(14,5,2) \end{cases}$$

$$= max\begin{cases} 3 \\ min(2,x,y) \\ 2 \end{cases}$$

$$= max\begin{cases} 3 \\ z \ where\ z = min(2,x,y) \leq 2 = 3 \\ 2 \end{cases}$$

In other words, the value of the root and hence the minimax decision are independent of the values of the pruned leaves x and y.

Alpha–beta pruning can be applied to trees of any depth, and it is often possible to prune entire subtrees rather than just leaves. The general principle is this: consider a node n somewhere in the tree (see Figure 5.6), such that Player has a choice of moving to that node. If Player has a better choice m either at the parent node of n or at any choice point further up, then n will never be reached in actual play. So, once we have found out enough about n (by examining some of its descendants) to reach this conclusion, we can prune it.

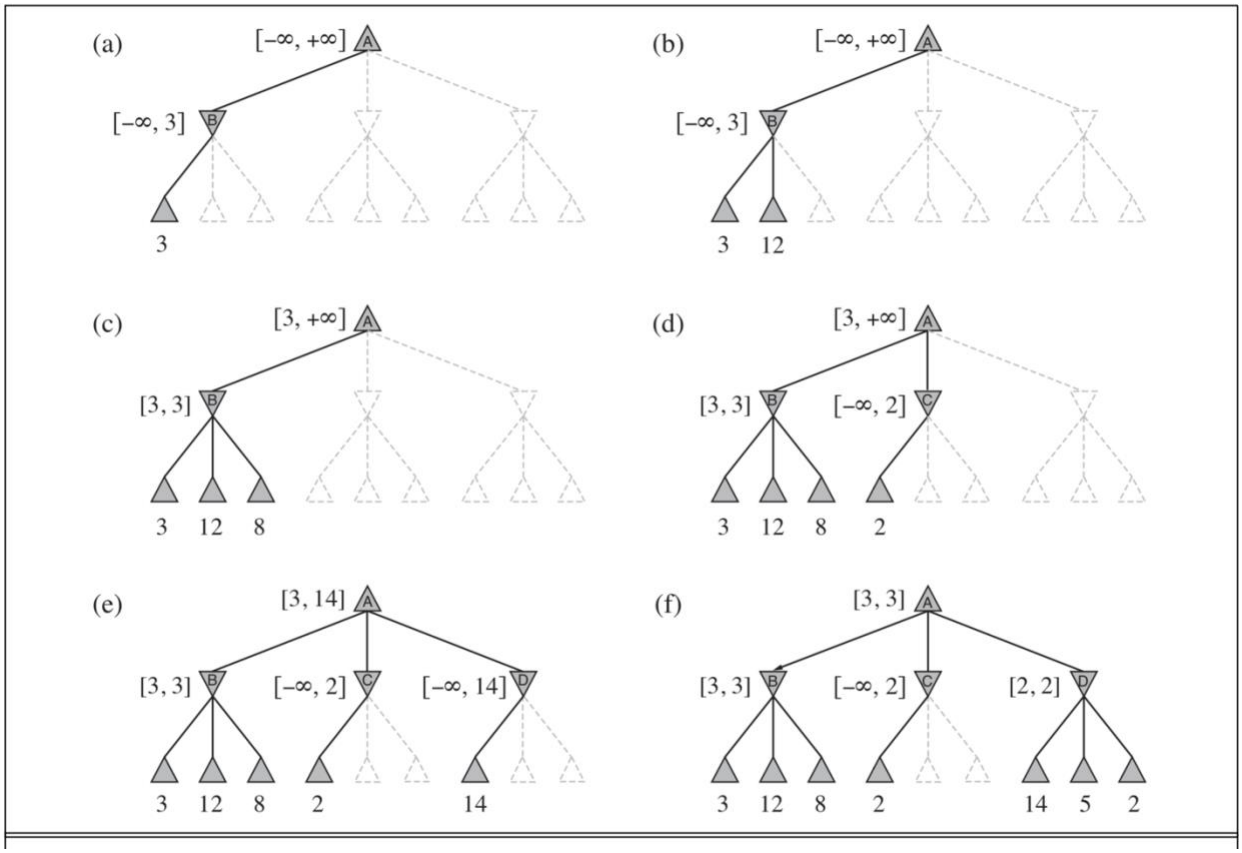


Figure 5.5 Stages in the calculation of the optimal decision for the game tree in Figure 5.2. At each point, we show the range of possible values for each node. (a) The first leaf below B has the value 3. Hence, B, which is a MIN node, has a value of *at most* 3. (b) The second leaf below B has a value of 12; MIN would avoid this move, so the value of B is still at most 3. (c) The third leaf below B has a value of 8; we have seen all B's successor states, so the value of B is exactly 3. Now, we can infer that the value of the root is *at least* 3, because MAX has a choice worth 3 at the root. (d) The first leaf below C has the value 2. Hence, C, which is a MIN node, has a value of *at most* 2. But we know that B is worth 3, so MAX would never choose C. Therefore, there is no point in looking at the other successor states of C. This is an example of alpha–beta pruning. (e) The first leaf below D has the value 14, so D is worth *at most* 14. This is still higher than MAX's best alternative (i.e., 3), so we need to keep exploring D's successor states. Notice also that we now have bounds on all of the successors of the root, so the root's value is also at most 14. (f) The second successor of D is worth 5, so again we need to keep exploring. The third successor is worth 2, so now D is worth exactly 2. MAX's decision at the root is to move to B, giving a value of 3.

Remember that minimax search is depth-first, so at any one time we just have to con- sider the nodes along a single path in the tree. Alpha–beta pruning gets its name from the following two parameters that describe bounds on the backed-up values that appear anywhere along the path:

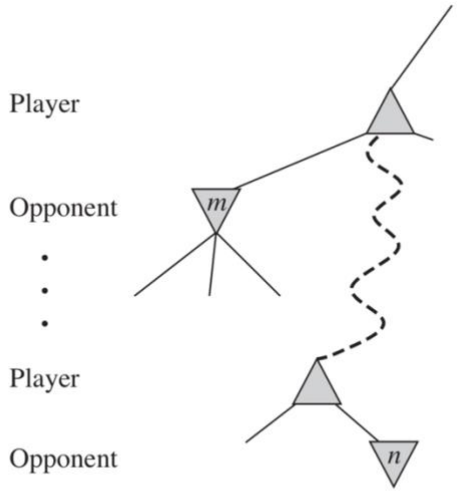


Figure 5.6 The general case for alpha–beta pruning. If m is better than n for Player, we will never get to n in play.

α = the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX.

β = the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN.

Alpha–beta search updates the values of α and β as it goes along and prunes the remaining branches at a node (i.e., terminates the recursive call) as soon as the value of the current node is known to be worse than the current α or β value for MAX or MIN, respectively. The complete algorithm is given in Figure 5.7. We encourage you to trace its behavior when applied to the tree in Figure 5.5.

ALPHA-BETA SEARCH ALGORITHM

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Figure 5.7 The alpha–beta search algorithm. Notice that these routines are the same as the MINIMAX functions in Figure 5.3, except for the two lines in each of MIN-VALUE and MAX-VALUE that maintain α and β (and the bookkeeping to pass these parameters along).

MINMAX A-B ALGORITHM

MINIMAX-A-B(Position, Depth, Player, Use-Thresh, Pass-Thresh)

1. If DEEP-ENOUGH(Position, Depth), then return the structure

   VALUE = STATIC (Position, Player);

   PATH = nil

2. Otherwise, generate one more ply of the tree by calling the function MOVE-GEN(Position, Player) and setting SUCCESSORS to the list it returns.
3. If SUCCESSORS is empty, there are no moves to be made; return the same structure that would have been returned if DEEP-ENOUGH had returned TRUE.
4. If SUCCESSORS is not empty, then go through it, examining each element and keeping track of the best one. This is done as follows.
5. For each element SUCC of SUCCESSORS:

   Set RESULT-SUCC to MINIMAX-A-B(SUCC, Depth + 1, OPPOSITE (Player), Pass-Thresh, - Use-Thresh)

   Set NEW-VALUE to - VALUE(RESULT-SUCC)

   If NEW-VALUE> Pass-Thresh, then we have found a successor that is better than any that have been examined so far. Record this by doing the following:

   i. Set Pass-Thresh to NEW-VALUE.

   ii. The best-known path is now from CURRENT to SUCC and then on to the appropriate path from SUCC as determined by the recursive call to MINIMAX-A-B.

iii. So set BEST-PATH to the result of attaching SUCC to the front of PATH(RESULT-SUCC).

iv. If Pass-Thresh (reflecting the current best value) is not better than Use-Thresh, then we should stop examining this branch. But both thresholds and values have been inverted. So if Pass-Thresh >= Use-Thresh, then return immediately with the value

VALUE = Pass-Thresh

PATH = BEST-PATH

6. Return the structure

VALUE = Pass-Thresh

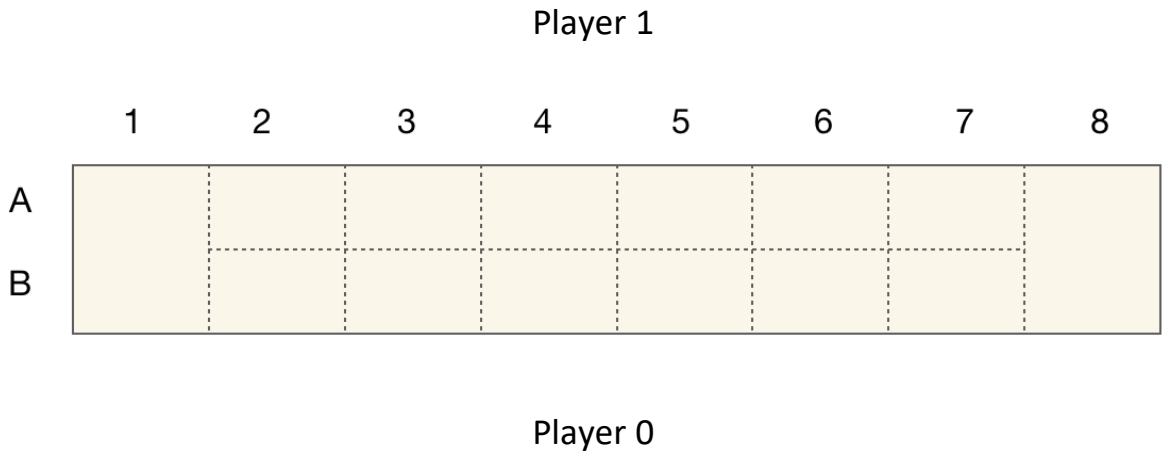
PATH = BEST-PATH

EVALUATION FUNCTIONS

EVALUATION FUNCTION WRITTEN BY SATISH:

The goal of the game is to collect maximum number of stones by the end of the game and to win the game; you need to collect more stones than your opponent. So, the evaluation function for any legal move is computed as the difference between the numbers of stones in both players' Kalah if that move is chosen.

EvaluationFunction(Player) = Number_of_Stones_player - Number_of_Stones_opponent



For example, for the current board position shown above, if Player-0 chooses pit B5 as his/her move, then the value of the evaluation function would EvaluationFunction (B5) = (1-0) = 1. Note: We can hence assume that our agent is always the "max" player.

The static evaluation function "**evaluationFunction2(Board, int)**;" used in our program is,

```
int evaluationFunction2(Board b, int player)
        {
                int player0Kalah = b.getBoardValueByPosition(0);
                int player1Kalah = b.getBoardValueByPosition(7);
                if(player == 0)
                {
                        if(player1Kalah != 0)
                        {
```

```
                    return (b.getBoardValueByPosition(0)-
b.getBoardValueByPosition(7));
                }
                else
                {
                    return -1;
                }
            }

            else
            {
                if(player0Kalah != 0)
                {
                    return (b.getBoardValueByPosition(7)-
b.getBoardValueByPosition(0));
                }
                else
                {
                    return -1;
                }
            }

        }
```

For simplicity and speed, this evaluation function is the best as it performs well. But the concept of evaluation function is very complex in terms of choosing evaluation function while playing Kalah game to make a best move.

Whenever player 0's turn comes to make a move, he/she just considers the difference between his/her kalah and opponents kalah.

That is, (b.getBoardValueByPosition(0)- b.getBoardValueByPosition(7))

And Whenever player 1's turn comes to make a move, he/she just considers the difference between his/her kalah and opponents kalah.

That is, (b.getBoardValueByPosition(7)- b.getBoardValueByPosition(0)).

EVALUATION FUNCTION WRITTEN BY ROHIT:

So, what I thought while developing function is choosing the move which had the biggest difference on our scoring well. I wanted our function to try and protect some of the stones left in play as they are automatically captured when the opponent runs out of moves. This tactic is not as effective in early or mid-game scenarios but proves very effective when reaching end game, although it is risky as pits can be captured but that is protected against as a move resulting in an opponent capture would cause a massive difference in scoring wells giving a massive difference in the result.

I wanted our function to have some value to the stones that the player had left in play but not to value them as significantly as the stones that had already been captured, this is where I decided that the use of a scaling factor would be incredibly useful, we added a scaling factor to multiply the stones already captured by in order to reflect the fact they had greater significance than seeds in play. We tested scaling factor from a range of values and multipliers and found that multiplying it by 2 had the best result.

This Evaluation function is ((total number of player1 stones + stones in player1 scoring well) - (total number of player2 stones + stones in player2 scoring well)) * weight + ((total number of player1 stones + stones in player1 scoring well) - (total number of player2 stones + stones in player2 scoring well)).

So, the board positions 1 to 7 are owned by player 1. And 8 to 13 by Player 2. It sums up the values from 1 to 7 stones into player1stones and 8 to 13 stone values into player2stones.

According to our game rules, boardposition(0) = Player1 scoring well, boardposition(7) = Player2 scoring well.

If player1 is playing then it takes difference of (player1stones+boardposition(0) – player2stones+boardposition(7))*2+(player1stones+boardposition(0) – player2stones+boardposition(7))

If player2 is playing then it takes difference of (player2stones+boardposition(7) – player1stones+boardposition(0))*2+(player2stones+boardposition(7)) – player1stones+boardposition(0)).

EVALUATION FUNCTION WRITTEN BY NAGENDRA:

We have total 3 evaluation functions implemented for the Kalah game developed, one by each team member. When I learned the importance of good evaluation function, I understood that "good ness" of any position in the game can be best calculated if I studied enough about the game. I downloaded the game in my mobile and played with 2nd player as computer, and people around as well. After playing enough times I understood that merely trying to collect stones in one's Kalah, though sounds like a simple evaluation function candidate, is not enough.

If you can make a move that increases the number of stones in your Kalah and at the same time increase stones in opposite player pits, it would benefit greatly in the end. Thanks to the improved evaluation function, player can enjoy the extra chances of winning.

Evaluation function:

```
Int EvaluationFunction3()

{

        A<-#stones in your kalah

        B<-#stones in opposite player pits

        Return A+B

}
```

For example, in initial position where every player has 6 stones in their pits, player could choose any pit from first to last to make a legal move since there are more than 0 stones in every pit. If used above evaluation function; the best move would be to take all the stones from last, drop one stone in player's kalah and remaining on opponent's pits.

SOURCE CODE IMPLEMENTATION

The main functionalities of our source code are "**Move-Gen**" and "**Deep-Enough**" functions which are the core part of implementation of MINMAX-A-B and ALPHA-BETA-SEARCH algorithms. The complete source code for playing Kalah game with these two algorithms has many functions which are 3 different evaluation function developed by each team member, playerpitchoice(), updateBoard(), getPossibleMoves(), MinMaxAB(), AlphaBeta(), MinMaxABAlphaBeta(), AlphaBetaMinMaxAB() to name a few.
The source code also includes GUI functionalities which is projects highlight. With GUI implementation the user interface with the code and results looks very beautiful.
Though all the functionalities are important part of the project's source code, we will concentrate on the main functionalities of the code wjich are "**Move-Gen**" and "**Deep-Enough**" functions.

Move-Gen function is a plausible-move generator function and is implemented to generate all the possible move that can be generated from the current position. And this function returns a list of nodes representing the moves that can be made by Player in Position. So, this function calls getPossibleMoves() function to collect all the moves possible at the current position.

ArrayList possible = getPossibleMoves(b, player);
loop through those moves and collects the moves in an arraylist, clones the Board object, calls another function named "move" that performs legal move and returns the Board object.

```
for(int i = 0; i < possible.size(); i++)
                {
                        Board b1 = new Board(b);
                        b1.move((int)(possible.get(i)));
                        boardStorage.add(b1);
                }
```

Finally, Move-Gen function returns an ArrayList of Board objects consists of all possible moves from given position.

```
ArrayList<Board> boardStorage = new ArrayList<Board>();
return boardStorage;
```

"getPossibleMoves" function loops through player's pits, examining number of stones in every pit, including the pit position if it contains positive number of stones and ignoring if not.

For 1st player.

```
for(int i = 1; i <= 6; i++)
{
        if(b.getBoardValueByPosition(i) > 0)
        {
                possible.add(i);
        }
}
```

For 2nd player.

```
for(int i = 8; i <= 13; i++)
{
        if(b.getBoardValueByPosition(i) > 0)
        {
                possible.add(i);
        }
}
```

Then it returns an arraylist consists of positions

```
return possible;
```

DEEP-ENOUGH function is implemented to return TRUE if the search should be stopped at the current level and FALSE otherwise.

Deep-Enough function has been implemented in two ways,

Firstly, the function considers integer value from user as "depth" which is formal parameter for this function. It returns TRUE if the game tree expanded by Move-Gen function matches with user input.

```
//return true if depth matched with user input false otherwise
        boolean deepEnough(int depth)
        {
                if(depth == new GamePlay().depth)
                {
                        return true;
                }
                else
                {
```

```
                return false;
        }
    }
```

Finally, function keeps track of number of plys generated as “currentDepth” and matches the integer value of “depth” taken as user input which are formal parameters for this function.

```
boolean deepEnoughAB(int depth, int currentDepth)
    {
        if(depth == currentDepth)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
```

Both algorithms make use of these two functions to terminate if deep-enough is satisfied or to generate next level nodes in the game tree.

The fancy function of this source code is GUI implementation. Instead of just making use of console inputs we used Graphical User Interface for user inputs to chose which algorithm shall be used for Kalah game play and player can choose different evaluation functions to play and we can use different depths. We can also choose which player can start the first move in the game.
Based on all these user inputs, source generates outputs and results will be displayed on the GUI.

