CS 5346: ADVANCED ARTIFICIAL INTELLIGENCE

KALAH GAME

IMPLEMENTATION OF KALAH GAME USING MIN-MAX-AB AND ALPHA-BETA-SEARCH ALGORITHMS

AUTHOR: VIDHYASHREE NAGABHUSHANA
NET ID: V_N63
STUDENT ID: A04927921

PROJECT BY:
JAIN, AKSHATHA MANOHAR
SUNDARA RAJ SREENATH, SAHANA
NAGABHUSHANA, VIDHYASHREE

TABLE OF CONTENTS

1 INTRODUCTION

1.1 THE PROBLEM DESCRIPTION

Games hold an inexplicable fascination for many people, and the notion that computers might play games has existed at least as long as computers. Games appear to be a good domain to explore Artificial Intelligence as they provide a structured task in which it is very easy to measure success or failure.

Two-person games are complicated, unlike single player games because of the existence of a hostile and essentially unpredictable opponent. Thus, they provide some interesting opportunities for developing heuristics, as weIl as greater difficulties in developing search algorithms. In 2 player games, strategy plays a very important role. Having said that, the probability of the win or lose of a player can be decided based on specific strategies and best possible moves that can lead to a win for the player. It becomes very important to choose and use certain search algorithms that can serve the purpose, at the same time, be efficient.

1.2 KALAH GAME

Mancala 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.

Kalah game is a variation of the Mancala game, one of the most popular variants in the western world. Kalah is played by two players on a board with two rows of 6 holes facing each other and two "kalahs" (the rightmost hole of each player). The stones are called "beans". At the beginning of the game, there are 6 beans in every hole. The kalahs are empty.

The object of Kalah is to get as many beans into your own kalah by distributing them. A player moves by taking all the beans in one of his 6 holes and distributing them counterclockwise, by putting one bean in every hole including his, but excluding the opponent's kalah. The game may start with a number of seeds in each house different from four. A nomenclature has been developed to describe these variations: Kalah(h,s), where h designates the number of houses on each side, and s designates the number of seeds that start out in each house.

In this project, we chose to focus on Kalah(6,6), i.e., each player has 6 houses and each house has 6 seeds at the start of the game. There are 2 players, 12 holes, 2 kalah’s and 72 stones/ beans.

The Kalah game is conceptually very simple, but is strategy based. The strategy of the Kalah Game can be very challenging. This very aspect makes it well suited for the analysis of the Min-Max AB and Alpha Beta Search algorithms.

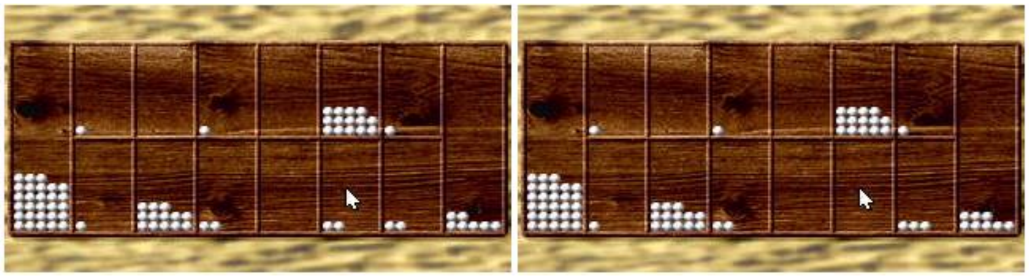
A glimpse of the Kalah Board initially is as below:

1.3 RULES OF KALAH GAME