SOURCE CODE

GameEngine.java

```
package txstate.cs.ai.kala;

import java.util.ArrayList;
import java.util.Random;
import java.util.Scanner;



class MiniMaxABStruct{
    int value;
    ArrayList<Board> boardPath;



    public MiniMaxABStruct()
    {
        this.value = 0;
        this.boardPath = new ArrayList<Board>();
    }
}

class ABData{
    int value;
    Board boardPath;
```

```
    public ABData()
    {
        this.value = 0;
    }
}

class UserPlayer
{
    String randomplayer;

    public UserPlayer()
    {
        randomplayer="";
    }

    public String getPlayerName()
    {
        return randomplayer;
    }

    public void setPlayerName(String tempStr)
    {
        randomplayer=tempStr;
    }
}

public class GameEngine {
```

```
int succLength;
int depth;
int numberofnodesgenerated1;
int numberofnodesgenerated2;
Board board = new Board();
ArrayList<UserPlayer> randomplayers = new ArrayList<UserPlayer>();
int currentPlayerIndex;
int winnerIndex;
int numberOfStonesInplayerOneKalah;
int numberOfStonesInplayerTwoKalah;
int stoneThreshold;
int halfPits;
int sidePits;
int numPits;
int initStones;
int minDepth;
int value;
int player;
int oppositePlayer;
int currentPath;
int currentDepth;
int EvalChoice;
ArrayList<ABData> arrayList = new ArrayList<ABData>();
StringBuilder str = new StringBuilder();
//constructor for kalah game
GameEngine()
{
```

```
    //Top player's currentPlayerIndex = 0;
    //Down Player's currentPlayerIndex = 1
    numberofnodesgenerated1=1;
    numberofnodesgenerated2=1;
    currentPlayerIndex = 1;
    numberOfStonesInplayerOneKalah = 0;
    numberOfStonesInplayerTwoKalah = 0;
    stoneThreshold = 36;
    halfPits = 7;
    sidePits = 6;
    numPits = 14;
    initStones = 6;
    winnerIndex = -1;
    minDepth = 0;
    succLength = 0;
    board.player = 0;
    board.oppositePlayer=1;
}
//constructor for kalah game
GameEngine(int dep, int pl)
{
    numberofnodesgenerated1=1;
    numberofnodesgenerated2=1;
    currentPlayerIndex = 0;
    numberOfStonesInplayerOneKalah = 0;
    numberOfStonesInplayerTwoKalah = 0;
    stoneThreshold = 36;
```

```
        halfPits = 7;
        sidePits = 6;
        numPits = 14;
        initStones = 6;
        winnerIndex = -1;
        minDepth = 0;
        succLength = 0;
        depth = dep;
        player = pl;
        oppositePlayer=1-pl;
        board.player = pl;
        board.oppositePlayer=1-pl;
        currentDepth = 0;
    }
    //initialize players
    void init()
    {
        for(int i = 0; i < 2; i++)
        {
            UserPlayer p = new UserPlayer();
            p.setPlayerName("Player#"+i);
            randomplayers.add(p);
        }
    }
    //--------------------Block start for player to player kalah game----------------------------------
--
    //checking for game over for user to user kalah game
```

```
    boolean isUserUserGameOver()
    {
        if(numberOfStonesInplayerOneKalah > stoneThreshold
||numberOfStonesInplayerTwoKalah > stoneThreshold)
        {
            return true;
        }
        else if(numberOfStonesInplayerOneKalah == stoneThreshold &&
numberOfStonesInplayerTwoKalah == stoneThreshold)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    //getting correct pit for the player
    int playerPitChoice(GameGUI frame)
    {
        int pitInput;
        //Scanner sc = new Scanner(System.in);
        System.out.println("Player "+(board.player+1)+" turn");
        frame.txtArea.append("Player "+(board.player+1)+" turn\n");
        pitInput = new Random().nextInt(7);
        System.out.println(pitInput);
        while(pitInput < 1 || pitInput > 6)
```

```
    {
        pitInput = new Random().nextInt(6);
        System.out.println(pitInput);
    }
    return pitInput;
}
//update board based on player choice
void updateBoard(int position, GameGUI frame)
{
    System.out.println("Expanded move #"+position);
    frame.txtArea.append("Expanded move #"+position);

    int playerKalah,opponentKalah, pitInput, posIndex;
    int trackPosition = position;
    int grabOppositePosition, numOfOppPositionStones;
    int numberOfStones = board.getBoardValueByPosition(position);
    board.setBoardValueByPosition(position, 0);
    playerKalah = currentPlayerIndex * halfPits;
    opponentKalah = halfPits - playerKalah;

    int tempPosition=0;
    if(position >= 1 && position <= 6)
    {
        tempPosition = 1;
    }
    else if(position >= 8 && position <= 13)
    {
```

```
            tempPosition = 8;
        }
        if(numberOfStones == 1)
        {
            trackPosition--;
            if(trackPosition < 0)
            {
                trackPosition = numPits-1;
            }
            if(trackPosition != opponentKalah)
            {
                numberOfStones--;
                if((board.getBoardValueByPosition(trackPosition) == 0 && trackPosition <= 6
&& trackPosition >= 1) && tempPosition == 1 && numberOfStones == 0)
                {
                    if(board.getBoardValueByPosition(trackPosition) == 0 && numberOfStones
== 0)
                    {
                        grabOppositePosition = numPits-trackPosition;
                        numOfOppPositionStones =
                        board.getBoardValueByPosition(grabOppositePosition);
                        board.setBoardValueByPosition(grabOppositePosition, 0);
                        board.setBoardValueByPosition(playerKalah,
                        board.getBoardValueByPosition(playerKalah) +
numOfOppPositionStones + 1);
                    }
                }
```

```
            else if(board.getBoardValueByPosition(trackPosition) == 0 && (trackPosition
<= 13 && trackPosition >= 8) && tempPosition == 8 && numberOfStones == 0)
            {
               if(board.getBoardValueByPosition(trackPosition) == 0 && numberOfStones
== 0)
               {
                  grabOppositePosition = numPits-trackPosition;
                  numOfOppPositionStones =
                  board.getBoardValueByPosition(grabOppositePosition);
                  board.setBoardValueByPosition(grabOppositePosition, 0);
                  board.setBoardValueByPosition(playerKalah,
                  board.getBoardValueByPosition(playerKalah) +
numOfOppPositionStones + 1);
               }
            }

            else
            {
               board.setBoardValueByPosition(trackPosition,
               board.getBoardValueByPosition(trackPosition) + 1);
            }
         }
      }
      else
      {
         while(numberOfStones > 0)
         {
```

```
            trackPosition--;
            if(trackPosition < 0)
            {
                trackPosition = numPits-1;
            }
            if(trackPosition != opponentKalah)
            {
                numberOfStones--;
                board.setBoardValueByPosition(trackPosition,
                board.getBoardValueByPosition(trackPosition) + 1);
            }
        }
    }

    if(trackPosition == playerKalah)
    {
        displayBoard(frame);
        pitInput = playerPitChoice(frame);
        if(currentPlayerIndex == 0)
        {
            board.player = 0;
            numberOfStones = board.getBoardValueByPosition(pitInput);
            if(numberOfStones > 0)
            {
                updateBoard(pitInput, frame);
            }
        }
```

```
            else if(currentPlayerIndex == 1)
            {
                board.player = 1;
                pitInput = pitInput+halfPits;
                numberOfStones = board.getBoardValueByPosition(pitInput);
                if(numberOfStones > 0)
                {
                    updateBoard(pitInput, frame);
                }
            }
        }
        numberOfStonesInplayerOneKalah =
board.getBoardValueByPosition(playerKalah);
        numberOfStonesInplayerTwoKalah =
board.getBoardValueByPosition(opponentKalah);

        if(trackPosition != playerKalah && !isUserUserGameOver())
        {
            currentPlayerIndex = 1-currentPlayerIndex;
            board.player = 1-board.player;
        }
    }
    //operating user to user kalah game
    void playUserUser(GameGUI frame)
    {
        int pitInput, position=0;
        int numberOfStones;
```

```
int playerKalah, opponentKalah;
playerKalah = currentPlayerIndex * halfPits;
opponentKalah = halfPits - playerKalah;
board.player = 1;
do
{
   pitInput = playerPitChoice(frame);
   if(currentPlayerIndex == 1)
   {
      position = pitInput+halfPits;
   }
   else if(currentPlayerIndex == 0)
   {
      position = pitInput;
   }
   numberOfStones = board.getBoardValueByPosition(position);
   if(numberOfStones > 0)
   {
      System.out.println("Expanded move #"+position);
      frame.txtArea.append("Expanded move #"+position+"\n");

      updateBoard(position, frame);
   }
   displayBoard(frame);
}while(!isUserUserGameOver());

if(numberOfStonesInplayerOneKalah == numberOfStonesInplayerTwoKalah)
```

```
        {
            System.out.println("Game has been drawn");
            frame.txtArea.append("Game has been drawn\n");
        }
        else
        {
            System.out.println("Player
"+randomplayers.get(currentPlayerIndex).getPlayerName()+" wins");
            frame.txtArea.append("Player
"+randomplayers.get(currentPlayerIndex).getPlayerName()+" wins\n");
        }
    }

    //--------------------Block end for player to player kalah game---------------------------------
-
    //--------------------Block Start for AI algorithm-----------------------------
    //checking game over for AI algorithm
    boolean isGameOver()
    {
        return board.isGameOver();
    }

    //------Block start for Minimax-Alpha-Beta Search-----//
    //return true if depth matched with user input else false
    boolean deepEnough(int depth)
    {
        if(depth == this.depth)
```

```
    {
        return true;
    }
    else
    {
        return false;
    }
}

//getting possible moves for the given player in the given board
ArrayList getPossibleMoves(Board b, int player)
{
    ArrayList possible = new ArrayList();
    if(player == 0)
    {
        for(int i = 1; i <= 6; i++)
        {
            if(b.getBoardValueByPosition(i) > 0)
            {
                possible.add(i);
            }
        }
    }

    else
    {
        for(int i = 8; i <= 13; i++)
```

```
        {
            if(b.getBoardValueByPosition(i) > 0)
            {
                possible.add(i);
            }
        }
    }
    return possible;
}

//opening one more ply for the possible moves
ArrayList<Board> MoveGen(Board b, int player)
{
    ArrayList possible = getPossibleMoves(b, player);
    ArrayList<Board> boardStorage = new ArrayList<Board>();

    if(possible.size() == 0)
    {
        return boardStorage;
    }

    //Board b1;
    for(int i = 0; i < possible.size(); i++)
    {
        Board b1 = new Board(b);
        Board temp = b;
        b1.move((int)(possible.get(i)));
```

```
            b=temp;
            boardStorage.add(b1);
        }

        return boardStorage;
    }
    //performing minimax-a-b algorithm
    MiniMaxABStruct miniMaxABRK(Board b, int depth, int player, int ut, int pt)
    {
        int newValue;
        int val;
        MiniMaxABStruct s = new MiniMaxABStruct();

        if(deepEnough(depth))
        {
            b.player = player;
            //change evaluation function here
            s.value = eval(b, player);
            return s;
        }
        int thisPlayer=player;
        ArrayList<Board> succ = MoveGen(b, player);
        if(succ.size() == 0)
        {
            return s;
        }
```

```
        else
        {
            //numberofnodesgenerated += succ.size();
            NodesGenerated(succ.size(), thisPlayer);
            for(int i = 0; i < succ.size(); i++)
            {
                MiniMaxABStruct res = miniMaxABRK(succ.get(i), depth+1,
b.getOppositePlayer(), -pt, -ut);
                newValue = -(res.value);
                if(newValue > pt)
                {
                    pt = newValue;
                    s.value = pt;
                    s.boardPath.add(b);
                    for(int j = 0; j < res.boardPath.size(); j++)
                    {
                        s.boardPath.add(res.boardPath.get(j));
                    }
                    s.boardPath.add(succ.get(i));
                    for (int j = 0; j < s.boardPath.size(); j++)
                    {
                        for (int k = 0; k < j; k++)
                        {
                            if(s.boardPath.get(j).equal(s.boardPath.get(k)))
                            {
                                s.boardPath.remove(s.boardPath.size()-1);
                                j--;
```

```
                    k--;
                }
            }
        }
    }

    if(pt >= ut)
    {
        return s;
    }
  }

  return s;
 }
}

//------Block end for Minimax-Alpha-Beta Search-----//
//------Block start for Alpha-Beta Search-----//
//checking the depth with currentDepth
boolean deepEnoughAB(int depth, int currentDepth)
{
 if(depth == currentDepth)
 {
  return true;
 }
 else
 {
```

```
            return false;
        }
    }

    int maxValue(Board b, int alpha, int beta)
    {
        if(deepEnoughAB(depth, currentDepth))
        {
            return eval(b, player);
        }

        int v = -1000;
        int thisPlayer=player;
        ArrayList<Board> succ = MoveGen(b, player);
        player=b.getOppositePlayer(); //swap the player
        b.player=player;
        if(succ.isEmpty())
        {
            return -1;
        }
        else
        {
            currentDepth++;
            //numberofnodesgenerated += succ.size();
            NodesGenerated(succ.size(), thisPlayer);
            for(int i=0; i<succ.size(); i++)
            {
```

```
int minvalue = minValue(succ.get(i), alpha, beta);
if(v<minvalue)
{
    v = minvalue;
}

ABData action = new ABData();
action.value = v;
action.boardPath = succ.get(i);
arrayList.add(action);
if(v<=alpha)
{
    System.out.println("tree with alpha < "+v+" is pruned");
    return v;
}
if(beta>v)
{
    beta = v;
}

else
{
    if(v < minValue(succ.get(i), alpha, beta))
    {
        v = minValue(succ.get(i), alpha, beta);
    }
```

```
            if(v > beta)
            {
               return v;
            }

            if(alpha < v)
            {
               alpha = v;
            }
         }
      }

      return v;
   }
}

/*
returning max turn best value,
alpha-is the best already explored value to the root for max
beta-is the best already explored value to the root for min
*/

int minValue(Board b, int alpha, int beta)
{

   if (deepEnoughAB(depth, currentDepth))
```

```
    {

        ABData ac = new ABData();
        //change evaluation function here
        ac.value = eval(b, player);
        ac.boardPath = b;
        arrayList.add(ac);

        return ac.value;
}
int v = 1000;
int thisPlayer=player;
ArrayList<Board> succ = MoveGen(b, player);
player=b.getOppositePlayer(); //swap the player
b.player=player;
if (succ.size() == 0)
{
    return -1;
}
else
{
    currentDepth++;
    //numberofnodesgenerated += succ.size();
    NodesGenerated(succ.size(), thisPlayer);
    for(int i = 0; i < succ.size(); i++)
    {
        if (v > maxValue(succ.get(i), alpha, beta))
```

```
        {
            v = maxValue(succ.get(i), alpha, beta);
        }
        if (v <= alpha)
        {
            return v;
        }
        if (beta > v)
        {
            beta = v;
        }
    }
return v;
}
}
//Function to back up the best value at each node and update the board state
Board getActionBoard(int v)
{
    if(v == -1)
    {
        System.out.println("Integer -1 returned");
    }
    if(arrayList.size() == 0)
    {
        System.out.println("Arraylist is blank");
        return this.board;
    }
```

```
    for(int i = 0; i< arrayList.size()-1 ; i++)
    {
        if(arrayList.get(i).value == v)
        {
            return arrayList.get(i).boardPath;
        }
    }
    ABData max = arrayList.get(0);
    int maxVal = arrayList.get(0).value;
    for(int i = 0; i < arrayList.size(); i++)
    {
        ABData track = arrayList.get(i);
        if(track.value > maxVal)
        {
            max = arrayList.get(i);
            maxVal = track.value;
        }
    }
    //System.out.println("\nSize : " + arrayList.size()+" \nReturned null at the end\n");
    return max.boardPath;
}

//performing alpha beta search
Board alphaBetaSearch(Board b)
{
    //player = b.player;
    //oppositePlayer = b.getOppositePlayer();
```

```
    int value = maxValue(b, -1000, 1000);
    Board b1 = getActionBoard(value);
    currentDepth = 0;
    arrayList.clear();
    return b1;
}
//------Block end for Alpha-Beta Search-----//
//--------------------Block end for AI algorithm------------------------------



//--------------------Block Start For Evaluation function----------------------
//returning the evaluation value
int evaluationFunction1(Board b, int player)
{
    int player1stones=0;
    for(int i=1;i<7;i++)
    {
        player1stones+=b.getBoardValueByPosition(i);
    }

    int player2stones=0;
    for(int i=8;i<13;i++)
    {
        player2stones+=b.getBoardValueByPosition(i);
    }

    if(player == 0)
```

```
        {

            return ((int)b.getBoardValueByPosition(0)+player1stones) -
((int)b.getBoardValueByPosition(7)+player2stones)*2+((int)b.getBoardValueByPosition(
0)+player1stones) - ((int)b.getBoardValueByPosition(7)+player2stones);
        }
        else
        {
            return ((int)b.getBoardValueByPosition(7)+player2stones) -
((int)b.getBoardValueByPosition(0)+player1stones)*2+((int)b.getBoardValueByPosition(
7)+player2stones) - ((int)b.getBoardValueByPosition(0)+player1stones);
        }

    }

    //returning the evaluation value
    int evaluationFunction2(Board b, int player)
    {
        int player0Kalah = b.getBoardValueByPosition(0);
        int player1Kalah = b.getBoardValueByPosition(7);
        if(player == 0)
        {
            if(player1Kalah != 0)
            {
                return (b.getBoardValueByPosition(0)-b.getBoardValueByPosition(7));
            }
            else
```

```
        {
            return -1;
        }
    }

    else
    {
        if(player0Kalah != 0)
        {
            return (b.getBoardValueByPosition(7)-b.getBoardValueByPosition(0));
        }
        else
        {
            return -1;
        }
    }
}
//returning the evaluation value
int evaluationFunction3(Board b, int player)
{
    int player1stones=0;
    for(int i=1;i<7;i++)
    {
        player1stones+=b.getBoardValueByPosition(i);
    }

    int player2stones=0;
```

```
    for(int i=8;i<13;i++)
    {
        player2stones+=b.getBoardValueByPosition(i);
    }

    if(player == 0)
    {

        return (int)b.getBoardValueByPosition(0)+player2stones;
    }
    else
    {
        return (int)b.getBoardValueByPosition(7)+player1stones;
    }

}

int eval(Board b, int player)
{
    int ret = 0;
    switch(EvalChoice)
    {
    case 1:
        ret= evaluationFunction1(b, player);
        break;
    case 2:
        ret= evaluationFunction2(b, player);
```

```
            break;
        case 3:
            ret= evaluationFunction3(b, player);
            break;
        }

        return ret;

    }
    //--------------------Block End for Evaluation function-----------------------


    //----------------------Block start for basic functionality--------------------

    void NodesGenerated(int nodes, int TempPlayer)
    {
        if(TempPlayer==0)
        {
            numberofnodesgenerated1+=nodes;
        }
        else
        {
            numberofnodesgenerated2+=nodes;
        }
    }
    //displaying board
    void displayBoard(GameGUI frame)
```

```
{
    board.displayBoard(frame);
}
//checking board is empty or not
boolean isBoardEmpty()
{
    return board.isBoardEmpty();
}
//getting current board
Board getBoard()
{
    return board;
}
//setting current board with given board and player
void setBoard(Board b, int player)
{
    board = b;
    board.player = player;
}
//returning the current player
int getPlayer()
{
    return board.getPlayer();
}
//returning the opposite player
int getOppositePlayer()
{
```

```
    return board.getOppositePlayer();
}
//returning the opposite player for the given player
int getOppositePlayerFromBoard(int player)
{
    return 1-player;
}
//setting next player of the given player
void setNextPlayer(int player)
{
    board.player = 1-player;
}
//returning true if the player turns into the same kalah
boolean itsInSameKalah()
{
    return board.itsInSameKalah();
}
//returning player1 kalah value
int getPlayer1Kalah()
{
    return board.getBoardValueByPosition(0);
}
//returning player2 kalah value
int getPlayer2Kalah()
{
    return board.getBoardValueByPosition(7);
}
```

```
    //----------------------Block end for basic functionality----------------------

}
```

GamePlay.java

```
package txstate.cs.ai.kala;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.Scanner;

public class GamePlay {

    int depth;
    int player;
    int EvalChoice1;
    int EvalChoice2;

    ArrayList<Board> path = new ArrayList<Board>();
    MiniMaxABStruct s = new MiniMaxABStruct();

    long totalTime = 0;
    int count = 0;
    Board board;
    StringBuilder str = new StringBuilder();
    public GameGUI newFrame = new GameGUI();
```

```
    public static void main(String[] args) {

        StaticGUI gui1 = new StaticGUI();
        OptionsGUI optionsFrame = new OptionsGUI();
        gui1.frmKalahGame.setVisible(true);



        gui1.mainbuttonlist.get("alphabetasearch").addActionListener(new ActionListener()
{
            public void actionPerformed(ActionEvent e) {
                optionsFrame.setVisible(true);
                optionsFrame.start.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                        GamePlay obj = new GamePlay();
                        obj.SetOptions(Integer.parseInt(optionsFrame.txtPlayer.getText()),
Integer.parseInt(optionsFrame.txtDepth.getText()),
Integer.parseInt(optionsFrame.txtEval1.getText()),
Integer.parseInt(optionsFrame.txtEval2.getText()));
                        obj.AlphaBeta();
                    }
                });
            }
        });

        gui1.mainbuttonlist.get("minmaxab").addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
```

```
            optionsFrame.setVisible(true);
            optionsFrame.start.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    GamePlay obj = new GamePlay();
                    obj.SetOptions(Integer.parseInt(optionsFrame.txtPlayer.getText()),
Integer.parseInt(optionsFrame.txtDepth.getText()),
Integer.parseInt(optionsFrame.txtEval1.getText()),
Integer.parseInt(optionsFrame.txtEval2.getText()));
                    obj.MinMaxAB();
                }
            });
        }
    });

    gui1.mainbuttonlist.get("minalpha").addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            optionsFrame.setVisible(true);
            optionsFrame.start.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    GamePlay obj = new GamePlay();
                    obj.SetOptions(Integer.parseInt(optionsFrame.txtPlayer.getText()),
Integer.parseInt(optionsFrame.txtDepth.getText()),
Integer.parseInt(optionsFrame.txtEval1.getText()),
Integer.parseInt(optionsFrame.txtEval2.getText()));
                    obj.MinMaxABAlphaBeta();
                }
            });
```

```
        }
    });

    gui1.mainbuttonlist.get("alphamin").addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            optionsFrame.setVisible(true);
            optionsFrame.start.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    GamePlay obj = new GamePlay();
                    obj.SetOptions(Integer.parseInt(optionsFrame.txtPlayer.getText()),
Integer.parseInt(optionsFrame.txtDepth.getText()),
Integer.parseInt(optionsFrame.txtEval1.getText()),
Integer.parseInt(optionsFrame.txtEval2.getText()));
                    obj.AlphaBetaMinMaxAB();
                }
            });
        }
    });

    gui1.mainbuttonlist.get("uservsuser").addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            optionsFrame.setVisible(true);
            optionsFrame.start.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    new
GamePlay().SetOptions(Integer.parseInt(optionsFrame.txtPlayer.getText()),
Integer.parseInt(optionsFrame.txtDepth.getText()),
```

```
Integer.parseInt(optionsFrame.txtEval1.getText()),
Integer.parseInt(optionsFrame.txtEval2.getText()));
                    new GamePlay().UservsUser();
                }
            });
        }
    });



}

public void SetOptions(int tempPlayer, int tempDepth, int tempEval1, int tempEval2)
{
    depth = tempDepth;
    player = tempPlayer;
    EvalChoice1 = tempEval1;
    EvalChoice2 = tempEval2;



}

public void SetEvalForPlayer(GameEngine tempGameEngine)
{
    tempGameEngine.EvalChoice=((tempGameEngine.board.player) ==
0)?EvalChoice1:EvalChoice2;
}
```

```
public void MinMaxAB()
{
    newFrame.gametype.setText("MINMAX-AB vs MINMAX-AB");

    GameEngine k = new GameEngine(depth, player);
    System.out.println("\n\nInitial State: ");
    newFrame.txtArea.append("Initial State: ");
    k.displayBoard(newFrame);
    path.add(k.getBoard());
    //k.EvalChoice=EvalChoice;

    int moved=0;
    int count1=1;
    int count2=1;
    boolean playerchanged=false;
    int val=0;

    long start_s=System.currentTimeMillis();
    while(!k.isGameOver())
    {

        SetEvalForPlayer(k);

        s = k.miniMaxABRK(k.getBoard(), 0, k.getPlayer(), 100, -100);
        if(s.boardPath.size() > 0)
        {
            moved=s.boardPath.get(1).expanded;
```

```
            val=s.value;
            k.setBoard(s.boardPath.get(1), k.getPlayer());
            path.add(k.getBoard());
            playerchanged=s.boardPath.get(1).isPlayerChanged;



        }

        System.out.println("Player P"+k.getPlayer()+" expanded Node #"+moved+" with
Evl: "+val);
        newFrame.txtArea.append("Player P"+k.getPlayer()+" expanded Node
#"+moved+" with Evl: "+val);
        System.out.println("New Board position is: ");
        newFrame.txtArea.append("\nNew Board position is: ");
        k.displayBoard(newFrame);

        count1=(k.getPlayer()==0)?count1+1:count1;
        count2=(k.getPlayer()==1)?count2+1:count2;

        if(playerchanged)
        {
            k.setNextPlayer(k.getPlayer());
        }
        else
        {
            System.out.println("Free move!");
            newFrame.txtArea.append("\nFree move!");
```

```
            }


        }
        if(k.getPlayer1Kalah() == k.getPlayer2Kalah())
        {
            System.out.println("Game is drawn");
            newFrame.txtArea.append("Game is drawn\n");

        }
        else if(k.getPlayer1Kalah() > k.getPlayer2Kalah())
        {
            System.out.println("\nPlayer P"+k.getPlayer()+" won the game!");
            newFrame.txtArea.append("\nPlayer P"+k.getPlayer()+" won the game!\n");
        }
        else
        {
            System.out.println("\nPlayer P"+k.getPlayer()+" won the game!");
            newFrame.txtArea.append("\nPlayer P"+k.getPlayer()+" won the game!\n");

        }

        long stop_s=System.currentTimeMillis();
        totalTime += (stop_s-start_s);

        System.out.println("Number of nodes generated by Player#1 :
"+k.numberofnodesgenerated1);
```

```
        newFrame.txtArea.append("Number of nodes generated by Player#1 :
"+k.numberofnodesgenerated1);
        System.out.println("\nNumber of nodes generated by Player#2 :
"+k.numberofnodesgenerated2);
        newFrame.txtArea.append("\nNumber of nodes generated by Player#2 :
"+k.numberofnodesgenerated2);
        System.out.println("\nNumber of nodes expanded by Player#1 : "+count1);
        newFrame.txtArea.append("\nNumber of nodes expanded by Player#1 : "+count1);
        System.out.println("\nNumber of nodes expanded by Player#2 : "+count2);
        newFrame.txtArea.append("\nNumber of nodes expanded by Player#2 : "+count2);
        System.out.println("\nExecution time : "+(totalTime)+" milliseconds");
        newFrame.txtArea.append("\nExecution time : "+(totalTime)+" milliseconds");
        System.out.println("Total memory occupied :
"+(k.numberofnodesgenerated1+k.numberofnodesgenerated2) * 4+" bytes");
        newFrame.txtArea.append("\nTotal memory occupied :
"+(k.numberofnodesgenerated1+k.numberofnodesgenerated2) * 4+" bytes");

        newFrame.setVisible(true);
    }

    public void AlphaBeta()
    {
        newFrame.gametype.setText("ALPHABETA SEARCH vs ALPHABETA
SEARCH");

        GameEngine k = new GameEngine(depth, player);
        int moved=0;
```

```
        int count1=1;
        int count2=1;
        boolean playerchanged=false;

        //k.EvalChoice=EvalChoice;

        long start_s=System.currentTimeMillis();

        System.out.println("\nInitial state: \n"+depth+player);
        newFrame.txtArea.append("\nInitial State: \n");
        k.displayBoard(newFrame);

        while(!k.isGameOver())
        {

            SetEvalForPlayer(k);

            Board board = k.alphaBetaSearch(k.getBoard());

            k.setBoard(board, board.getPlayer());
            moved=board.expanded;
            playerchanged=board.isPlayerChanged;

            System.out.println("Player P"+k.board.getPlayer()+" expanded Node #"+moved);
            newFrame.txtArea.append("Player P"+k.board.getPlayer()+" expanded Node
#"+moved);
            System.out.println("New Board position is: ");
```

```
        newFrame.txtArea.append("\nNew Board position is: ");
        k.displayBoard(newFrame);

        count1=(k.getPlayer()==0)?count1+1:count1;
        count2=(k.getPlayer()==1)?count2+1:count2;

        if(playerchanged)
        {
            k.setNextPlayer(k.getPlayer());
        }
        else
        {
            System.out.println("Free move!");
            newFrame.txtArea.append("\nFree move!");
        }


    }

    if(k.getPlayer1Kalah() == k.getPlayer2Kalah())
    {
        System.out.println("Game is drawn");
        newFrame.txtArea.append("Game is drawn\n");

    }
    else if(k.getPlayer1Kalah() > k.getPlayer2Kalah())
    {
```

```
            System.out.println("\nPlayer P"+k.getPlayer()+" won the game!");
            newFrame.txtArea.append("\nPlayer P"+k.getPlayer()+" won the game!\n");
        }
        else
        {
            System.out.println("\nPlayer P"+k.getPlayer()+" won the game!");
            newFrame.txtArea.append("\nPlayer P"+k.getPlayer()+" won the game!\n");

        }

        long stop_s=System.currentTimeMillis();
        totalTime += (stop_s-start_s);

        System.out.println("Number of nodes generated by Player#1 :
"+k.numberofnodesgenerated1);
        newFrame.txtArea.append("Number of nodes generated by Player#1 :
"+k.numberofnodesgenerated1);
        System.out.println("\nNumber of nodes generated by Player#2 :
"+k.numberofnodesgenerated2);
        newFrame.txtArea.append("\nNumber of nodes generated by Player#2 :
"+k.numberofnodesgenerated2);
        System.out.println("\nNumber of nodes expanded by Player#1 : "+count1);
        newFrame.txtArea.append("\nNumber of nodes expanded by Player#1 : "+count1);
        System.out.println("\nNumber of nodes expanded by Player#2 : "+count2);
        newFrame.txtArea.append("\nNumber of nodes expanded by Player#2 : "+count2);
        System.out.println("\nExecution time : "+(totalTime)+" milliseconds");
        newFrame.txtArea.append("\nExecution time : "+(totalTime)+" milliseconds");
```

```
        System.out.println("Total memory occupied :
"+(k.numberofnodesgenerated1+k.numberofnodesgenerated2) * 4+" bytes");
        newFrame.txtArea.append("\nTotal memory occupied :
"+(k.numberofnodesgenerated1+k.numberofnodesgenerated2) * 4+" bytes");

        newFrame.setVisible(true);

    }


    public void MinMaxABAlphaBeta()
    {
        newFrame.gametype.setText("MINMAX-AB vs Alpha-Beta Search");

        GameEngine k = new GameEngine(depth, player);
        System.out.println("\n\nInitial State: ");
        newFrame.txtArea.append("Initial State: ");
        k.displayBoard(newFrame);
        path.add(k.getBoard());
        //k.EvalChoice=EvalChoice;

        int moved=0;
        int count1=1;
        int count2=1;
        boolean playerchanged=false;
        int val=0;
```

```
long start_s=System.currentTimeMillis();
while(!k.isGameOver())
{
   SetEvalForPlayer(k);

   s = k.miniMaxABRK(k.getBoard(), 0, k.getPlayer(), 100, -100);
   if(s.boardPath.size() > 0)
   {
      moved=s.boardPath.get(1).expanded;
      val=s.value;
      k.setBoard(s.boardPath.get(1), k.getPlayer());
      path.add(k.getBoard());
      playerchanged=s.boardPath.get(1).isPlayerChanged;



   }

      System.out.println("Player P"+k.getPlayer()+" expanded Node #"+moved+" with
Evl: "+val);
      newFrame.txtArea.append("Player P"+k.getPlayer()+" expanded Node
#"+moved+" with Evl: "+val);
      System.out.println("New Board position is: ");
      newFrame.txtArea.append("\nNew Board position is: ");
      k.displayBoard(newFrame);

      count1=(k.getPlayer()==0)?count1+1:count1;
      count2=(k.getPlayer()==1)?count2+1:count2;
```

```
        if(playerchanged)
        {
            k.setNextPlayer(k.getPlayer());

            Board board = k.alphaBetaSearch(k.getBoard());

            k.setBoard(board, board.getPlayer());
            moved=board.expanded;
            playerchanged=board.isPlayerChanged;

            System.out.println("Player P"+k.board.getPlayer()+" expanded Node
#"+moved);
            newFrame.txtArea.append("Player P"+k.board.getPlayer()+" expanded Node
#"+moved);
            System.out.println("New Board position is: ");
            newFrame.txtArea.append("\nNew Board position is: ");
            k.displayBoard(newFrame);

            count1=(k.getPlayer()==0)?count1+1:count1;
            count2=(k.getPlayer()==1)?count2+1:count2;

        }
        else
        {
            System.out.println("Free move!");
            newFrame.txtArea.append("\nFree move!");
```

```
            }


        }
        if(k.getPlayer1Kalah() == k.getPlayer2Kalah())
        {
            System.out.println("Game is drawn");
            newFrame.txtArea.append("Game is drawn\n");

        }
        else if(k.getPlayer1Kalah() > k.getPlayer2Kalah())
        {
            System.out.println("\nPlayer P"+k.getPlayer()+" won the game!");
            newFrame.txtArea.append("\nPlayer P"+k.getPlayer()+" won the game!\n");
        }
        else
        {
            System.out.println("\nPlayer P"+k.getPlayer()+" won the game!");
            newFrame.txtArea.append("\nPlayer P"+k.getPlayer()+" won the game!\n");

        }

        long stop_s=System.currentTimeMillis();
        totalTime += (stop_s-start_s);

        System.out.println("Number of nodes generated by Player#1 :
"+k.numberofnodesgenerated1);
```

```
        newFrame.txtArea.append("Number of nodes generated by Player#1 :
"+k.numberofnodesgenerated1);
        System.out.println("\nNumber of nodes generated by Player#2 :
"+k.numberofnodesgenerated2);
        newFrame.txtArea.append("\nNumber of nodes generated by Player#2 :
"+k.numberofnodesgenerated2);
        System.out.println("\nNumber of nodes expanded by Player#1 : "+count1);
        newFrame.txtArea.append("\nNumber of nodes expanded by Player#1 : "+count1);
        System.out.println("\nNumber of nodes expanded by Player#2 : "+count2);
        newFrame.txtArea.append("\nNumber of nodes expanded by Player#2 : "+count2);
        System.out.println("\nExecution time : "+(totalTime)+" milliseconds");
        newFrame.txtArea.append("\nExecution time : "+(totalTime)+" milliseconds");
        System.out.println("Total memory occupied :
"+(k.numberofnodesgenerated1+k.numberofnodesgenerated2) * 4+" bytes");
        newFrame.txtArea.append("\nTotal memory occupied :
"+(k.numberofnodesgenerated1+k.numberofnodesgenerated2) * 4+" bytes");

        newFrame.setVisible(true);
    }



    public void AlphaBetaMinMaxAB()
    {
        newFrame.gametype.setText("ALPHABETA SEARCH vs ALPHABETA
SEARCH");
```

```
GameEngine k = new GameEngine(depth, player);
int moved=0;
int count1=1;
int count2=1;
int val=0;
boolean playerchanged=false;

//k.EvalChoice=EvalChoice;

long start_s=System.currentTimeMillis();

System.out.println("\nInitial state: \n"+depth+player);
newFrame.txtArea.append("\nInitial State: \n");
k.displayBoard(newFrame);

while(!k.isGameOver())
{


   SetEvalForPlayer(k);

   Board board = k.alphaBetaSearch(k.getBoard());

   k.setBoard(board, board.getPlayer());
   moved=board.expanded;
   playerchanged=board.isPlayerChanged;
```

```
        System.out.println("Player P"+k.board.getPlayer()+" expanded Node #"+moved);
        newFrame.txtArea.append("Player P"+k.board.getPlayer()+" expanded Node
#"+moved);
        System.out.println("New Board position is: ");
        newFrame.txtArea.append("\nNew Board position is: ");
        k.displayBoard(newFrame);

        count1=(k.getPlayer()==0)?count1+1:count1;
        count2=(k.getPlayer()==1)?count2+1:count2;

        if(playerchanged)
        {
           k.setNextPlayer(k.getPlayer());
           s = k.miniMaxABRK(k.getBoard(), 0, k.getPlayer(), 100, -100);
           if(s.boardPath.size() > 0)
           {
              moved=s.boardPath.get(1).expanded;
              val=s.value;
              k.setBoard(s.boardPath.get(1), k.getPlayer());
              path.add(k.getBoard());
              playerchanged=s.boardPath.get(1).isPlayerChanged;


           }

           System.out.println("Player P"+k.getPlayer()+" expanded Node #"+moved+"
with Evl: "+val);
```

```
            newFrame.txtArea.append("Player P"+k.getPlayer()+" expanded Node
#"+moved+" with Evl: "+val);
            System.out.println("New Board position is: ");
            newFrame.txtArea.append("\nNew Board position is: ");
            k.displayBoard(newFrame);

            count1=(k.getPlayer()==0)?count1+1:count1;
            count2=(k.getPlayer()==1)?count2+1:count2;

        }
        else
        {
            System.out.println("Free move!");
            newFrame.txtArea.append("\nFree move!");
        }


    }

    if(k.getPlayer1Kalah() == k.getPlayer2Kalah())
    {
        System.out.println("Game is drawn");
        newFrame.txtArea.append("Game is drawn\n");

    }
    else if(k.getPlayer1Kalah() > k.getPlayer2Kalah())
    {
```

```
            System.out.println("\nPlayer P"+k.getPlayer()+" won the game!");
            newFrame.txtArea.append("\nPlayer P"+k.getPlayer()+" won the game!\n");
        }
        else
        {
            System.out.println("\nPlayer P"+k.getPlayer()+" won the game!");
            newFrame.txtArea.append("\nPlayer P"+k.getPlayer()+" won the game!\n");

        }

        long stop_s=System.currentTimeMillis();
        totalTime += (stop_s-start_s);

        System.out.println("Number of nodes generated by Player#1 :
"+k.numberofnodesgenerated1);
        newFrame.txtArea.append("Number of nodes generated by Player#1 :
"+k.numberofnodesgenerated1);
        System.out.println("\nNumber of nodes generated by Player#2 :
"+k.numberofnodesgenerated2);
        newFrame.txtArea.append("\nNumber of nodes generated by Player#2 :
"+k.numberofnodesgenerated2);
        System.out.println("\nNumber of nodes expanded by Player#1 : "+count1);
        newFrame.txtArea.append("\nNumber of nodes expanded by Player#1 : "+count1);
        System.out.println("\nNumber of nodes expanded by Player#2 : "+count2);
        newFrame.txtArea.append("\nNumber of nodes expanded by Player#2 : "+count2);
        System.out.println("\nExecution time : "+(totalTime)+" milliseconds");
        newFrame.txtArea.append("\nExecution time : "+(totalTime)+" milliseconds");
```

```
        System.out.println("Total memory occupied :
"+(k.numberofnodesgenerated1+k.numberofnodesgenerated2) * 4+" bytes");
        newFrame.txtArea.append("\nTotal memory occupied :
"+(k.numberofnodesgenerated1+k.numberofnodesgenerated2) * 4+" bytes");

        newFrame.setVisible(true);

    }

    public void UservsUser()
    {
        newFrame.gametype.setText("Random Bot vs Random Bot");

        newFrame.setVisible(true);

        GameEngine k = new GameEngine();
        k.init();
        System.out.println("\nInitial state: \n");
        newFrame.txtArea.append("\nInitial State: \n");
        k.displayBoard(newFrame);
        k.playUserUser(newFrame);

    }

}
```

Board.java

```
package txstate.cs.ai.kala;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Vector;
import java.util.concurrent.TimeUnit;

public class Board {

    int player;
    int oppositePlayer;
    int stoneThreshold;
    boolean isPlayerChanged;
    int numberOfStonesInplayerOneKalah;
    int numberOfStonesInplayerTwoKalah;
    int[] board;
    int numPits;
    int trackPosition;
    int expanded;
    String sameKalah;

    Board()
    {
        board=new int[14];
        board[0]=0;
```

```
    for(int i = 1; i < 7; i++)
    {
        board[i]=6;
    }

    board[7]=0;
    for(int i = 8; i < 14; i++)
    {
        board[i]=6;
    }
    numberOfStonesInplayerOneKalah = 0;
    numberOfStonesInplayerTwoKalah = 0;
    numPits = 14;
    player = 1;
    sameKalah = "no";
    stoneThreshold = 36;
    isPlayerChanged = false;
}

Board(Board MainBoard)
{
    int[] temparr = new int[14];
    for(int i=0;i<temparr.length;i++)
    {
        temparr[i]=MainBoard.getBoard()[i];
    }
    board=temparr;
```

```
        isPlayerChanged=MainBoard.isPlayerChanged;

numberOfStonesInplayerOneKalah=MainBoard.numberOfStonesInplayerOneKalah;

numberOfStonesInplayerTwoKalah=MainBoard.numberOfStonesInplayerTwoKalah;
        numPits=MainBoard.numPits;
        oppositePlayer=MainBoard.oppositePlayer;
        player=MainBoard.player;
        sameKalah=MainBoard.sameKalah;
        stoneThreshold=MainBoard.stoneThreshold;
        trackPosition=MainBoard.trackPosition;
    }

    //constructor
    Board(int[] vec)
    {
        board = vec;
    }
    //displaying board
    void displayBoard(GameGUI frame)
    {

        String msg="";
        System.out.println("\n\n");
        msg+="\n";
        for(int i = 0; i < 38; i++)
        {
```

```
    System.out.print("-");
    msg+="-";
}

System.out.println();
msg+="\n";
System.out.println("P#0");
System.out.print("| ");
msg+="P#0";
msg+="\n| ";

for(int i = 1; i < 7; i++)
{
    System.out.print(board[i]+" ");
    msg+=board[i]+" ";
    frame.buttonlist[i].setText(board[i]+"");
}
System.out.println("|");
msg+="|\n";
System.out.println("|"+board[0]+"          "+board[7]+ "|");
msg+="|"+board[0]+"          "+board[7]+ "|";
frame.buttonlist[0].setText(board[0]+"");
frame.buttonlist[7].setText(board[7]+"");
msg+="\n";
System.out.print("| ");
msg+="| ";
```

```
    for(int i = 13; i >= 8; i--)
    {
        System.out.print(board[i]+" ");
        msg+=board[i]+" ";
        frame.buttonlist[i].setText(board[i]+"");
    }
    System.out.print("|");
    msg+="|";
    System.out.println();
    msg+="\n";
    System.out.println("P#1");
    msg+="P#1\n";


    for(int i = 0; i < 38; i++)
    {
        System.out.print("-");
        msg+="-";
    }
    System.out.println();
    msg+="\n";
    System.out.println("\n\n");

    frame.txtArea.append(msg);

}
```

```
//return true if board is empty
boolean isBoardEmpty()
{
    boolean result = true;
    for(int i = 0; i < 14; i++)
    {
        if(i != 0 && i != 7)
        {
            if((int)board[i] > 0)
            {
                result = false;
            }
        }
    }
    return result;
}

//returning board vector
int[] getBoard()
{
    return board;
}

//setting board array for the given board
void setBoard(int[] b)
{
    board = b;
```

```
}

//returning current player
int getPlayer()
{
   return player;
}

//returning opposite player
int getOppositePlayer()
{
   return 1-player;
}

//returning board value for the given position
int getBoardValueByPosition(int pos)
{
   return board[pos];
}

//set board value for the given position and value
void setBoardValueByPosition(int pos,int val)
{
   board[pos]=val;
}

//returning kalah position for the given position index
```

```
int getPlayerKalahPositionByIndex(int pos)
{
    int ret=0;

    if(pos >= 1 && pos <= 6)
    {
        ret = 0;
    }
    else if(pos >= 8 && pos <= 13)
    {
        ret = 7;
    }

    return ret;
}

//returning opponent kalah position of the given position
int getOpponentKalahPositionByIndex(int pos)
{
    int ret=0;

    if(pos >= 1 && pos <= 6)
    {
        ret = 7;
    }
    else if(pos >= 8 && pos <= 13)
    {
```

```
            ret = 0;
        }

        return ret;
    }

    //returning player kalah value of the given position
    int getPlayerKalahValueByIndex(int pos)
    {
        int playerKalahIndex = getPlayerKalahPositionByIndex(pos);
        return board[playerKalahIndex];
    }

    //returning opponent kalah value of the given position
    int getOpponentKalahValueByIndex(int pos)
    {
        int opponentKalahIndex = getOpponentKalahPositionByIndex(pos);
        return board[opponentKalahIndex];
    }

    //returning number of stones of player one kalah
    int getNumberOfStonesInPlayerOneKalah()
    {
        return numberOfStonesInplayerOneKalah;
    }

    //returning number of stones of player two kalah
```

```
int getNumberOfStonesInPlayerTwoKalah()
{
    return numberOfStonesInplayerTwoKalah;
}

//returning true if the player turn goes into player kalah
boolean itsInSameKalah()
{
    if(sameKalah == "yes")
    {
        return true;
    }
    else
    {
        return false;
    }
}

//getting notification for player turn kalah position
void getSameKalahOrNot(int trackPosition, int playerKalah)
{
    if(trackPosition == playerKalah)
    {
        sameKalah = "yes";
    }
    else
    {
```

```
            sameKalah = "no";
        }
    }

    //checking kalah game is over or not
    boolean isGameOver()
    {
        if(numberOfStonesInplayerOneKalah > stoneThreshold ||
numberOfStonesInplayerTwoKalah > stoneThreshold)
        {
            return true;
        }

        else if(numberOfStonesInplayerOneKalah == stoneThreshold &&
numberOfStonesInplayerTwoKalah == stoneThreshold)
        {
            return true;
        }

        else
        {
            return false;
        }
    }

    //returning true if the current board is equal to the given board
    boolean equal(Board b1)
```

```
{
    int[] b1Vec = b1.getBoard();
    boolean result = true;
    for(int i = 0; i < 14; i++)
    {
        if(board[i] != b1Vec[i])
        {
            result = false;
        }
    }
    return result;
}

//moving stones from the given position
void move(int position)
{
    int playerKalah,opponentKalah, pitInput, posIndex;
    trackPosition = position;
    expanded=position;
    int grabOppositePosition, numOfOppPositionStones;
    int numberOfStones = getBoardValueByPosition(position);
    setBoardValueByPosition(position, 0);
    playerKalah =getPlayerKalahPositionByIndex(position);
    opponentKalah = getOpponentKalahPositionByIndex(position);

    int tempPosition=0;
    if(position >= 1 && position <= 6)
```

```
        {
            tempPosition = 1;
        }

        else if(position >= 8 && position <= 13)
        {
            tempPosition = 8;
        }

        if(numberOfStones == 1)
        {
            trackPosition--;
            if(trackPosition < 0)
            {
                trackPosition = numPits-1;
            }
            if(trackPosition != opponentKalah)
            {
                numberOfStones--;
                if((getBoardValueByPosition(trackPosition) == 0 && trackPosition <= 6 &&
trackPosition >= 1) && tempPosition == 1 && numberOfStones == 0)
                {
                    if(getBoardValueByPosition(trackPosition) == 0 && numberOfStones == 0)
                    {
                        grabOppositePosition = numPits-trackPosition;
                        numOfOppPositionStones =
getBoardValueByPosition(grabOppositePosition);
```

```
                setBoardValueByPosition(grabOppositePosition, 0);
                setBoardValueByPosition(playerKalah,
getBoardValueByPosition(playerKalah) + numOfOppPositionStones + 1);
            }
        }

        else if(getBoardValueByPosition(trackPosition) == 0 && (trackPosition <= 13
&& trackPosition >= 8) && tempPosition == 8 && numberOfStones == 0)
        {
            if(getBoardValueByPosition(trackPosition) == 0 && numberOfStones == 0)
            {
                grabOppositePosition = numPits-trackPosition;
                numOfOppPositionStones =
getBoardValueByPosition(grabOppositePosition);
                setBoardValueByPosition(grabOppositePosition, 0);
                setBoardValueByPosition(playerKalah,
getBoardValueByPosition(playerKalah) +    numOfOppPositionStones + 1);
            }
        }

        else
        {
            setBoardValueByPosition(trackPosition,
getBoardValueByPosition(trackPosition) +    1);
        }
    }
}
```

```
        else
        {
            while(numberOfStones > 0)
            {
                trackPosition--;
                if(trackPosition < 0)
                {
                    trackPosition = numPits-1;
                }
                if(trackPosition != opponentKalah)
                {
                    numberOfStones--;
                    setBoardValueByPosition(trackPosition,
(getBoardValueByPosition(trackPosition) + 1));
                }
            }
        }

        numberOfStonesInplayerOneKalah = getBoardValueByPosition(playerKalah);
        numberOfStonesInplayerTwoKalah = getBoardValueByPosition(opponentKalah);
        getSameKalahOrNot(trackPosition, playerKalah);

        if(trackPosition != playerKalah && !isGameOver())
        {
            player = 1-player;
            isPlayerChanged = true;
```

```
        }
        else
        {
            isPlayerChanged = false;
        }
    }


}
```

GameUI.java

```
package txstate.cs.ai.kala;

import java.awt.BorderLayout;
import java.awt.EventQueue;
import java.util.ArrayList;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JButton;
import javax.swing.JTextPane;
import javax.swing.JLabel;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;
import javax.swing.ScrollPaneConstants;
```

```
public class GameGUI extends JFrame {

    private JPanel contentPane;
    JButton[] buttonlist = new JButton[14];
    JTextArea txtArea;
    JLabel gametype;

    /**
     * Create the frame.
     */
    public GameGUI() {



        setTitle("Welcome to Kalah Game!");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 631, 511);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JButton zero = new JButton("");
        zero.setBounds(63, 123, 61, 34);
        contentPane.add(zero);
        buttonlist[0]=zero;
```

```
JButton two = new JButton("");
two.setBounds(197, 81, 44, 23);
contentPane.add(two);
buttonlist[2]=two;

JButton one = new JButton("");
one.setBounds(139, 81, 44, 23);
contentPane.add(one);
buttonlist[1]=one;

JButton four = new JButton("");
four.setBounds(311, 81, 44, 23);
contentPane.add(four);
buttonlist[4]=four;

JButton seven = new JButton("");
seven.setBounds(475, 123, 61, 34);
contentPane.add(seven);
buttonlist[7]=seven;

JButton thirteen = new JButton("");
thirteen.setBounds(139, 168, 44, 23);
contentPane.add(thirteen);
buttonlist[13]=thirteen;

JButton eight = new JButton("");
```

```
eight.setBounds(418, 168, 44, 23);
contentPane.add(eight);
buttonlist[8]=eight;

JButton ten = new JButton("");
ten.setBounds(311, 168, 44, 23);
contentPane.add(ten);
buttonlist[10]=ten;

JButton eleven = new JButton("");
eleven.setBounds(257, 168, 44, 23);
contentPane.add(eleven);
buttonlist[11]=eleven;

JButton twelve = new JButton("");
twelve.setBounds(197, 168, 44, 23);
contentPane.add(twelve);
buttonlist[12]=twelve;

JButton three = new JButton("");
three.setBounds(257, 81, 44, 23);
contentPane.add(three);
buttonlist[3]=three;

JButton five = new JButton("");
five.setBounds(365, 81, 44, 23);
contentPane.add(five);
```

```
        buttonlist[5]=five;

        JButton six = new JButton("");
        six.setBounds(419, 81, 44, 23);
        contentPane.add(six);
        buttonlist[6]=six;

        JButton nine = new JButton("");
        nine.setBounds(365, 168, 44, 23);
        contentPane.add(nine);
        buttonlist[9]=nine;

        JLabel lblNewLabel = new JLabel("Program trace:");
        lblNewLabel.setBounds(10, 246, 87, 14);
        contentPane.add(lblNewLabel);

        JTextArea textArea = new JTextArea();
        textArea.setBounds(-100, 271, 595, 190);
        //contentPane.add(textArea);
        txtArea = textArea;

        JScrollPane scrollPane = new JScrollPane(textArea);

scrollPane.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLL
BAR_ALWAYS);
```

```
scrollPane.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_A
LWAYS);
        scrollPane.setBounds(10, 271, 595, 190);
        contentPane.add(scrollPane);

        JLabel gametypelabel = new JLabel("");
        gametypelabel.setBounds(197, 11, 212, 14);
        contentPane.add(gametypelabel);
        gametype = gametypelabel;
    }

    public JButton[] getButtonList()
    {
        return buttonlist;
    }
}
```

OptionsGUI.java

```
package txstate.cs.ai.kala;

import java.awt.BorderLayout;
import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JPanel;
```

```
import javax.swing.border.EmptyBorder;
import javax.swing.JLabel;
import java.awt.Font;
import java.awt.Color;
import javax.swing.JTextField;
import javax.swing.JButton;

public class OptionsGUI extends JFrame {

    private JPanel contentPane;
    public JTextField txtPlayer;
    public JTextField txtDepth;
    public JTextField txtEval1;
    public JButton start;
    public JTextField txtEval2;



    /**
     * Create the frame.
     */
    public OptionsGUI() {
        setTitle("Kalah Game");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 562, 359);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
```

```
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JLabel lblNewLabel = new JLabel("Welcome to Kalah Game!");
        lblNewLabel.setBounds(128, 11, 424, 17);
        lblNewLabel.setForeground(Color.BLUE);
        lblNewLabel.setFont(new Font("Tahoma", Font.BOLD, 14));
        contentPane.add(lblNewLabel);

        JLabel lblNewLabel_1 = new JLabel("Choose Game config values");
        lblNewLabel_1.setFont(new Font("Tahoma", Font.BOLD | Font.ITALIC, 11));
        lblNewLabel_1.setForeground(Color.MAGENTA);
        lblNewLabel_1.setBounds(138, 39, 166, 14);
        contentPane.add(lblNewLabel_1);

        JLabel lblNewLabel_2 = new JLabel("Which Player goes first (0/1):");
        lblNewLabel_2.setBounds(10, 83, 168, 14);
        contentPane.add(lblNewLabel_2);

        JLabel lblNewLabel_2_1 = new JLabel("Choose the Game Tree depth (2/4):");
        lblNewLabel_2_1.setBounds(10, 128, 197, 14);
        contentPane.add(lblNewLabel_2_1);

        JLabel lblNewLabel_2_1_1 = new JLabel("Select the Evaluation Function for
Player #1 (1/2/3):");
        lblNewLabel_2_1_1.setBounds(10, 177, 264, 14);
        contentPane.add(lblNewLabel_2_1_1);
```

```
        txtPlayer = new JTextField();
        txtPlayer.setBounds(188, 80, 86, 20);
        contentPane.add(txtPlayer);
        txtPlayer.setColumns(10);

        txtDepth = new JTextField();
        txtDepth.setColumns(10);
        txtDepth.setBounds(218, 125, 86, 20);
        contentPane.add(txtDepth);

        txtEval1 = new JTextField();
        txtEval1.setColumns(10);
        txtEval1.setBounds(293, 174, 86, 20);
        contentPane.add(txtEval1);

        JButton btnNewButton = new JButton("Start!");
        btnNewButton.setBounds(243, 261, 89, 23);
        contentPane.add(btnNewButton);
        start=btnNewButton;

        JLabel lblNewLabel_2_1_1_1 = new JLabel("Select the Evaluation Function  for
Player #2(1/2/3):");
        lblNewLabel_2_1_1_1.setBounds(10, 208, 264, 14);
        contentPane.add(lblNewLabel_2_1_1_1);

        txtEval2 = new JTextField();
```

```
        txtEval2.setColumns(10);
        txtEval2.setBounds(293, 205, 86, 20);
        contentPane.add(txtEval2);
    }
}
```

StaticGUI.java

```
package txstate.cs.ai.kala;

import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JLabel;
import java.awt.BorderLayout;
import java.awt.GridBagLayout;
import java.awt.GridLayout;
import java.awt.Color;
import java.awt.Font;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.util.HashMap;
import java.awt.event.ActionEvent;
import javax.swing.ImageIcon;

public class StaticGUI {

    public JFrame frmKalahGame;
```

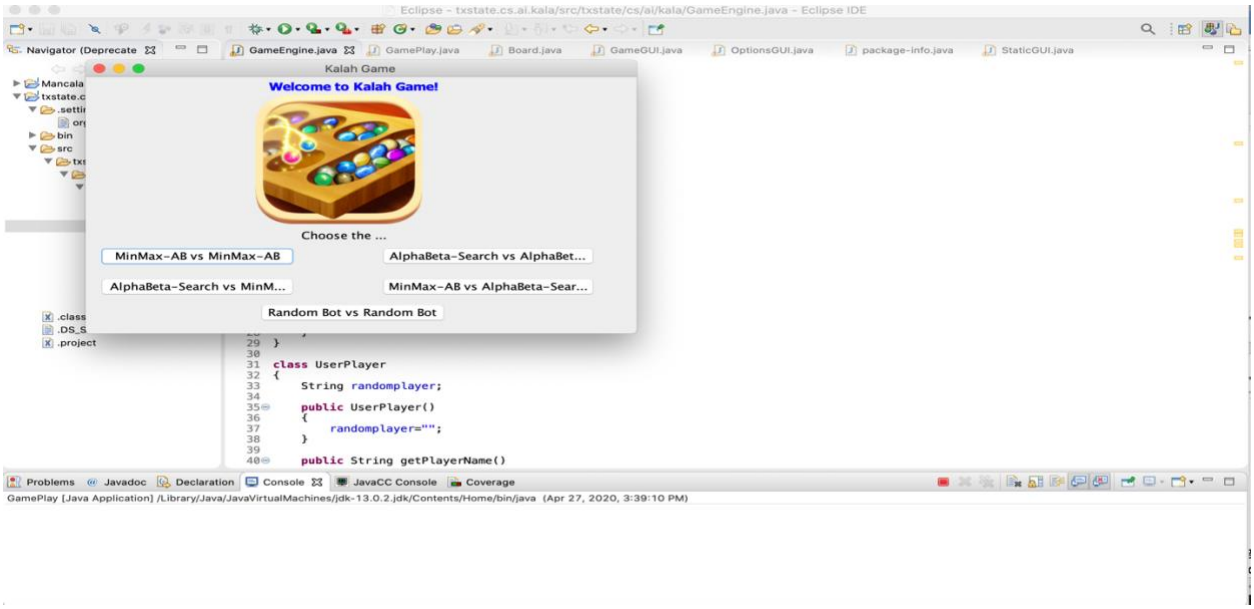
```
HashMap<String, JButton> mainbuttonlist = new HashMap<String, JButton>();
/**
 * Create the application.
 */
public StaticGUI() {
    initialize();
}

/**
 * Initialize the contents of the frame.
 */
private void initialize() {
    frmKalahGame = new JFrame();
    frmKalahGame.setTitle("Kalah Game");
    frmKalahGame.setBounds(100, 100, 562, 359);
    frmKalahGame.getContentPane().setLayout(null);

    JLabel lblNewLabel = new JLabel("Welcome to Kalah Game!");
    lblNewLabel.setFont(new Font("Tahoma", Font.BOLD, 14));
    lblNewLabel.setForeground(Color.BLUE);
    lblNewLabel.setBounds(187, 0, 218, 25);
    frmKalahGame.getContentPane().add(lblNewLabel);

    JLabel lblNewLabel_1 = new JLabel("Choose the play:");
    lblNewLabel_1.setBounds(220, 201, 93, 14);
    frmKalahGame.getContentPane().add(lblNewLabel_1);
```

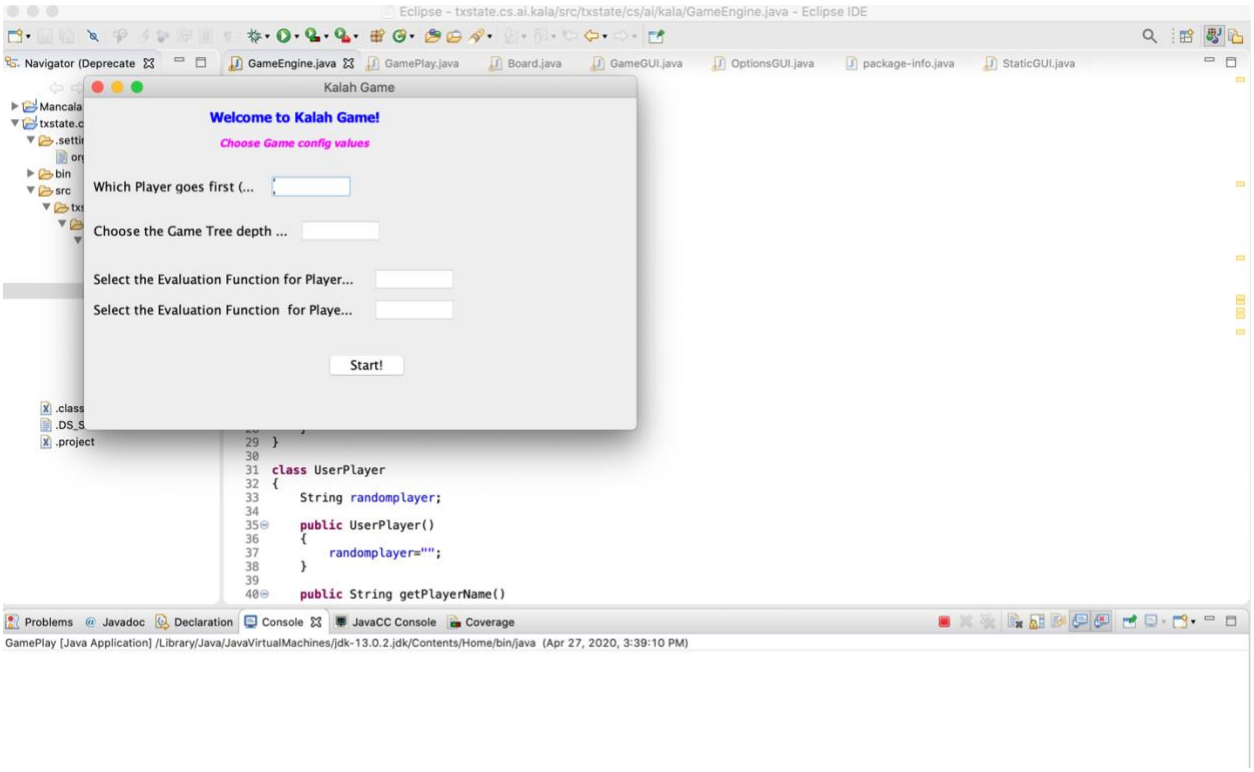
```
        JButton minmaxab = new JButton("MinMax-AB vs MinMax-AB");
        minmaxab.setBounds(10, 225, 208, 23);
        frmKalahGame.getContentPane().add(minmaxab);
        mainbuttonlist.put("minmaxab", minmaxab);

        JButton alphabetasearch = new JButton("AlphaBeta-Search vs AlphaBeta-
Search");
        alphabetasearch.setBounds(297, 224, 227, 24);
        frmKalahGame.getContentPane().add(alphabetasearch);
        mainbuttonlist.put("alphabetasearch", alphabetasearch);

        JButton uservsuser = new JButton("Random Bot vs Random Bot");
        uservsuser.setBounds(173, 299, 199, 23);
        frmKalahGame.getContentPane().add(uservsuser);
        mainbuttonlist.put("uservsuser", uservsuser);

        JButton alphamin = new JButton("AlphaBeta-Search vs MinMax-AB");
        alphamin.setBounds(10, 265, 208, 23);
        frmKalahGame.getContentPane().add(alphamin);
        mainbuttonlist.put("alphamin", alphamin);

        JLabel lblNewLabel_2 = new JLabel("New label");
        lblNewLabel_2.setIcon(new
ImageIcon(StaticGUI.class.getResource("/txstate/cs/ai/kala/icon.png")));
        lblNewLabel_2.setBounds(173, 22, 171, 173);
        frmKalahGame.getContentPane().add(lblNewLabel_2);
```

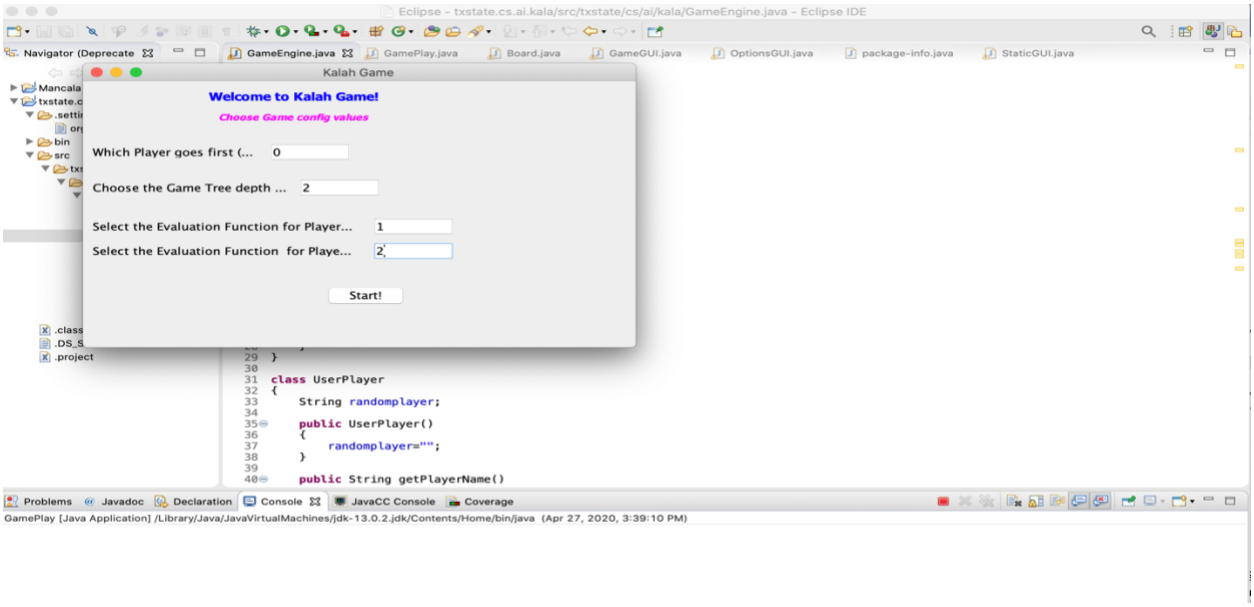
```
        JButton minalpha = new JButton("MinMax-AB vs AlphaBeta-Search ");
        minalpha.setBounds(297, 265, 227, 23);
        frmKalahGame.getContentPane().add(minalpha);
        mainbuttonlist.put("minalpha", minalpha);
    }
}
```

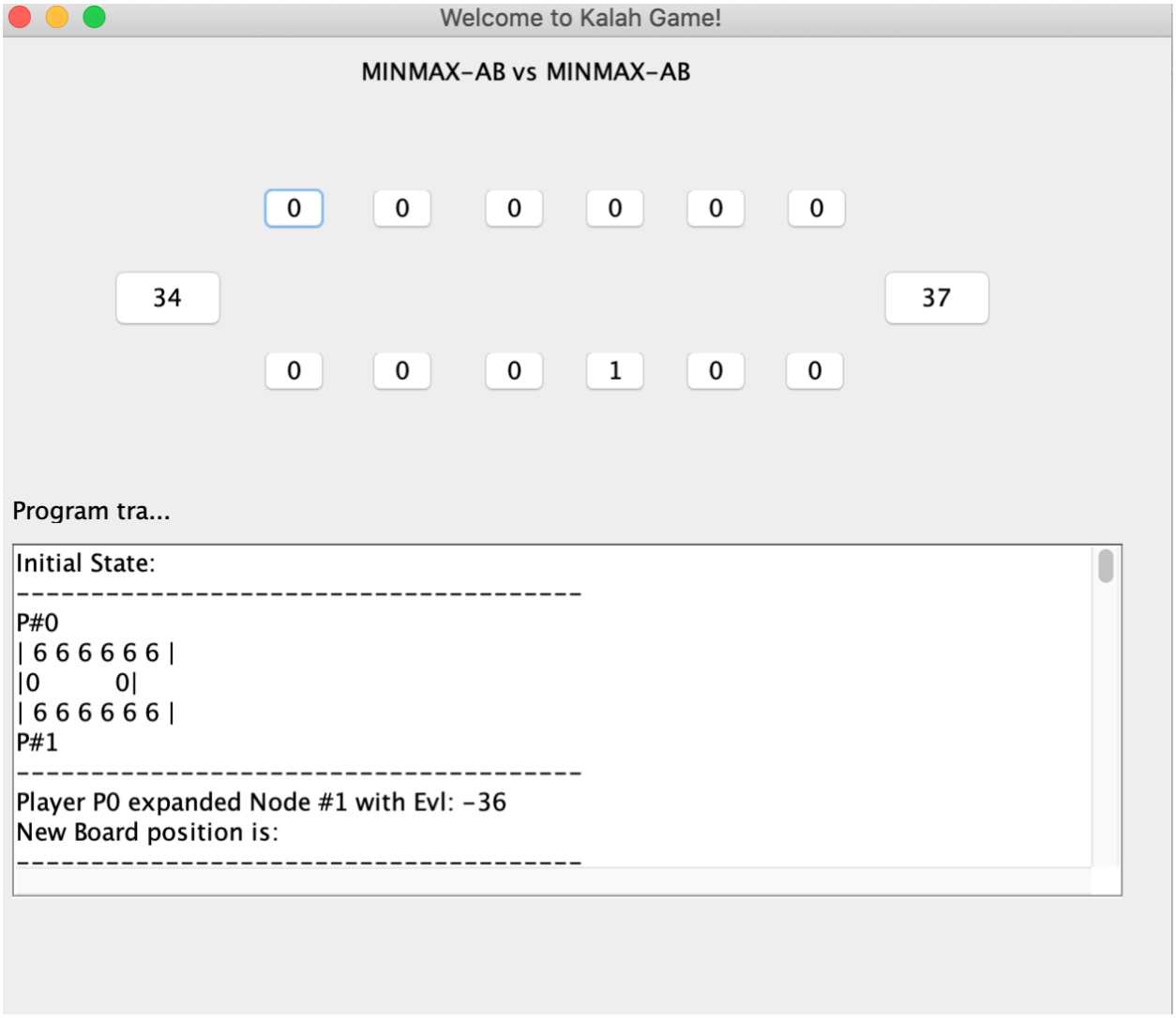
SCREENSHOTS OF SAMPLE RUNS

1) MINMAX-A-B Vs MINMAX-A-B



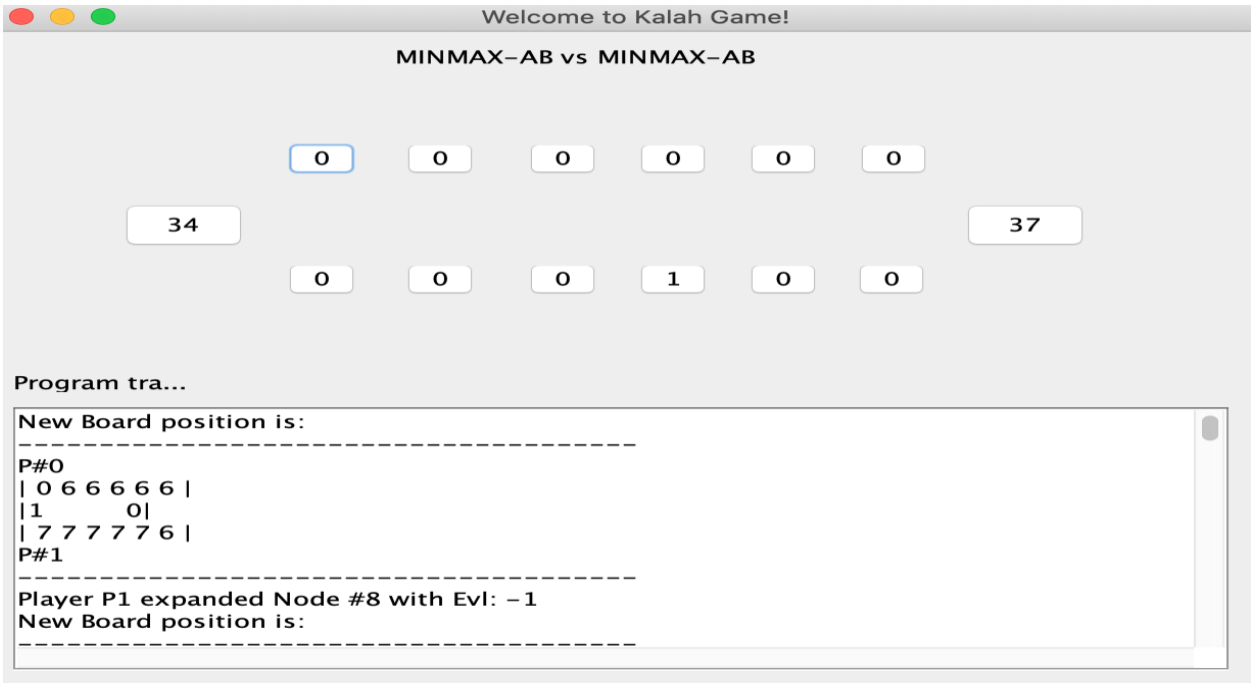

Eclipse - txstate.cs.ai.kala/src/txstate/cs/ai/kala/GameEngine.java - Eclipse IDE
Kalah Game
Welcome to Kalah Game!
Choose Game config values
Which Player goes first (...
Choose the Game Tree depth ...
Select the Evaluation Function for Player...
Select the Evaluation Function for Playe...
Start!
class UserPlayer
String randomplayer;
public UserPlayer()
randomplayer="";
public String getPlayerName()
Problems Javadoc Declaration Console JavaCC Console Coverage
GamePlay [Java Application] /Library/Java/JavaVirtualMachines/jdk-13.0.2.jdk/Contents/Home/bin/java (Apr 27, 2020, 3:39:10 PM)

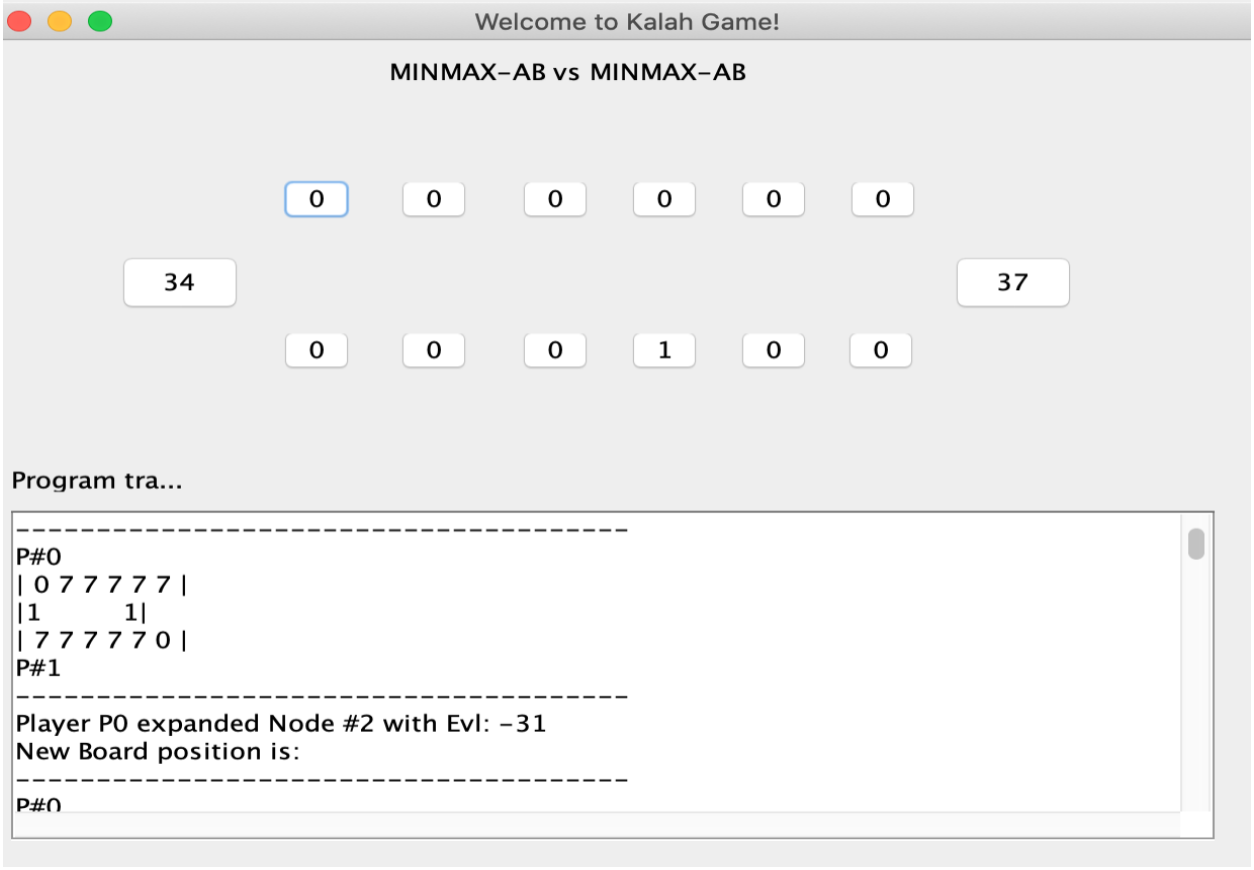

Eclipse - txstate.cs.ai.kala/src/txstate/cs/ai/kala/GameEngine.java - Eclipse IDE
Kalah Game
Welcome to Kalah Game!
Choose Game config values
Which Player goes first (... 0
Choose the Game Tree depth ... 2
Select the Evaluation Function for Player... 1
Select the Evaluation Function for Playe... 2
Start!
class UserPlayer
String randomplayer;
public UserPlayer()
randomplayer="";
public String getPlayerName()
Problems Javadoc Declaration Console JavaCC Console Coverage
GamePlay [Java Application] /Library/Java/JavaVirtualMachines/jdk-13.0.2.jdk/Contents/Home/bin/java (Apr 27, 2020, 3:39:10 PM)

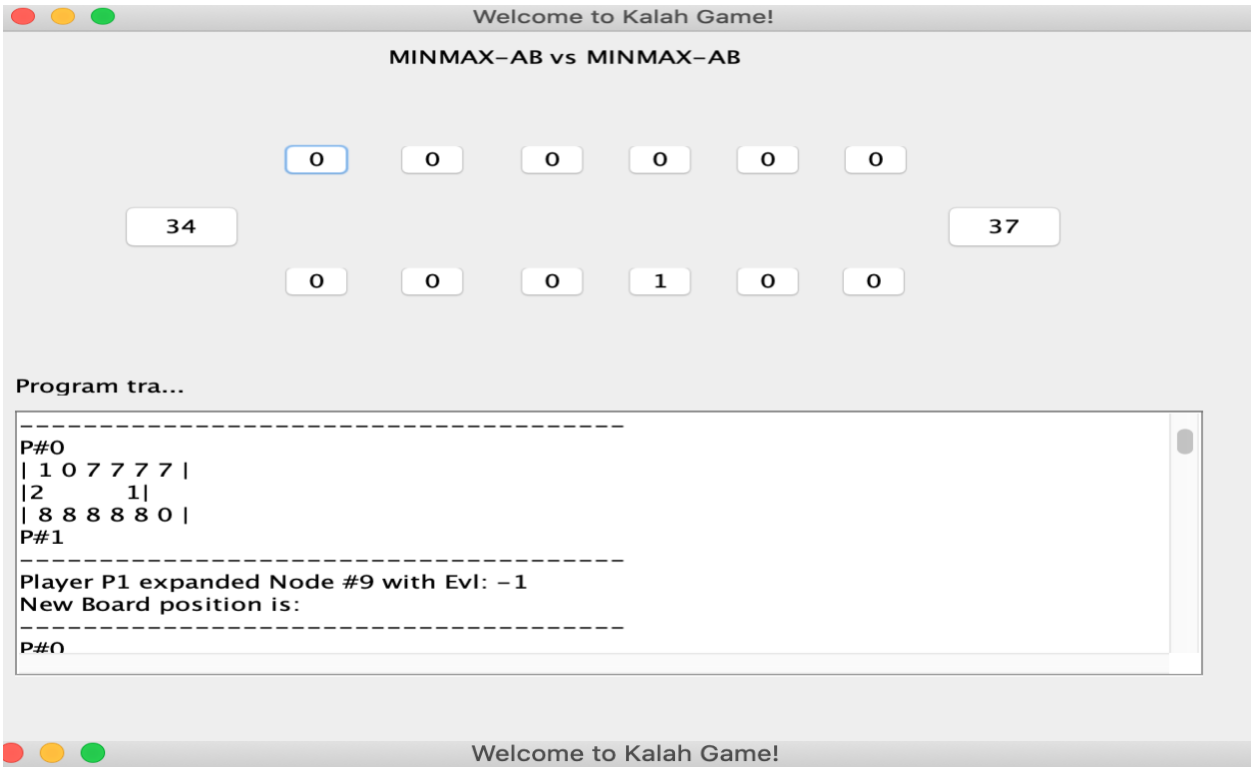

Welcome to Kalah Game!
MINMAX-AB vs MINMAX-AB
0 0 0 0 0 0
34
37
0 0 0 1 0 0
Program tra...
Initial State:

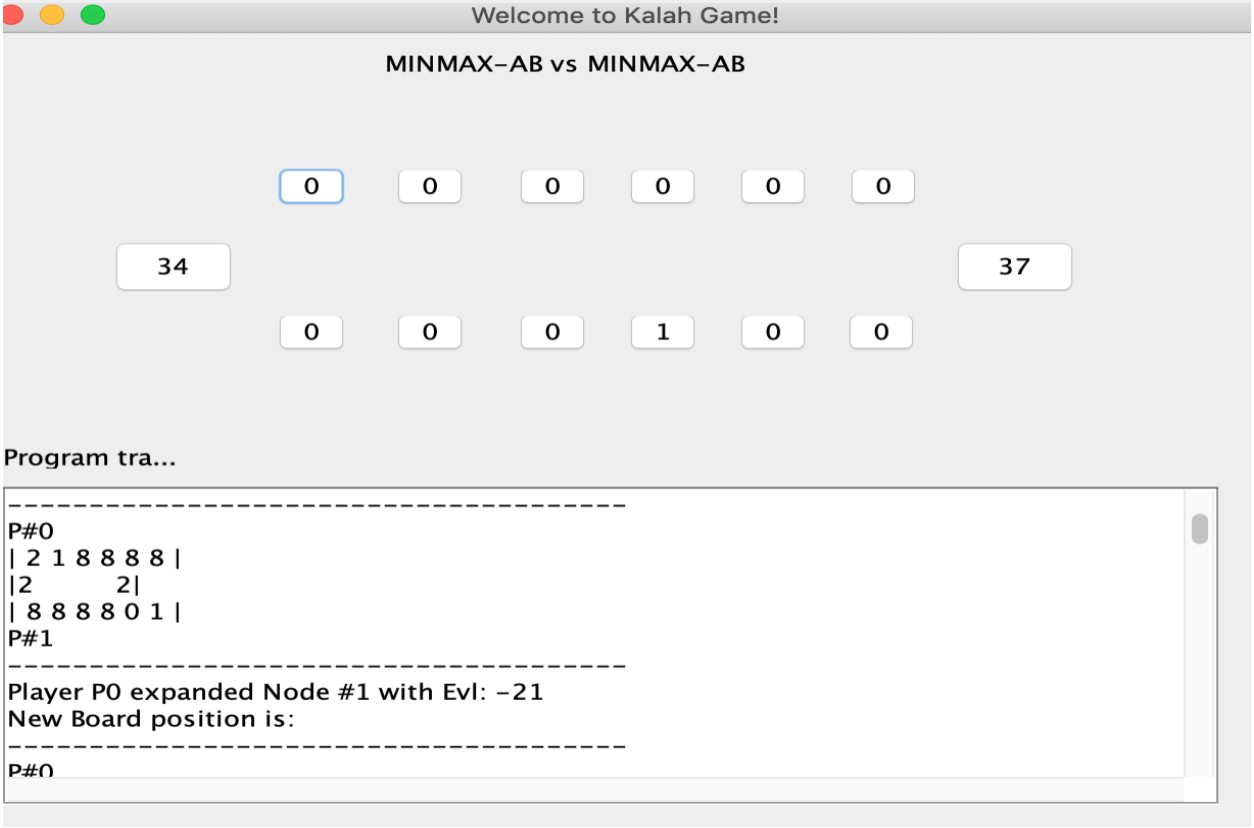
P#0
| 6 6 6 6 6 6 |
|0 0|
| 6 6 6 6 6 6 |
P#1

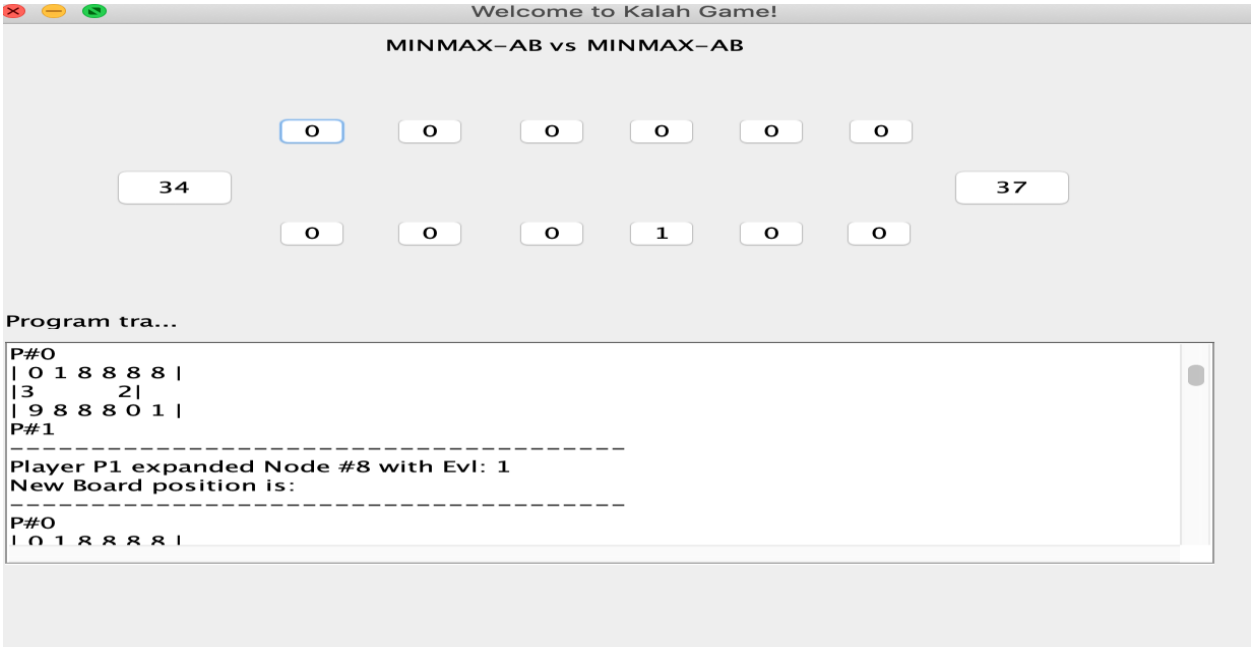
Player P0 expanded Node #1 with Evl: -36
New Board position is:

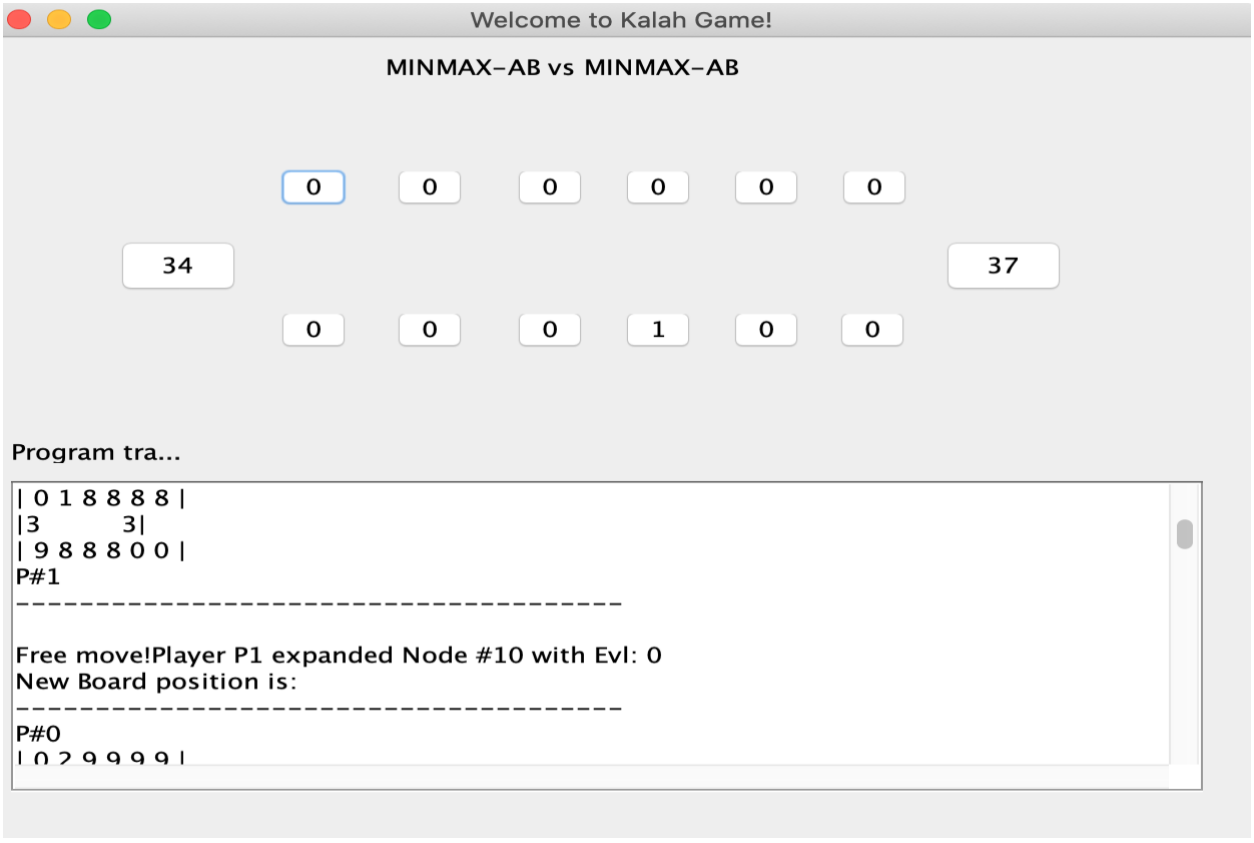

Welcome to Kalah Game!
MINMAX-AB vs MINMAX-AB
0 0 0 0 0 0
34
37
0 0 0 1 0 0
Program tra...
New Board position is:
P#0
| 0 6 6 6 6 6 |
|1 0|
| 7 7 7 7 7 6 |
P#1
Player P1 expanded Node #8 with Evl: -1
New Board position is:

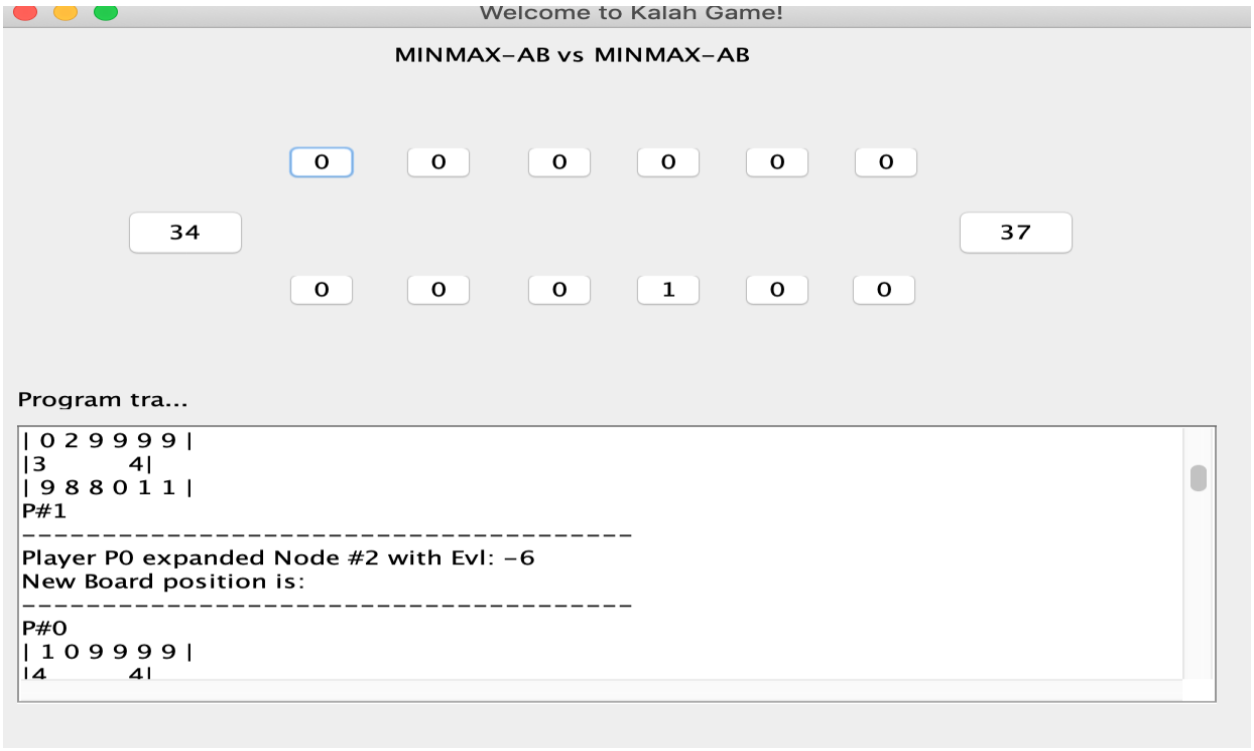

Welcome to Kalah Game!
MINMAX-AB vs MINMAX-AB
0 0 0 0 0 0
34
37
0 0 0 1 0 0
Program tra...
P#0
| 0 7 7 7 7 7 |
|1 1|
| 7 7 7 7 7 0 |
P#1
Player P0 expanded Node #2 with Evl: -31
New Board position is:
P#0

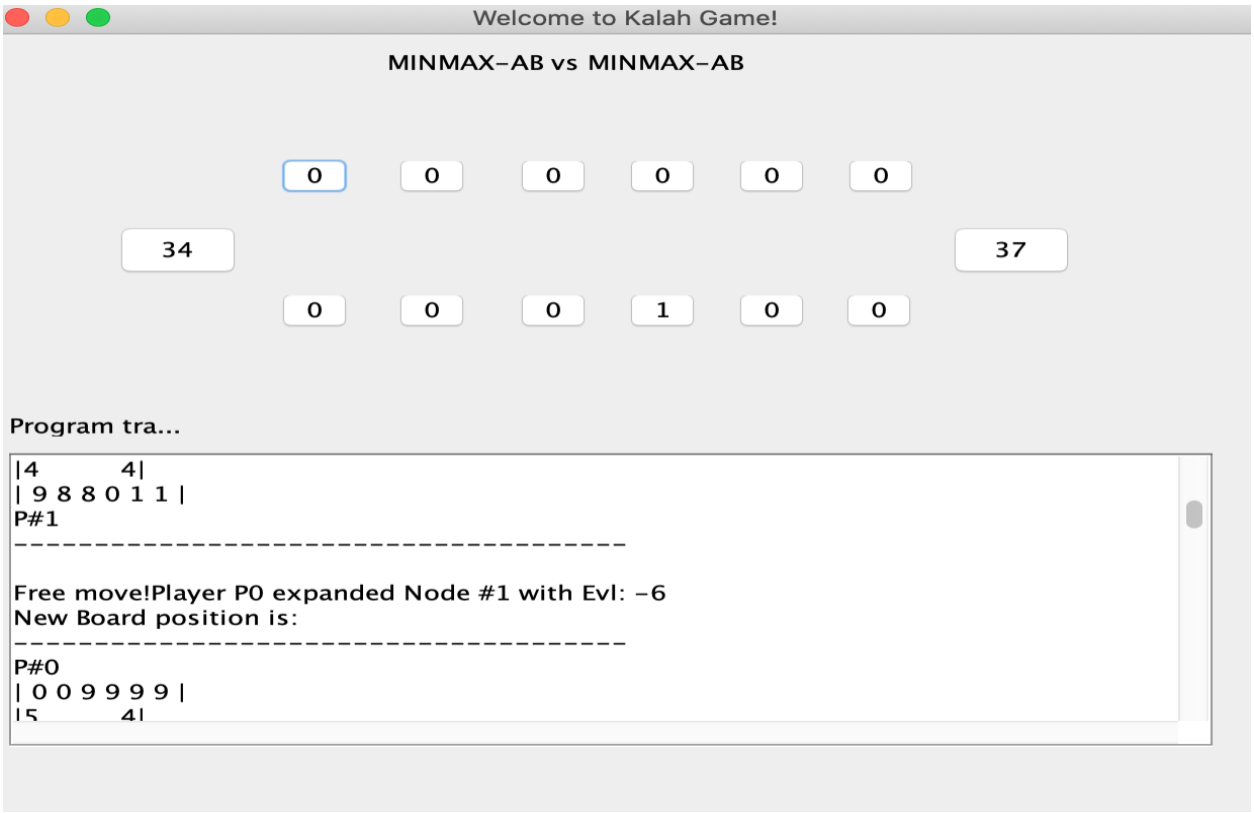

Welcome to Kalah Game!
MINMAX–AB vs MINMAX–AB
0 0 0 0 0 0
34
37
0 0 0 1 0 0
Program tra...
P#0
| 1 0 7 7 7 7 |
|2 1|
| 8 8 8 8 8 0 |
P#1
Player P1 expanded Node #9 with Evl: –1
New Board position is:
P#0

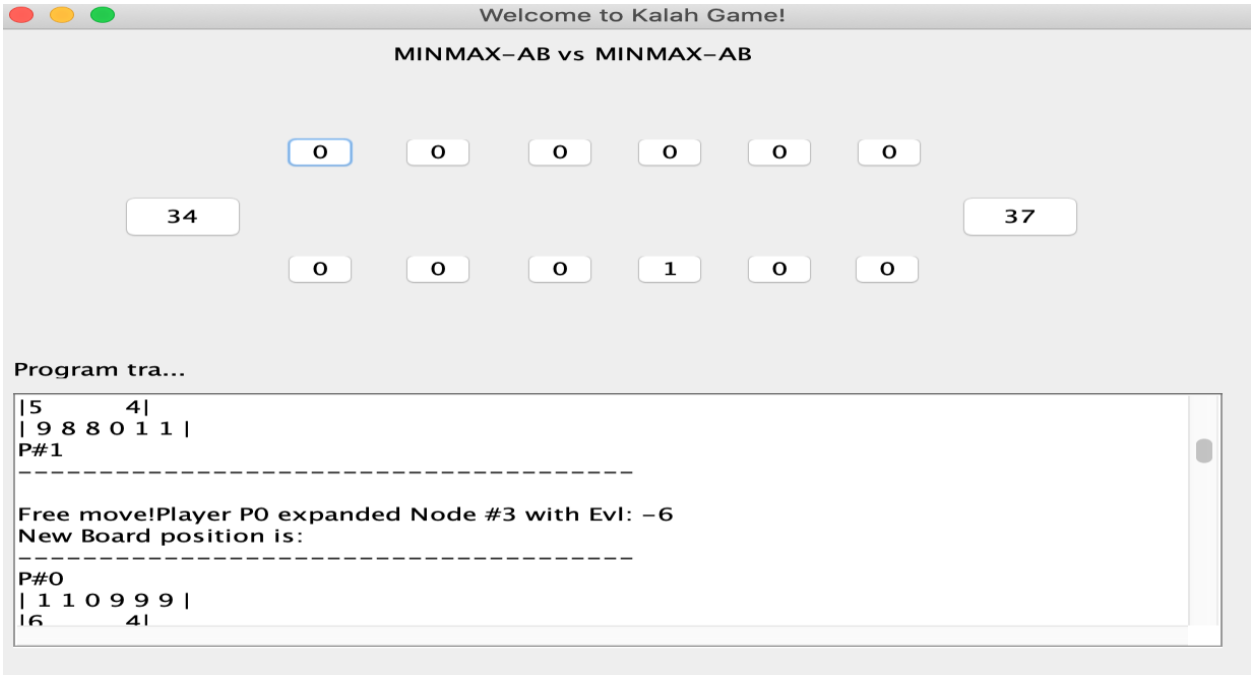

Welcome to Kalah Game!
MINMAX–AB vs MINMAX–AB
0 0 0 0 0 0
34
37
0 0 0 1 0 0
Program tra...
P#0
| 2 1 8 8 8 8 |
|2 2|
| 8 8 8 8 0 1 |
P#1
Player P0 expanded Node #1 with Evl: –21
New Board position is:
P#0

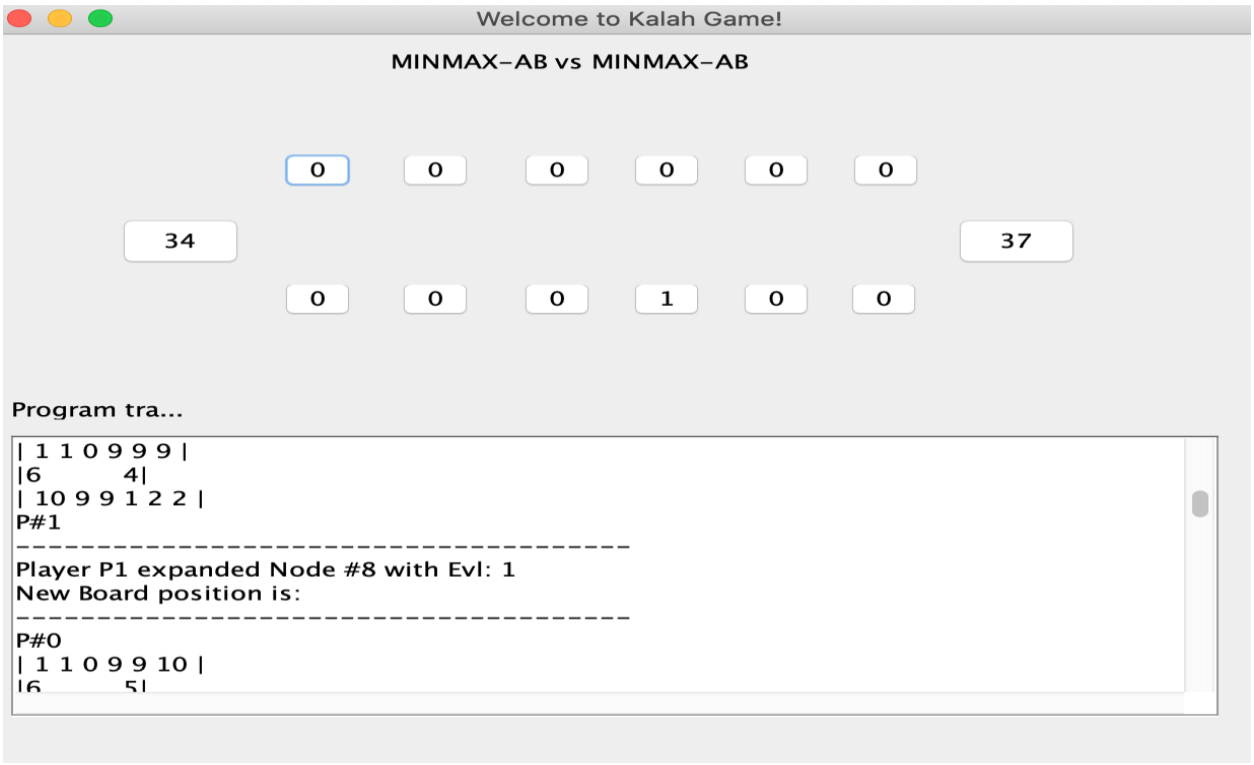

Welcome to Kalah Game!
MINMAX-AB vs MINMAX-AB
0 0 0 0 0 0
34
37
0 0 0 1 0 0
Program tra...
P#0
| 0 1 8 8 8 8 |
|3 2|
| 9 8 8 8 0 1 |
P#1
Player P1 expanded Node #8 with Evl: 1
New Board position is:
P#0

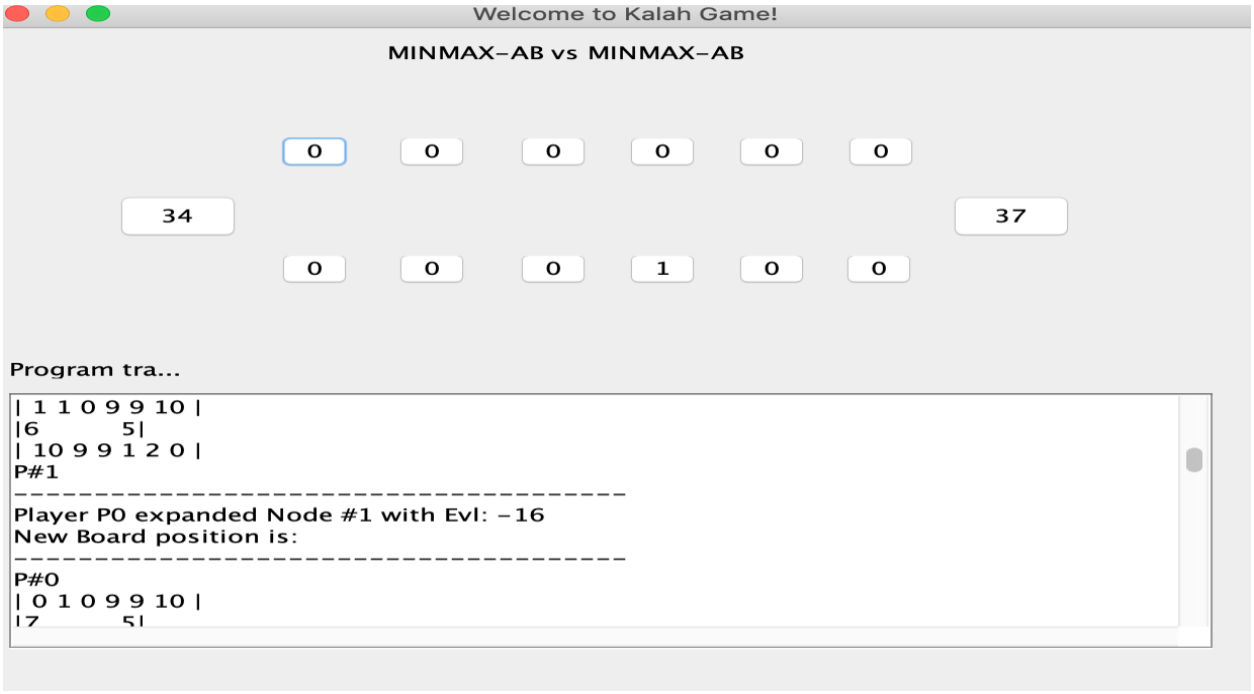

Welcome to Kalah Game!
MINMAX-AB vs MINMAX-AB
0 0 0 0 0 0
34
37
0 0 0 1 0 0
Program tra...
| 0 1 8 8 8 8 |
|3 3|
| 9 8 8 8 0 0 |
P#1
Free move!Player P1 expanded Node #10 with Evl: 0
New Board position is:
P#0

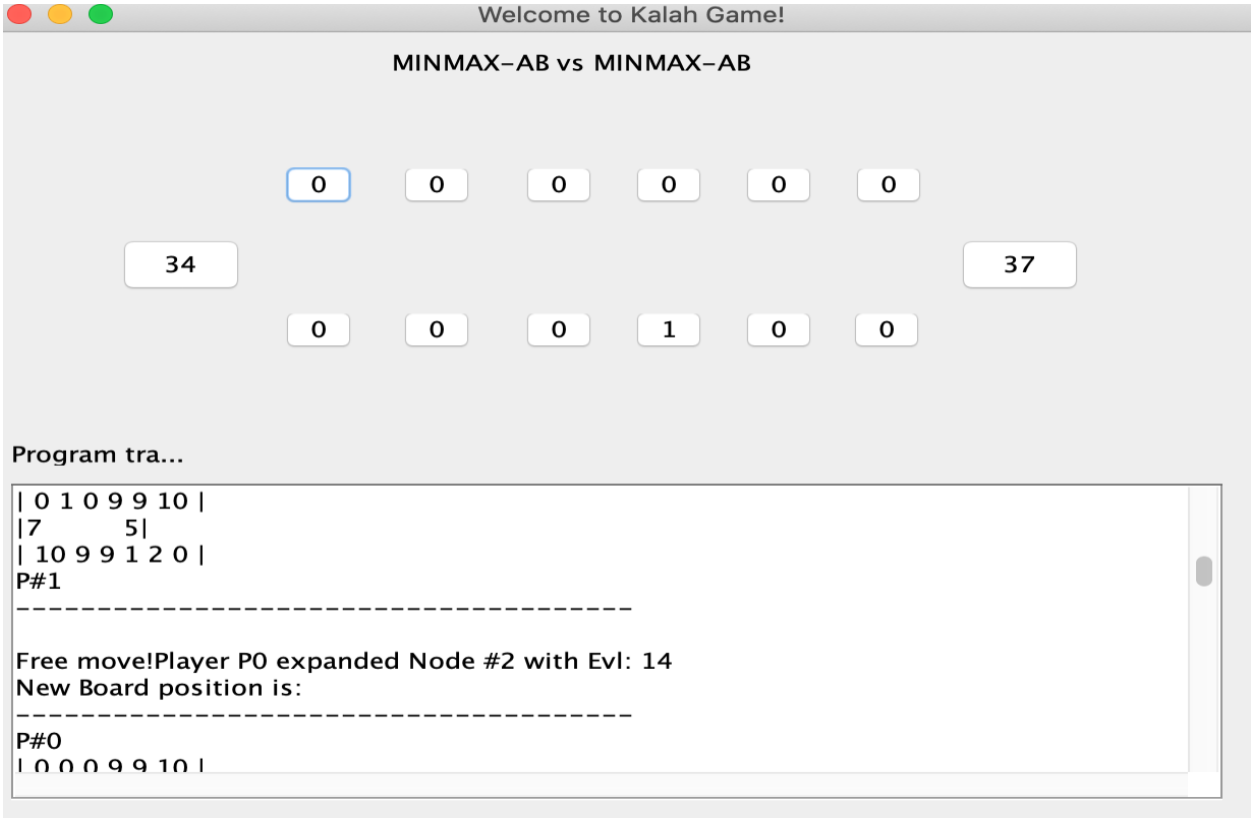

Welcome to Kalah Game!
MINMAX–AB vs MINMAX–AB
0 0 0 0 0 0
34
37
0 0 0 1 0 0
Program tra...
| 0 2 9 9 9 9 |
|3 4|
| 9 8 8 0 1 1 |
P#1
Player P0 expanded Node #2 with Evl: –6
New Board position is:
P#0
| 1 0 9 9 9 9 |

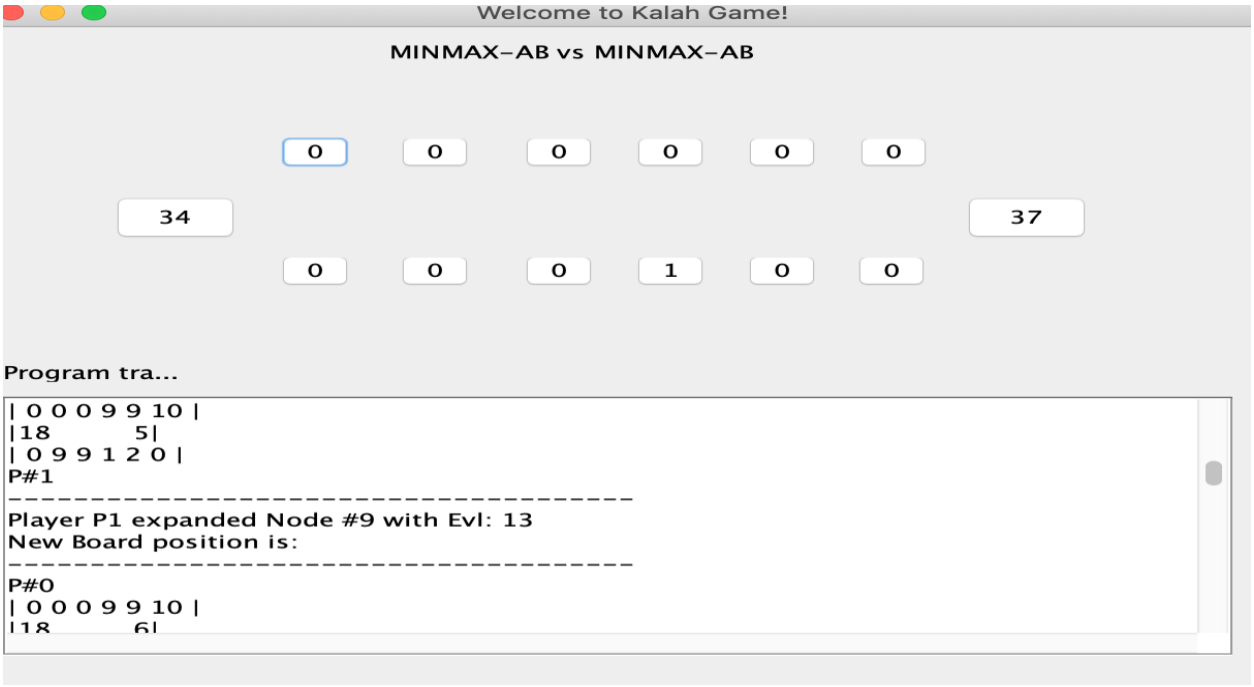

Welcome to Kalah Game!
MINMAX–AB vs MINMAX–AB
0 0 0 0 0 0
34
37
0 0 0 1 0 0
Program tra...
|4 4|
| 9 8 8 0 1 1 |
P#1
Free move!Player P0 expanded Node #1 with Evl: –6
New Board position is:
P#0
| 0 0 9 9 9 9 |

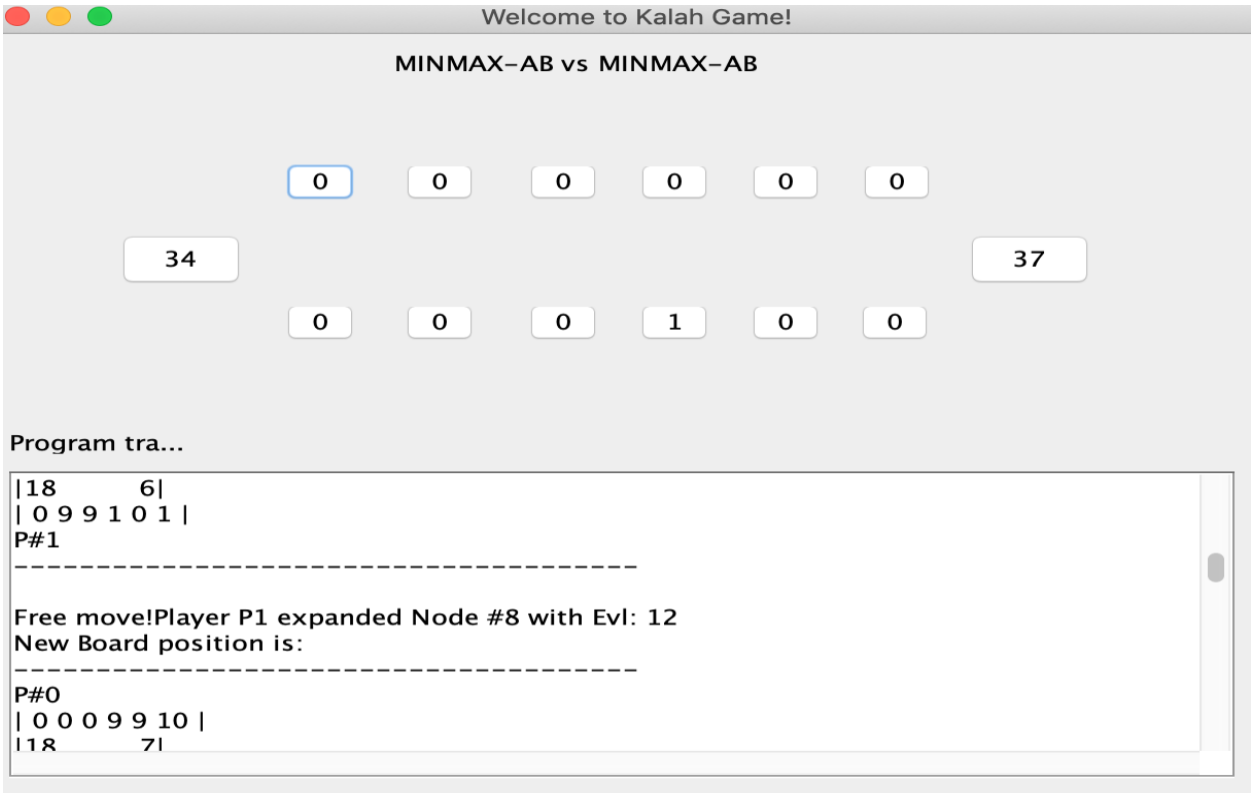

Welcome to Kalah Game!
MINMAX–AB vs MINMAX–AB
0 0 0 0 0 0
34
37
0 0 0 1 0 0
Program tra...
|5 4|
| 9 8 8 0 1 1 |
P#1
Free move!Player P0 expanded Node #3 with Evl: –6
New Board position is:
P#0
| 1 1 0 9 9 9 |
|6 4|

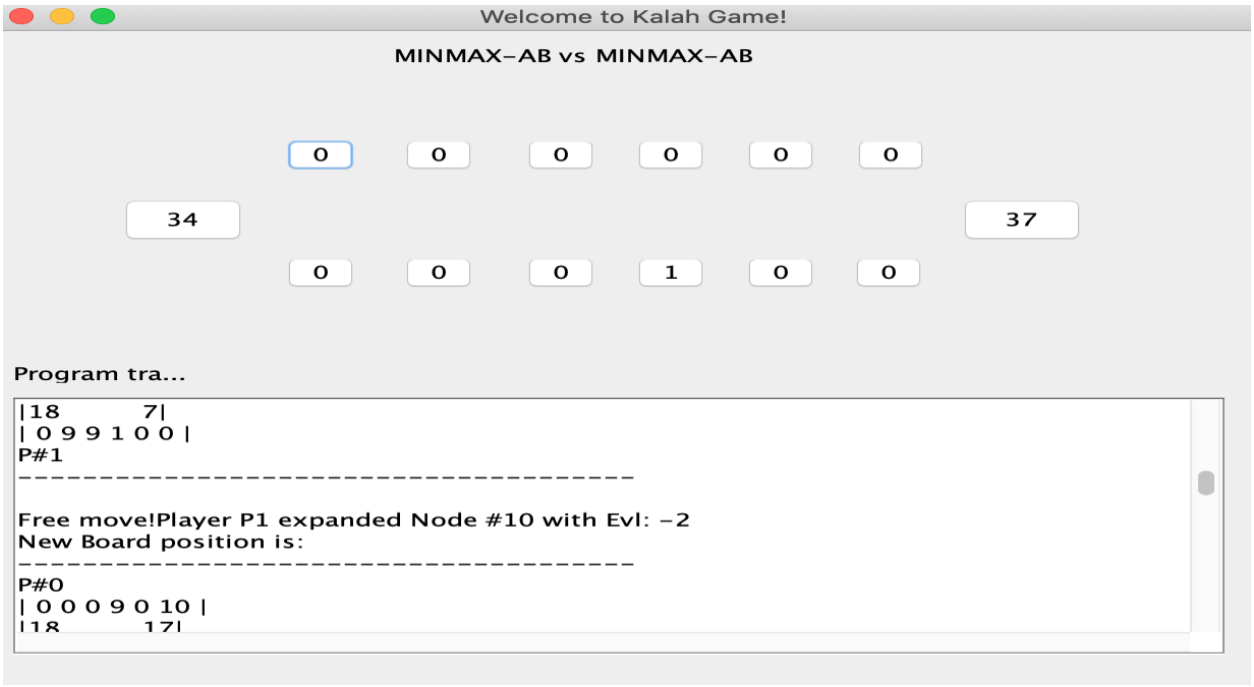

Welcome to Kalah Game!
MINMAX–AB vs MINMAX–AB
0 0 0 0 0 0
34
37
0 0 0 1 0 0
Program tra...
| 1 1 0 9 9 9 |
|6 4|
| 10 9 9 1 2 2 |
P#1
Player P1 expanded Node #8 with Evl: 1
New Board position is:
P#0
| 1 1 0 9 9 10 |
|6 5|


Welcome to Kalah Game!
MINMAX–AB vs MINMAX–AB
0 0 0 0 0 0
34
37
0 0 0 1 0 0
Program tra...
| 1 1 0 9 9 10 |
|6 5|
| 10 9 9 1 2 0 |
P#1
Player P0 expanded Node #1 with Evl: –16
New Board position is:
P#0
| 0 1 0 9 9 10 |


Welcome to Kalah Game!
MINMAX–AB vs MINMAX–AB
0 0 0 0 0 0
34
37
0 0 0 1 0 0
Program tra...
| 0 1 0 9 9 10 |
|7 5|
| 10 9 9 1 2 0 |
P#1
Free move!Player P0 expanded Node #2 with Evl: 14
New Board position is:
P#0

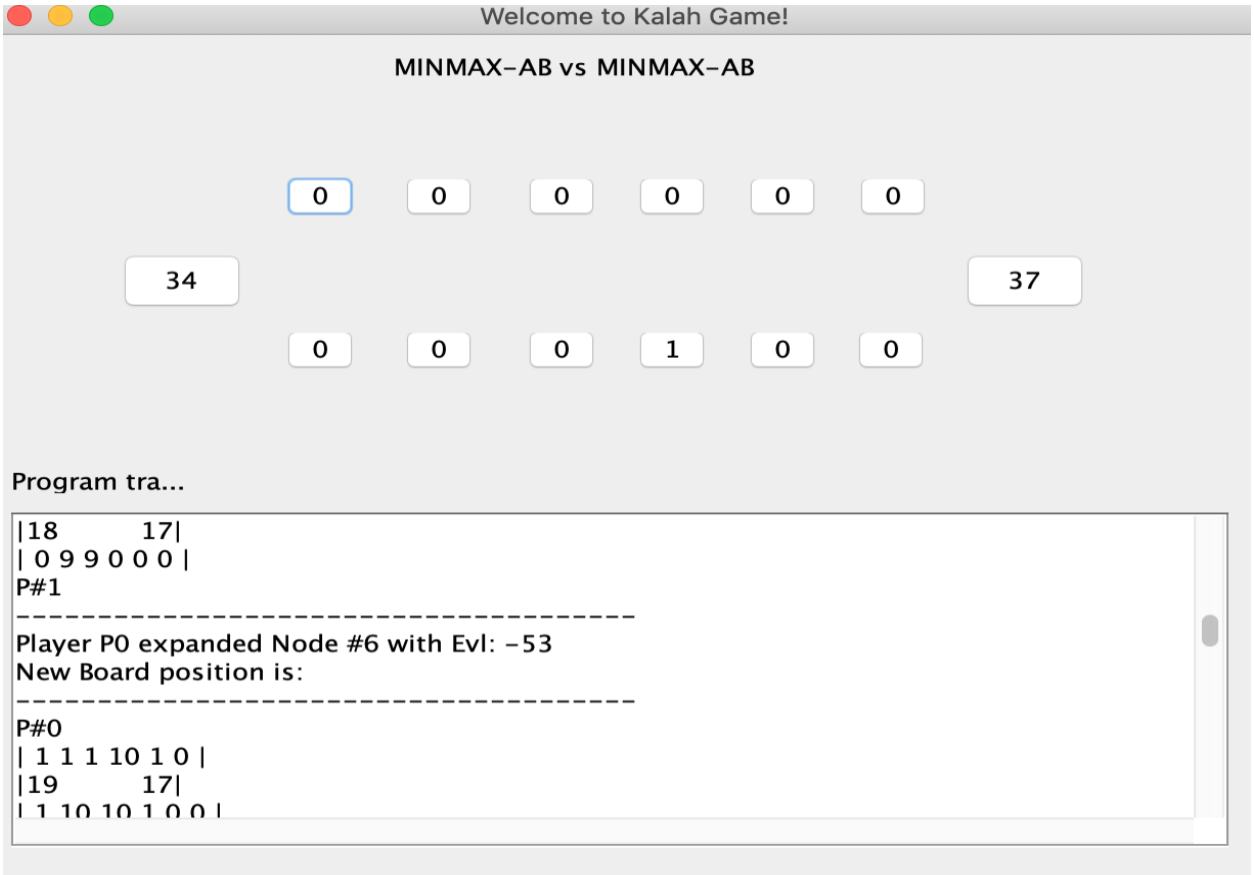

Welcome to Kalah Game!
MINMAX–AB vs MINMAX–AB
0 0 0 0 0 0
34
37
0 0 0 1 0 0
Program tra...
| 0 0 0 9 9 10 |
|18 5|
| 0 9 9 1 2 0 |
P#1
Player P1 expanded Node #9 with Evl: 13
New Board position is:
P#0
| 0 0 0 9 9 10 |


Welcome to Kalah Game!
MINMAX–AB vs MINMAX–AB
0 0 0 0 0 0
34
37
0 0 0 1 0 0
Program tra...
|18 6|
| 0 9 9 1 0 1 |
P#1
Free move!Player P1 expanded Node #8 with Evl: 12
New Board position is:
P#0
| 0 0 0 9 9 10 |


Welcome to Kalah Game!
MINMAX-AB vs MINMAX-AB
0 0 0 0 0 0
34
37
0 0 0 1 0 0
Program tra...
|18 7|
| 0 9 9 1 0 0 |
P#1

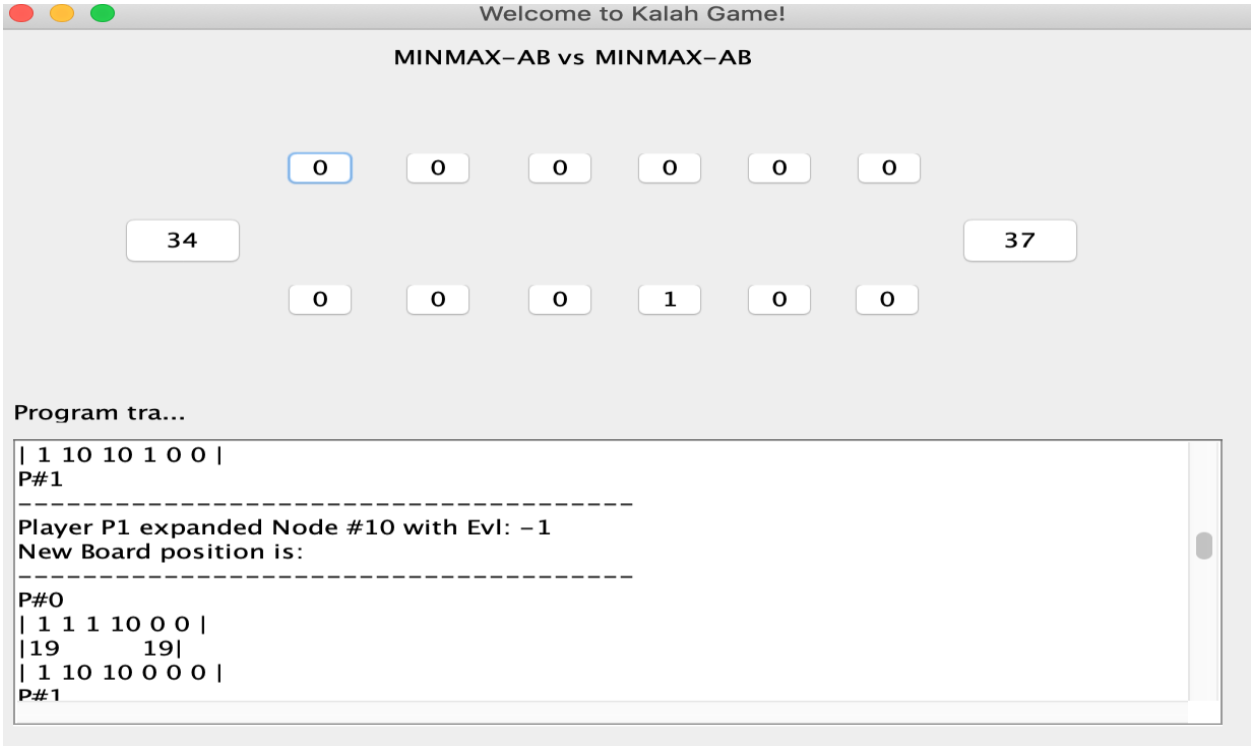
Free move!Player P1 expanded Node #10 with Evl: -2
New Board position is:

P#0
| 0 0 0 9 0 10 |
|18 17|


Welcome to Kalah Game!
MINMAX-AB vs MINMAX-AB
0 0 0 0 0 0
34
37
0 0 0 1 0 0
Program tra...
|18 17|
| 0 9 9 0 0 0 |
P#1

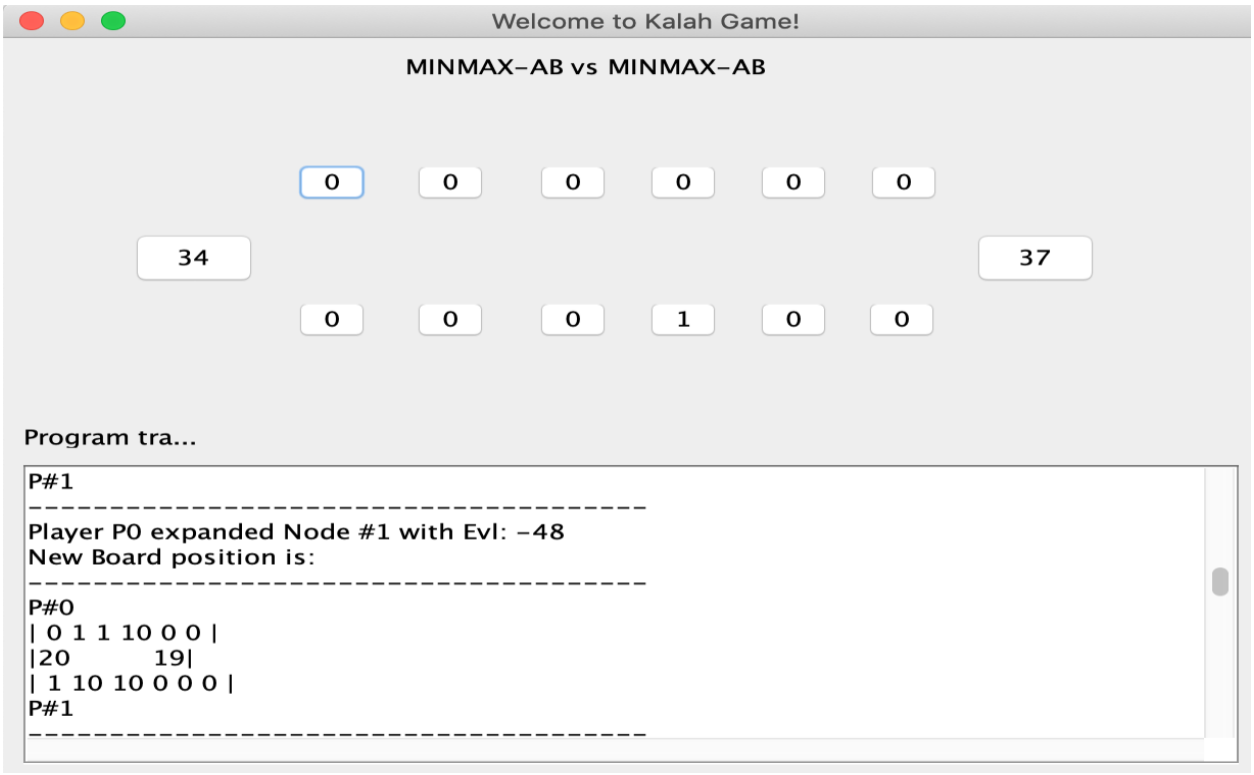
Player P0 expanded Node #6 with Evl: -53
New Board position is:

P#0
| 1 1 1 10 1 0 |
|19 17|
| 1 10 10 1 0 0 |


Welcome to Kalah Game!
MINMAX–AB vs MINMAX–AB
0 0 0 0 0 0
34
37
0 0 0 1 0 0
Program tra...
| 1 10 10 1 0 0 |
P#1

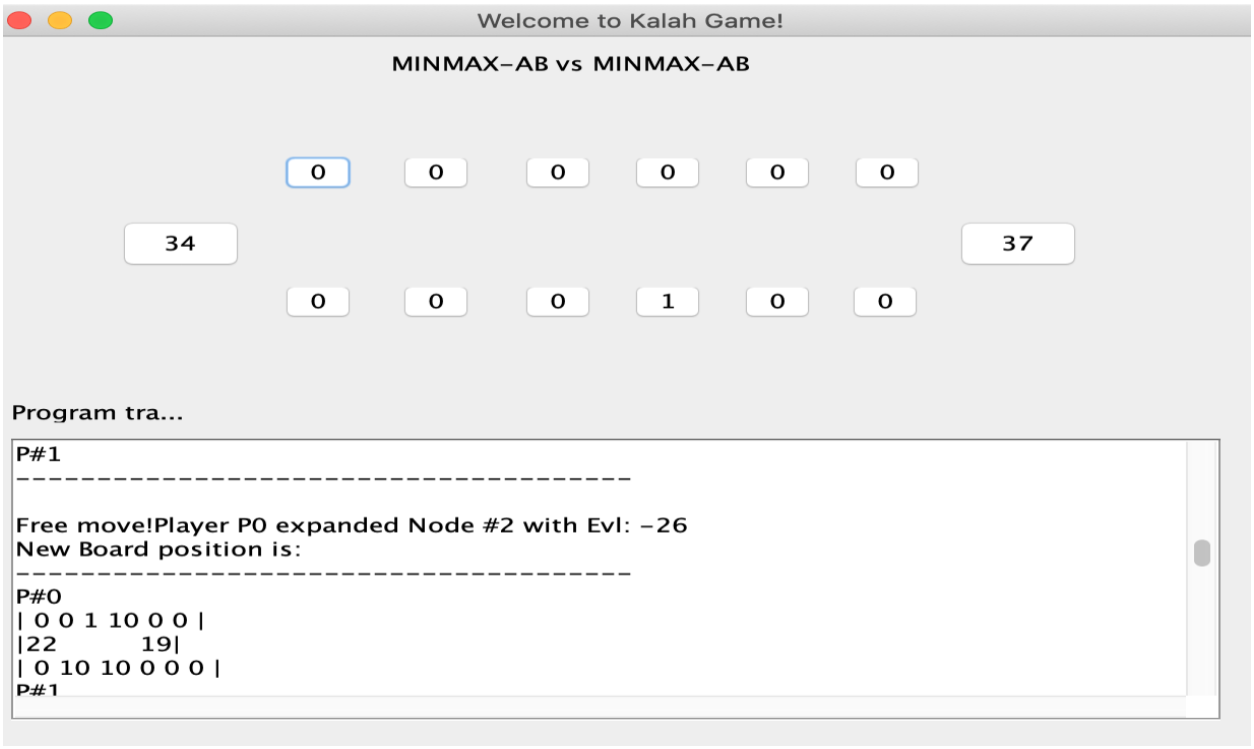
Player P1 expanded Node #10 with Evl: –1
New Board position is:

P#0
| 1 1 1 10 0 0 |
|19 19|
| 1 10 10 0 0 0 |
P#1


Welcome to Kalah Game!
MINMAX–AB vs MINMAX–AB
0 0 0 0 0 0
34
37
0 0 0 1 0 0
Program tra...
P#1

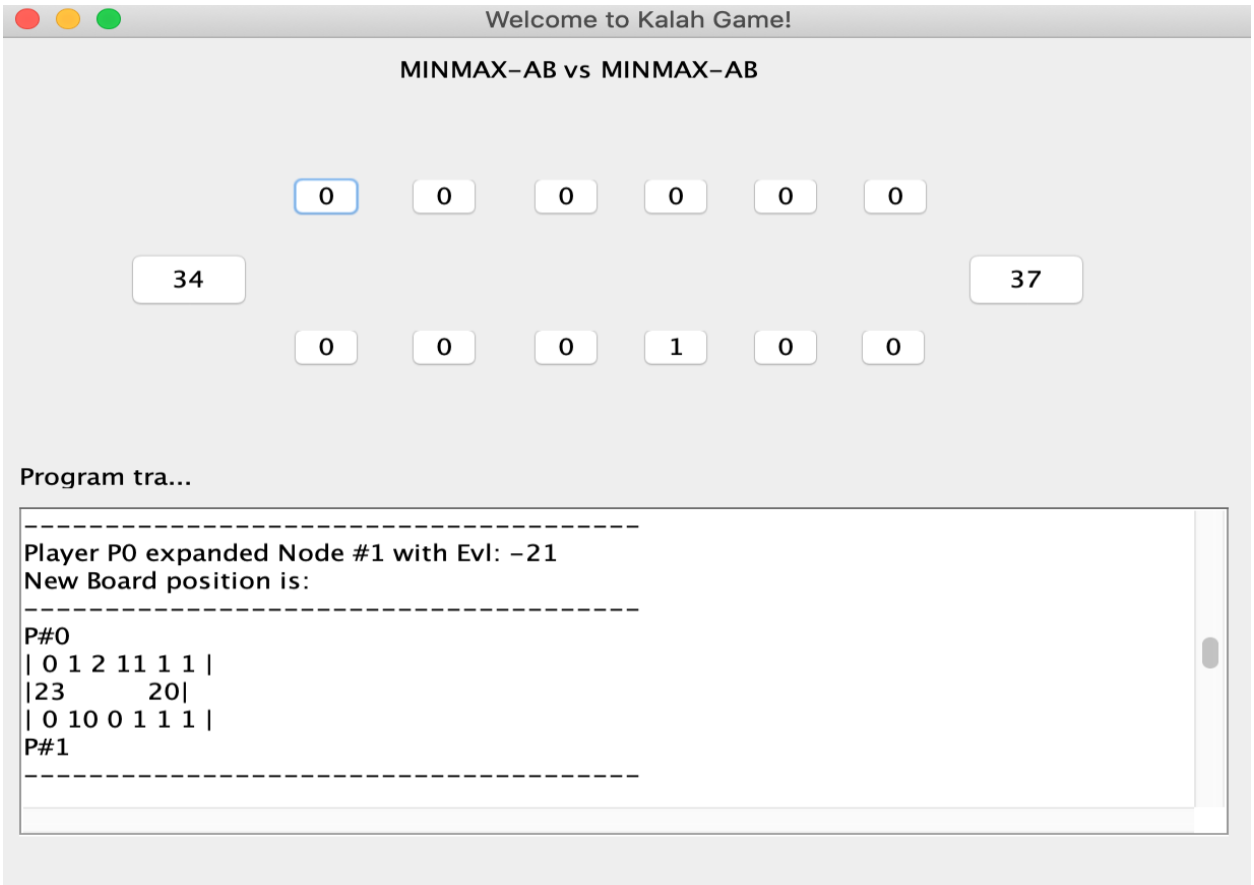
Player P0 expanded Node #1 with Evl: –48
New Board position is:

P#0
| 0 1 1 10 0 0 |
|20 19|
| 1 10 10 0 0 0 |
P#1


Welcome to Kalah Game!
MINMAX–AB vs MINMAX–AB
0 0 0 0 0 0
34
37
0 0 0 1 0 0
Program tra...
P#1

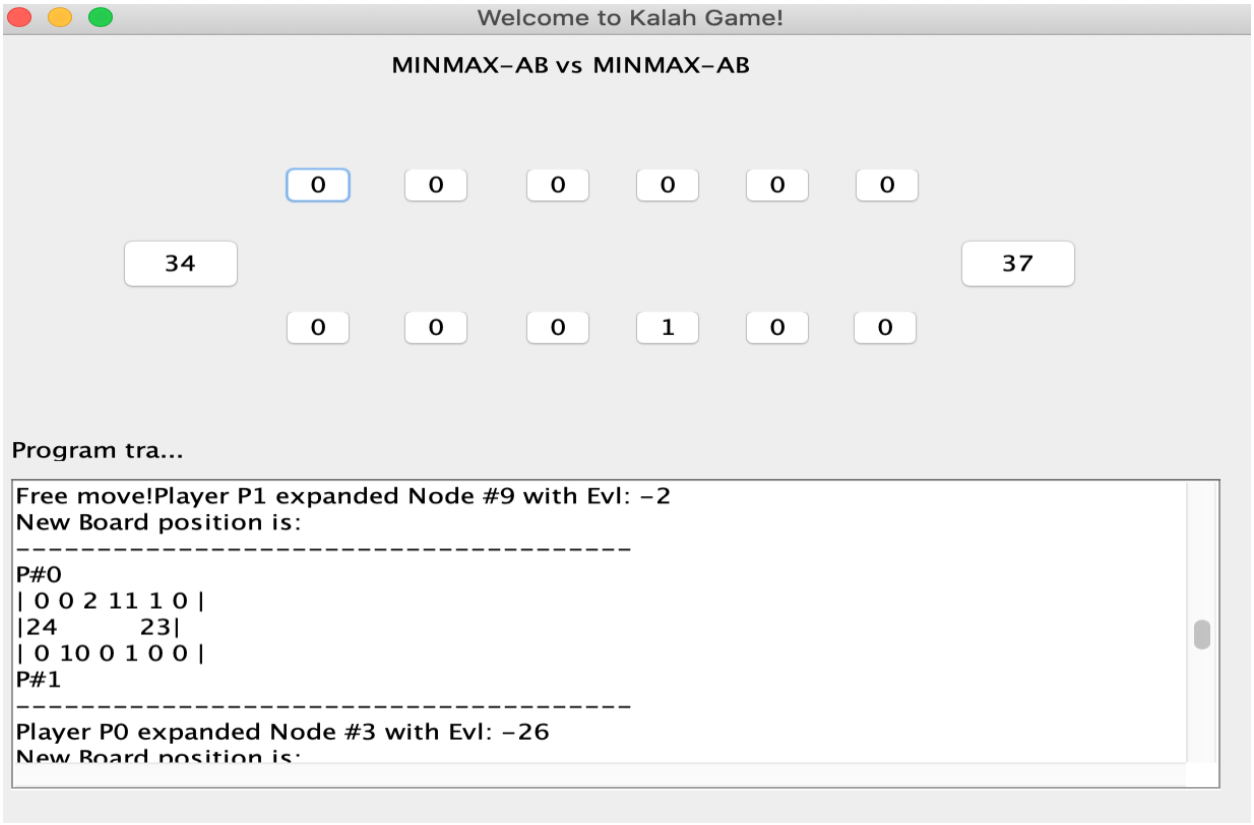
Free move!Player P0 expanded Node #2 with Evl: -26
New Board position is:

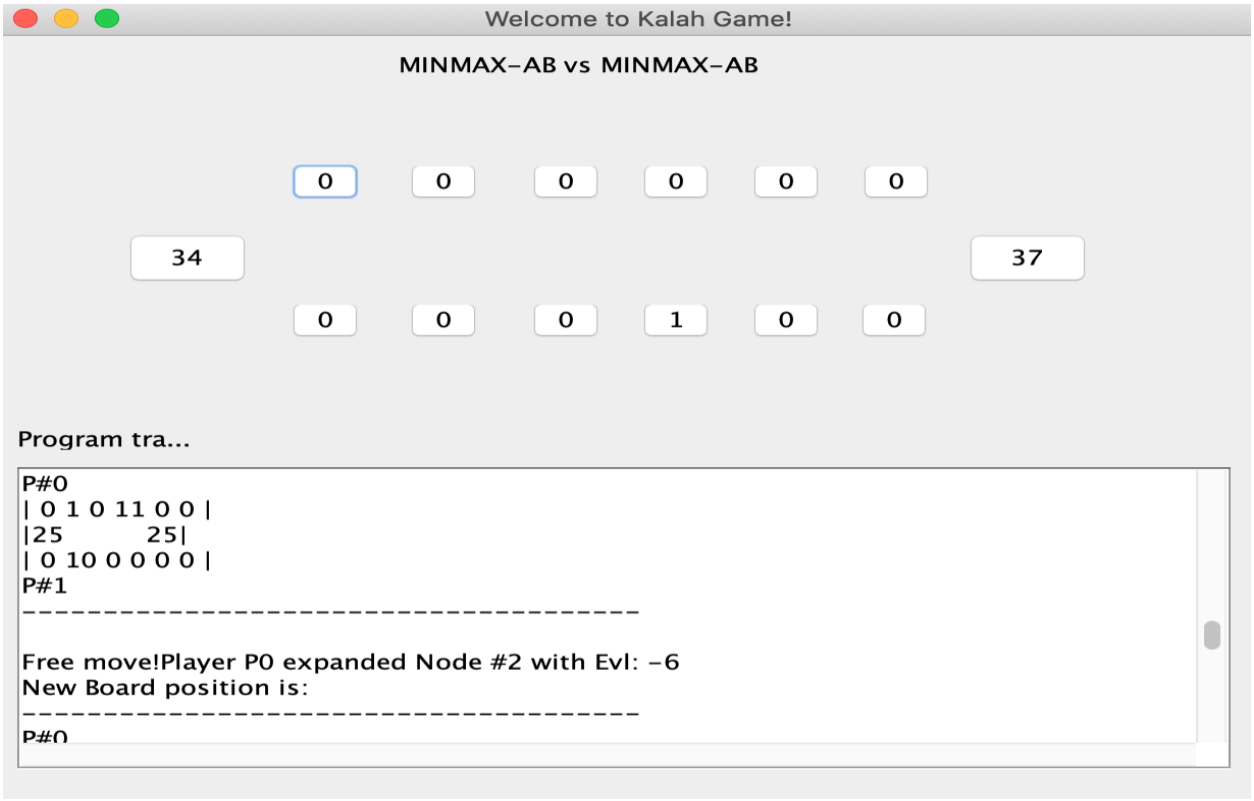
P#0
| 0 0 1 10 0 0 |
|22 19|
| 0 10 10 0 0 0 |
P#1

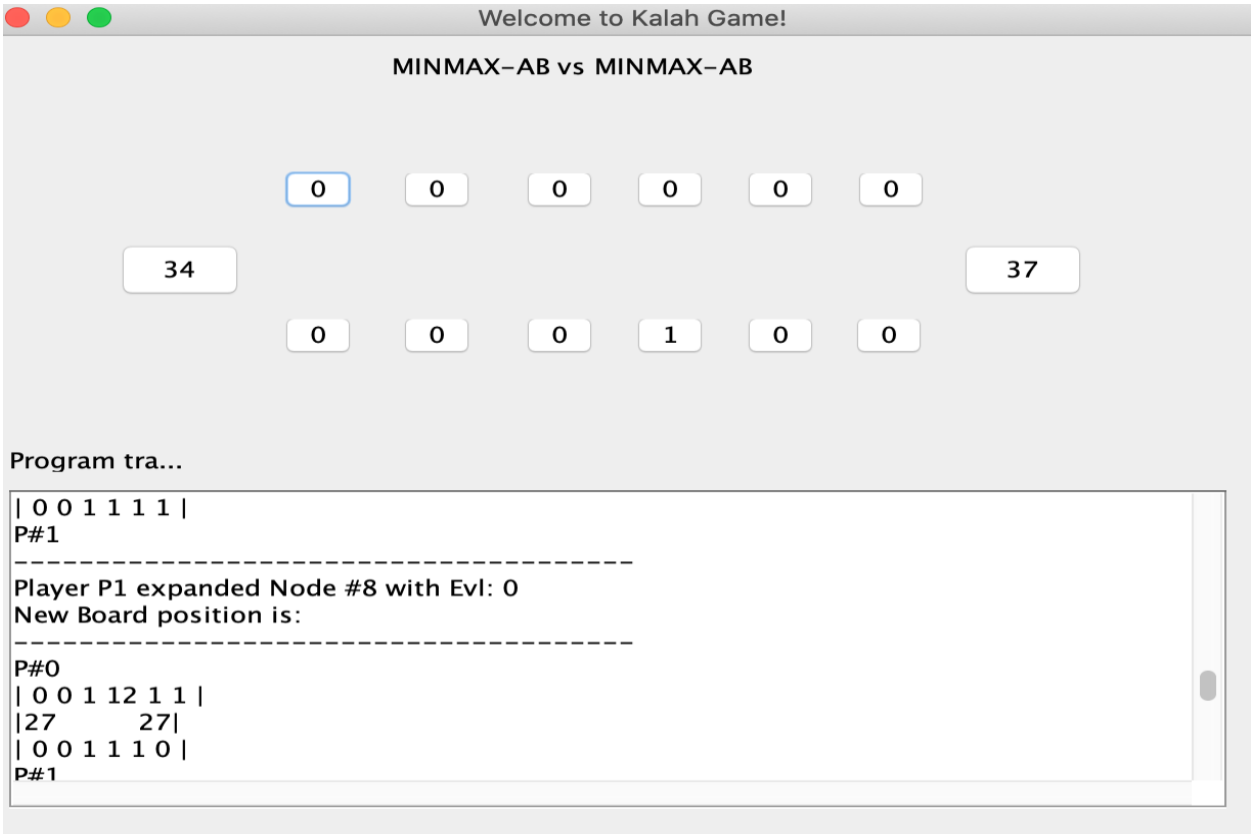

Welcome to Kalah Game!
MINMAX–AB vs MINMAX–AB
0 0 0 0 0 0
34
37
0 0 0 1 0 0
Program tra...

Player P0 expanded Node #1 with Evl: -21
New Board position is:

P#0
| 0 1 2 11 1 1 |
|23 20|
| 0 10 0 1 1 1 |
P#1

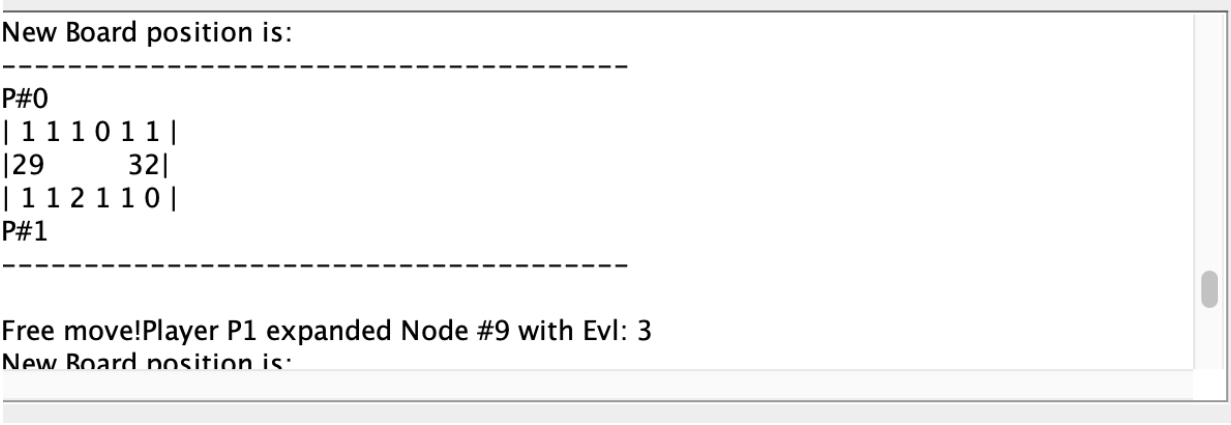

Welcome to Kalah Game!
MINMAX–AB vs MINMAX–AB
0 0 0 0 0 0
34
37
0 0 0 1 0 0
Program tra...
Free move!Player P1 expanded Node #9 with Evl: –2
New Board position is:
P#0
| 0 0 2 11 1 0 |
|24 23|
| 0 10 0 1 0 0 |
P#1
Player P0 expanded Node #3 with Evl: –26
New Board position is:


Welcome to Kalah Game!
MINMAX–AB vs MINMAX–AB
0 0 0 0 0 0
34
37
0 0 0 1 0 0
Program tra...
P#0
| 0 1 0 11 0 0 |
|25 25|
| 0 10 0 0 0 0 |
P#1
Free move!Player P0 expanded Node #2 with Evl: –6
New Board position is:
P#0


Welcome to Kalah Game!
MINMAX-AB vs MINMAX-AB
0 0 0 0 0 0
34
37
0 0 0 1 0 0
Program tra...
| 0 0 1 1 1 1 |
P#1

Player P1 expanded Node #8 with Evl: 0
New Board position is:

P#0
| 0 0 1 12 1 1 |
|27 27|
| 0 0 1 1 1 0 |
P#1


New Board position is:

P#0
| 1 1 1 0 1 1 |
|29 32|
| 1 1 2 1 1 0 |
P#1

Free move!Player P1 expanded Node #9 with Evl: 3
New Board position is:

```
Program tra...
--------------------------------------
Player P1 expanded Node #10 with Evl: 2
New Board position is:
--------------------------------------
P#0
| 0 0 0 12 0 0 |
|28         31|
| 0 0 1 0 0 0 |
P#1
--------------------------------------
Player P0 expanded Node #4 with Evl: -46
```

```
Player P0 expanded Node #4 with Evl: -46
New Board position is:
--------------------------------------
P#0
| 1 1 1 0 1 1 |
|29         31|
| 1 1 2 1 1 1 |
P#1
--------------------------------------
Player P1 expanded Node #8 with Evl: 1
New Board position is:
```

```
New Board position is:
--------------------------------------
P#0
| 1 1 1 0 1 0 |
|29         34|
| 1 1 2 1 0 0 |
P#1
--------------------------------------
Player P0 expanded Node #1 with Evl: -23
New Board position is:
--------------------------------------
```

```
--------------------------------------
P#0
| 0 1 1 0 1 0 |
|30         34|
| 1 1 2 1 0 0 |
P#1
--------------------------------------

Free move!Player P0 expanded Node #2 with Evl: -46
New Board position is:
--------------------------------------
```

```
P#0
| 0 0 1 0 1 0 |
|32         34|
| 0 1 2 1 0 0 |
P#1
--------------------------------------
Player P1 expanded Node #10 with Evl: 2
New Board position is:
--------------------------------------
P#0
| 0 0 1 0 0 0 |
```

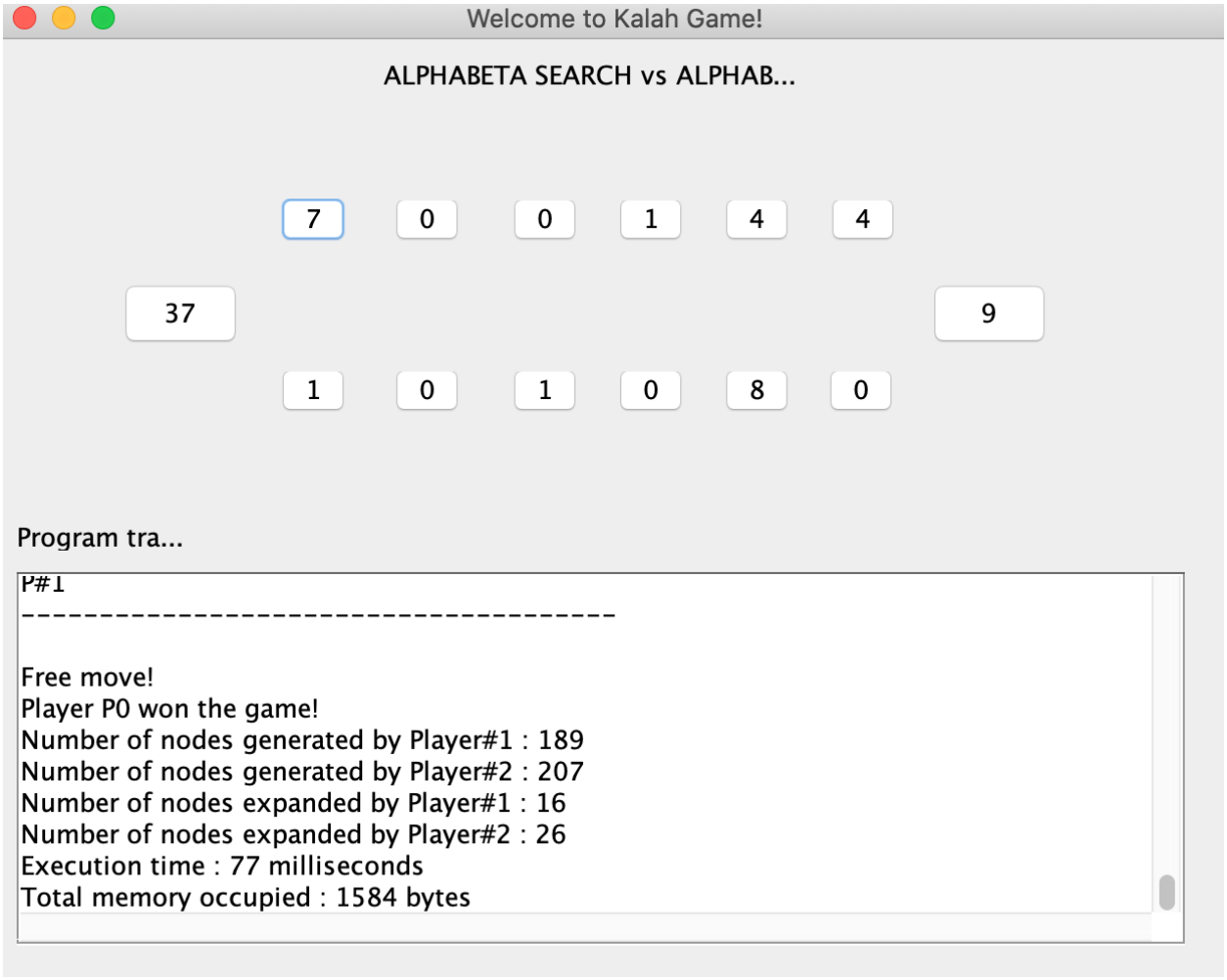
```
P#0
| 0 0 1 0 0 0 |
|32         36|
| 0 1 2 0 0 0 |
P#1
--------------------------------------
Player P0 expanded Node #3 with Evl: -46
New Board position is:
--------------------------------------
P#0
| 0 0 0 0 0 0 |
```

```
P#0
| 0 0 0 0 0 0 |
|34         36|
| 0 0 2 0 0 0 |
P#1
--------------------------------------
Player P1 expanded Node #11 with Evl: 0
New Board position is:
--------------------------------------
P#0
| 0 0 0 0 0 0 |
```

```
P#0
| 0 0 0 0 0 0 |
|34         36|
| 0 0 0 1 1 0 |
P#1
--------------------------------------
Player P0 expanded Node #11 with Evl: 0
New Board position is:
--------------------------------------
P#0
| 0 0 0 0 0 0 |
```

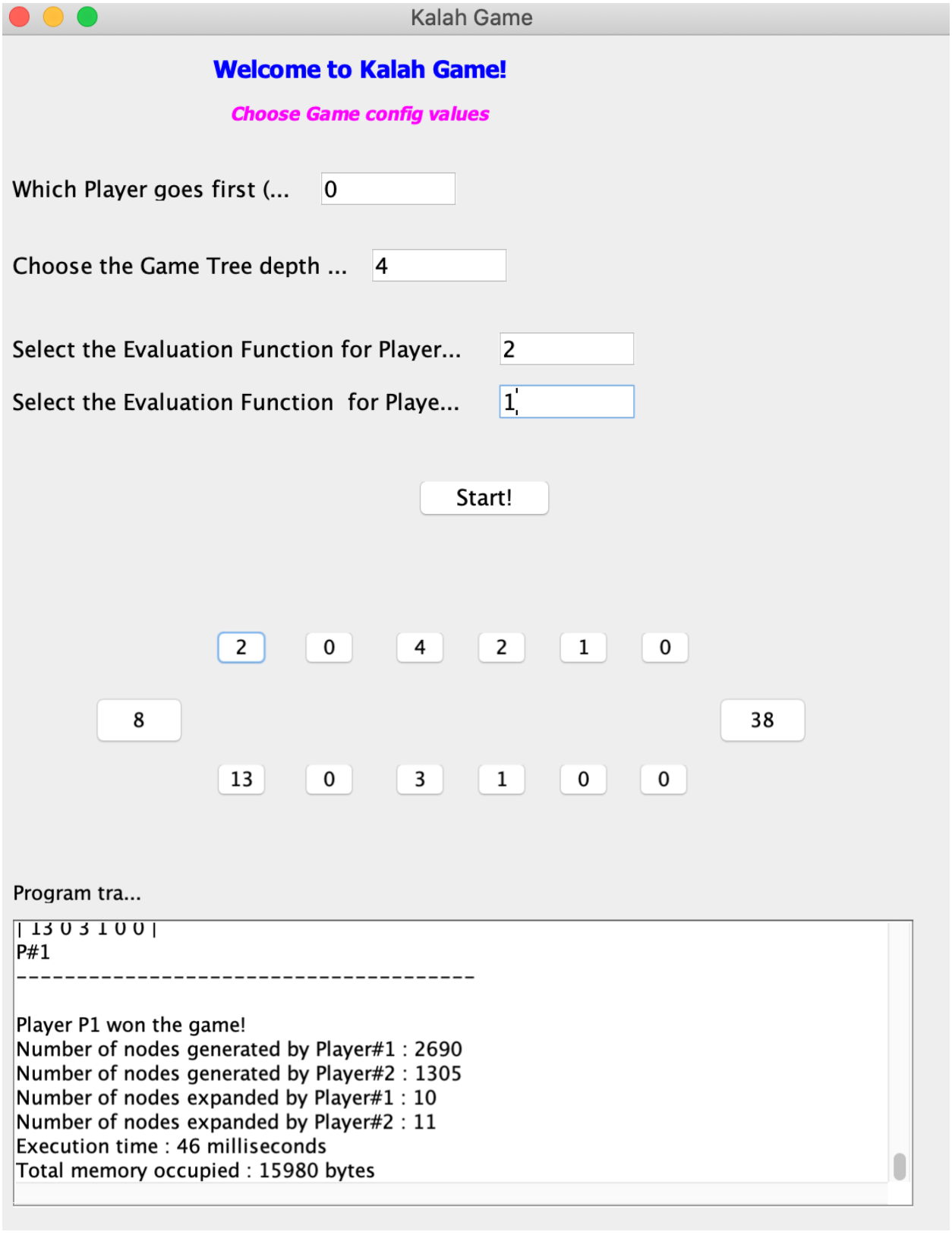
```
| 0 0 0 0 0 0 |
|34         36|
| 0 0 0 1 1 0 |
P#1
--------------------------------------
Player P1 expanded Node #9 with Evl: 0
New Board position is:
--------------------------------------
P#0
| 0 0 0 0 0 0 |
|34         37|
P#1
--------------------------------------

Free move!
Player P1 won the game!
Number of nodes generated by Player#1 : 425
Number of nodes generated by Player#2 : 467
Number of nodes expanded by Player#1 : 24
Number of nodes expanded by Player#2 : 23
Execution time : 69 milliseconds
Total memory occupied : 3568 bytes
```

2) ALPHA-BETA-SEARCH and ALPHA-BETA-SEARCH

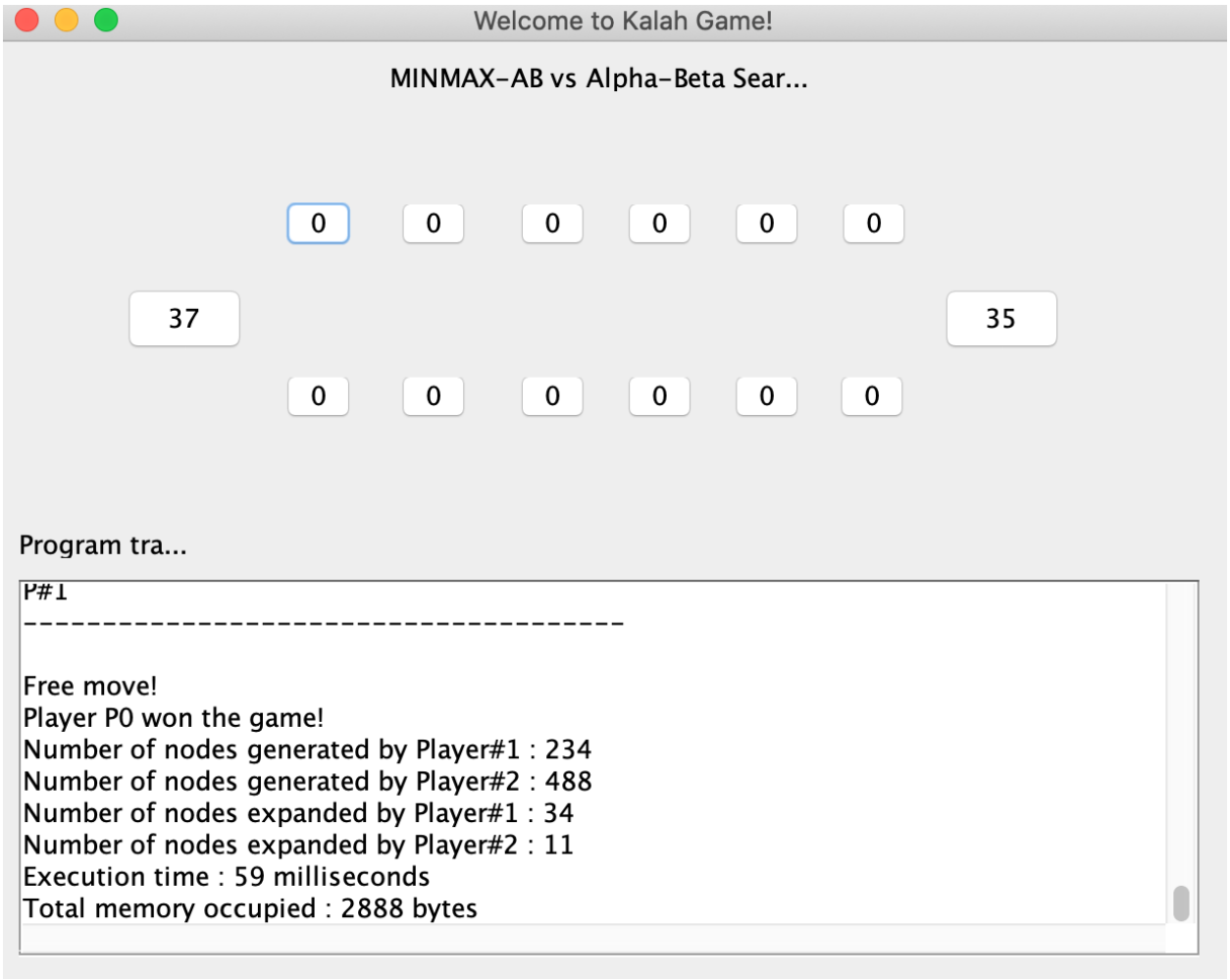

Welcome to Kalah Game!
ALPHABETA SEARCH vs ALPHAB...
7
0
0
1
4
4
37
9
1
0
1
0
8
0
Program tra...
P#1

Free move!
Player P0 won the game!
Number of nodes generated by Player#1 : 189
Number of nodes generated by Player#2 : 207
Number of nodes expanded by Player#1 : 16
Number of nodes expanded by Player#2 : 26
Execution time : 77 milliseconds
Total memory occupied : 1584 bytes

3) MINMAX-A-B and ALPHA-BETA-SEARCH



4) ALPHA-BETA-SEARCH and MINMAX-A-B



ANALYSIS OF THE PROGRAM

We initially came up with an idea of implementing both the algorithms and simple evaluation functions to check the correctness of the source code. Everything was working fine after some minor corrections. We then introduced little complex evaluation functions to make the decisions more accurate in making the moves. We decided to implement GUI functions to be more user friendly instead of just providing our inputs on the console.
We used different classes to be simpler and easier to understand code. With the help of the object oriented nature of the Java programming language, algorithms and functions have been isolated from the outside world with the abstraction feature.

Different functions have been implemented to avoid unnecessary repetition of code. We can just call the functions wherever same type of operations are repeating. Function calling has been achieved very beautifully at every step of the code. Every function is defined with purpose, input and return type hence increasing the readability of the code and explaining the purpose of the function then and there. Memory consumption is less when we used more functions as the memory will be released once the program comes out of the function. Hence, we achieve better memory management.

Evaluation functions are designed in such a way that it reduces the number of steps and is designed based on the total number of pieces available of their own Kalah, and opponent Kalah piece count.

The fancy function of this source code is GUI implementation. Instead of just making use of console inputs we used Graphical User Interface for user inputs to choose which algorithm shall be used for Kalah game play and player can choose different evaluation functions to play and we can use different depths. We can also choose which player can start the first move in the game.
Based on all these user inputs, source generates outputs and results will be displayed on the GUI.

We can input different player name, evaluation functions, algorithms and depths as per our requirements to test the working nature of our source code. The GUI is implemented very beautifully that inputting user data is very easy and output display is also user friendly.

Once we select our algorithms to play between computers, i.e MINMAX – A-B Vs MINMAX – A-B or MINMAX – A-B Vs ALPHABETA SEACRH or any other combinations, another GUI window will pop up to enter inputs for player number to start the game, depth, evaluation functions for computer 1 and 2. This feature of the source code is very beautiful. The results will be displayed on the GUI console as well as on the Eclpise editor console.
Initially we were changing the input values in the program for every run. Later we came to an idea of using GUI to provide input values to be more user friendly and which saves more time while running the program every time to verify for all the input combinations.

PROGRAM RESULT TABLES

CASE 1: **MINMAX - AB Vs MINMAX - AB**

a) Computer 1 chooses **evaluation function 1**.

Computer 2's evaluation functions	Depth	Number of Nodes Generated	Total Length of Path	Execution Time (in milliseconds)	The size of Memory used by the Program (in bytes)	Who Won?
evaluation function 1	2	Comp 1: 425 Comp 2: 467	Comp 1: 24 Comp 2: 23	57	3568	Player P1
evaluation function 2	2	Comp 1: 425 Comp 2: 467	Comp 1: 24 Comp 2: 23	65	3568	Player P1
evaluation function 3	2	Comp 1: 425 Comp 2: 467	Comp 1: 24 Comp 2: 23	66	3568	Player P1
evaluation function 1	4	Comp 1: 6539 Comp 2: 5729	Comp 1: 17 Comp 2: 15	77	49072	Player P0
evaluation function 2	4	Comp 1: 4291 Comp 2: 5313	Comp 1: 17 Comp 2: 15	63	38416	Player P0
evaluation function 3	4	Comp 1: 5912 Comp 2: 5557	Comp 1: 17 Comp 2: 15	71	45876	Player P0

b) Computer 1 chooses **evaluation function 2**.

Computer 2's evaluation functions	Depth	Number of Nodes Generated	Total Length of Path	Execution Time (in milliseconds)	The size of Memory used by the Program (in bytes)	Who Won?
evaluation function 1	2	Comp 1: 300 Comp 2: 333	Comp 1: 17 Comp 2: 15	44	2532	Player P0
evaluation function 2	2	Comp 1: 300 Comp 2: 333	Comp 1: 17 Comp 2: 15	44	2532	Player P0
evaluation function 3	2	Comp 1: 300 Comp 2: 333	Comp 1: 17 Comp 2: 15	47	2532	Player P0
evaluation function 1	4	Comp 1: 5990 Comp 2: 4071	Comp 1: 17 Comp 2: 15	57	40244	Player P0

evaluation function 2	4	Comp 1: 3742 Comp 2: 3655	Comp 1: 17 Comp 2: 15	61	29588	Player P0
evaluation function 3	4	Comp 1: 300 Comp 2: 333	Comp 1: 17 Comp 2: 15	68	2532	Player P0

c) Computer 1 chooses **evaluation function 3**.

Computer 2's evaluation functions	Depth	Number of Nodes Generated	Total Length of Path	Execution Time (in milliseconds)	The size of Memory used by the Program (in bytes)	Who Won?
evaluation function 1	2	Comp 1: 300 Comp 2: 333	Comp 1: 17 Comp 2: 15	42	2532	Player P0
evaluation function 2	2	Comp 1: 300 Comp 2: 333	Comp 1: 17 Comp 2: 15	45	2532	Player P0
evaluation function 3	2	Comp 1: 300 Comp 2: 333	Comp 1: 17 Comp 2: 15	43	2532	Player P0
evaluation function 1	4	Comp 1: 6624 Comp 2: 7386	Comp 1: 17 Comp 2: 15	61	56040	Player P0
evaluation function 2	4	Comp 1: 4376 Comp 2: 6970	Comp 1: 17 Comp 2: 15	70	45384	Player P0
evaluation function 3	4	Comp 1: 5997 Comp 2: 7214	Comp 1: 17 Comp 2: 15	64	52844	Player P0

CASE 2: **ALPHABETA SEARCH Vs ALPHABETA SEARCH**

a) Computer 1 chooses **evaluation function 1**.

Computer 2's evaluation functions	Depth	Number of Nodes Generated	Total Length of Path	Execution Time (in milliseconds)	The size of Memory used by the Program (in bytes)	Who Won?
evaluation function 1	2	Comp 1: 164 Comp 2: 219	Comp 1: 9 Comp 2: 33	70	1532	Player P0

evaluation function 2	2	Comp 1: 168 Comp 2: 253	Comp 1: 9 Comp 2: 37	65	1684	Player P0
evaluation function 3	2	Comp 1: 189 Comp 2: 207	Comp 1: 16 Comp 2: 26	24	1584	Player P0
evaluation function 1	4	Comp 1: 178 Comp 2: 252	Comp 1: 3 Comp 2: 22	50	1720	Player P1
evaluation function 2	4	Comp 1: 277 Comp 2: 370	Comp 1: 16 Comp 2: 29	25	2588	TIE
evaluation function 3	4	Comp 1: 368 Comp 2: 460	Comp 1: 15 Comp 2: 33	31	3312	Player P1

b) Computer 1 chooses **evaluation function 2**.

Computer 2's evaluation functions	Depth	Number of Nodes Generated	Total Length of Path	Execution Time (in milliseconds)	The size of Memory used by the Program (in bytes)	Who Won?
evaluation function 1	2	Comp 1: 168 Comp 2: 124	Comp 1: 23 Comp 2: 7	15	1168	Player P0
evaluation function 2	2	Comp 1: 215 Comp 2: 216	Comp 1: 26 Comp 2: 24	20	1724	Player P0
evaluation function 3	2	Comp 1: 147 Comp 2: 143	Comp 1: 17 Comp 2: 14	60	1160	Player P0
evaluation function 1	4	Comp 1: 380 Comp 2: 301	Comp 1: 30 Comp 2: 10	22	2724	Player P0
evaluation function 2	4	Comp 1: 220 Comp 2: 164	Comp 1: 14 Comp 2: 10	12	1536	Player P0
evaluation function 3	4	Comp 1: 244 Comp 2: 232	Comp 1: 12 Comp 2: 18	12	1904	Player P1

c) Computer 1 chooses **evaluation function 3**.

Computer 2's evaluation functions	Depth	Number of Nodes Generated	Total Length of Path	Execution Time (in milliseconds)	The size of Memory used by the Program (in bytes)	Who Won?
evaluation function 1	2	Comp 1: 188 Comp 2: 224	Comp 1: 17 Comp 2: 28	25	1648	Player P0
evaluation function 2	2	Comp 1: 210 Comp 2: 238	Comp 1: 17 Comp 2: 32	30	1792	Player P0
evaluation function 3	2	Comp 1: 182 Comp 2: 204	Comp 1: 13 Comp 2: 28	19	1544	Player P1
evaluation function 1	4	Comp 1: 112 Comp 2: 137	Comp 1: 5 Comp 2: 10	8	996	Player P1
evaluation function 2	4	Comp 1: 196 Comp 2: 215	Comp 1: 14 Comp 2: 11	12	1644	Player P1
evaluation function 3	4	Comp 1: 227 Comp 2: 247	Comp 1: 11 Comp 2: 18	10	1896	Player P0

CASE 3: **ALPHABETA SEARCH Vs MINMAX - AB**

a) Computer 1 chooses **evaluation function 1**.

Computer 2's evaluation functions	Depth	Number of Nodes Generated	Total Length of Path	Execution Time (in milliseconds)	The size of Memory used by the Program (in bytes)	Who Won?
evaluation function 1	2	Comp 1: 304 Comp 2: 285	Comp 1: 23 Comp 2: 23	56	2356	Player P0
evaluation function 2	2	Comp 1: 370 Comp 2: 492	Comp 1: 30 Comp 2: 25	70	3048	Player P0
evaluation function 3	2	Comp 1: 686 Comp 2: 328	Comp 1: 39 Comp 2: 36	85	4056	Player P0
evaluation function 1	4	Comp 1: 5096 Comp 2: 5440	Comp 1: 25 Comp 2: 26	87	42144	Player P1
evaluation function 2	4	Comp 1: 3325 Comp 2: 4652	Comp 1: 17 Comp 2: 16	73	31904	Player P0

evaluation function 3	4	Comp 1: 3933 Comp 2: 4828	Comp 1: 19 Comp 2: 20	76	35044	Player P1

b) Computer 1 chooses **evaluation function 2**.

Computer 2's evaluation functions	Depth	Number of Nodes Generated	Total Length of Path	Execution Time (in milliseconds)	The size of Memory used by the Program (in bytes)	Who Won?
evaluation function 1	2	Comp 1: 318 Comp 2: 321	Comp 1: 21 Comp 2: 20	62	2556	Player P0
evaluation function 2	2	Comp 1: 268 Comp 2: 167	Comp 1: 14 Comp 2: 17	50	1740	Player P1
evaluation function 3	2	Comp 1: 727 Comp 2: 441	Comp 1: 38 Comp 2: 40	94	4672	Player P1
evaluation function 1	4	Comp 1: 2690 Comp 2: 1305	Comp 1: 10 Comp 2: 11	43	15980	Player P1
evaluation function 2	4	Comp 1: 2650 Comp 2: 2560	Comp 1: 20 Comp 2: 22	70	20840	Player P0
evaluation function 3	4	Comp 1: 1948 Comp 2: 1374	Comp 1: 20 Comp 2: 20	70	13288	Player P1

c) Computer 1 chooses **evaluation function 3**.

Computer 2's evaluation functions	Depth	Number of Nodes Generated	Total Length of Path	Execution Time (in milliseconds)	The size of Memory used by the Program (in bytes)	Who Won?
evaluation function 1	2	Comp 1: 389 Comp 2: 546	Comp 1: 32 Comp 2: 34	86	3740	TIE
evaluation function 2	2	Comp 1: 240 Comp 2: 370	Comp 1: 19 Comp 2: 24	62	2440	Player P0
evaluation function 3	2	Comp 1: 385 Comp 2: 380	Comp 1: 23 Comp 2: 24	68	3060	Player P0

evaluation function 1	4	Comp 1: 2911 Comp 2: 2114	Comp 1: 13 Comp 2: 15	56	20100	Player P1
evaluation function 2	4	Comp 1: 2635 Comp 2: 4068	Comp 1: 17 Comp 2: 19	68	26812	Player P1
evaluation function 3	4	Comp 1: 3696 Comp 2: 2358	Comp 1: 11 Comp 2: 10	48	24216	Player P0

CASE 4: **MINMAX - AB Vs ALPHABETA SEARCH**

a) Computer 1 chooses **evaluation function 1**.

Computer 2's evaluation functions	Depth	Number of Nodes Generated	Total Length of Path	Execution Time (in milliseconds)	The size of Memory used by the Program (in bytes)	Who Won?
evaluation function 1	2	Comp 1: 478 Comp 2: 502	Comp 1: 33 Comp 2: 26	148	3920	Player P0
evaluation function 2	2	Comp 1: 314 Comp 2: 336	Comp 1: 21 Comp 2: 14	108	2600	Player P0
evaluation function 3	2	Comp 1: 562 Comp 2: 259	Comp 1: 8 Comp 2: 39	86	3284	Player P1
evaluation function 1	4	Comp 1: 3726 Comp 2: 4418	Comp 1: 21 Comp 2: 6	42	32576	Player P0
evaluation function 2	4	Comp 1: 2448 Comp 2: 3239	Comp 1: 14 Comp 2: 7	95	22748	Player P0
evaluation function 3	4	Comp 1: 4226 Comp 2: 3621	Comp 1: 10 Comp 2: 13	39	31388	Player P0

b) Computer 1 chooses **evaluation function 2**.

Computer 2's evaluation functions	Depth	Number of Nodes Generated	Total Length of Path	Execution Time (in milliseconds)	The size of Memory used by the Program (in bytes)	Who Won?
evaluation function 1	2	Comp 1: 250 Comp 2: 225	Comp 1: 13 Comp 2: 16	81	1900	Player P0

evaluation function 2	2	Comp 1: 378 Comp 2: 302	Comp 1: 15 Comp 2: 21	121	2720	Player P1
evaluation function 3	2	Comp 1: 239 Comp 2: 389	Comp 1: 18 Comp 2: 14	53	2512	Player P1
evaluation function 1	4	Comp 1: 5814 Comp 2: 2586	Comp 1: 9 Comp 2: 23	120	33600	Player P0
evaluation function 2	4	Comp 1: 1731 Comp 2: 2127	Comp 1: 13 Comp 2: 10	52	15432	Player P0
evaluation function 3	4	Comp 1: 6243 Comp 2: 2116	Comp 1: 4 Comp 2: 22	44	33436	Player P1

c) Computer 1 chooses **evaluation function 3**.

Computer 2's evaluation functions	Depth	Number of Nodes Generated	Total Length of Path	Execution Time (in milliseconds)	The size of Memory used by the Program (in bytes)	Who Won?
evaluation function 1	2	Comp 1: 234 Comp 2: 488	Comp 1: 34 Comp 2: 11	51	2888	Player P0
evaluation function 2	2	Comp 1: 173 Comp 2: 398	Comp 1: 21 Comp 2: 8	110	2284	Player P0
evaluation function 3	2	Comp 1: 239 Comp 2: 389	Comp 1: 18 Comp 2: 14	53	2512	Player P1
evaluation function 1	4	Comp 1: 6787 Comp 2: 3773	Comp 1: 18 Comp 2: 28	60	42240	Player P1
evaluation function 2	4	Comp 1: 1900 Comp 2: 3001	Comp 1: 15 Comp 2: 10	46	19604	Player P1
evaluation function 3	4	Comp 1: 6943 Comp 2: 2716	Comp 1: 14 Comp 2: 26	54	33436	Player P1

ANALYSIS OF THE TEST RESULTS

We ran our program for different algorithms, evaluation functions and depths of 2 and 4 to play the Kalah game. We tested our code for the different cases which are mentioned below,

CASE 1	MINMAX AB Vs MINMAX AB
CASE 2	ALHA-BETA SEARCH Vs ALHA-BETA SEARCH
CASE 3	ALHA-BETA SEARCH Vs MINMAX AB
CASE 4	MINMAX AB Vs ALHA-BETA SEARCH

Based on the results obtained by different runs for various algorithms and evaluation functions, we can observe that the player starting the game i.e, the one who is making is his first move after before opponent player, is having high probability of winning the game because this player can calculate his move before other player's move.

We can notice that total number of nodes generated are different for different depths chosen. More nodes have been generated when we chose the depth is 4 as compared to depth equal to 2 and also memory consumption is more in case of depth is equal to 4.

Based on the above result data tabulated in the tables, more memory is being consumed in case of depth 4 chosen. In case of MINMAX - AB Vs MINMAX – AB, more nodes have been generated as compared with the case of ALPHABETA SEARCH Vs ALPHABETA SEARCH.

The win/loss analysis has been tabulated below based on the results tables.

We can observe that evaluation functions 2 and 3 are performing better in several cases. And the performance of MINMAX – AB algorithm is better overall compared to ALPHABETA SEARCH algorithm.

In total MINMAX – AB algorithm won 18 times and ALPHABETA SEARCH algorithm won 17 times.

Hence, we can say that MINMAX – AB algorithm is more efficient than ALPHABETA SEARCH algorithm.

Win/Loss Statistics

Algorithm	Evaluation function	Winner
MINMAX - AB Vs MINMAX - AB	EV 1 Vs EV2 Vs EV3	EV 1 – 4 EV 2 - 7 EV 3 – 7 **EV 2 and EV3 are the winners**
ALPHABETA SEARCH Vs ALPHABETA SEARCH	EV 1 Vs EV2 Vs EV3	EV 1 – 5 EV 2 - 6 EV 3 – 6 TIE between EV 1 and EV 2 – 1 **EV 2 and EV3 are the winners**
ALPHABETA SEARCH Vs MINMAX - AB	EV 1, EV2 and EV3	ALPHABETA SEARCH – 9 MINMAX – AB – 8 TIE – 1 **ALPHABETA SEARCH is the winner**
MINMAX – AB Vs ALPHABETA SEARCH	EV 1, EV2 and EV3	MINMAX – AB – 10 ALPHABETA SEARCH – 8 **MINMAX – AB is the winner**

CONCLUSION

Based on the results tables and results analysis, we can conclude that MINMAX – AB algorithm is efficient compared to ALPHABETA SEARCH algorithm as it generates a smaller number of nodes and uses less memory to make any moves.

However, ALPHABETA SEARCH is almost winning equally as compared to MINMAX – AB's winning numbers. So, ALPHABETA SEARCH is intelligent.

As per the results data, ALPHABETA SEARCH algorithm can be used wherever winning probabilities shall be more and MINMAX – AB algorithm can be used in all the times where memory usage shall be less.

REFERENCES

[1] https://en.wikipedia.org/wiki/Kalah - Kalah Game history

[2] https://www.wikihow.com/Play-Kalaha - How to Play Kalah Game

[3] Dr. Moonis Ali, *Rich Chapter 12 Game Playing.pptx* - Lecture notes - Spring 2020

[4] Dr. Moonis Ali, *Luger Chapter 4 Two Player Games.pptx* - Lecture notes - Spring 2020

[5] Peter Norvig and Stuart J. Russell, *Artificial Intelligence: A Modern Approach: Third Edition*

[6] Elaine Rich Kevin Knight, *Artificial Intelligence*: *Third Edition*

[7] https://examples.javacodegeeks.com/desktop-java/swing/create-gui-java/ - How to create GUI with Java language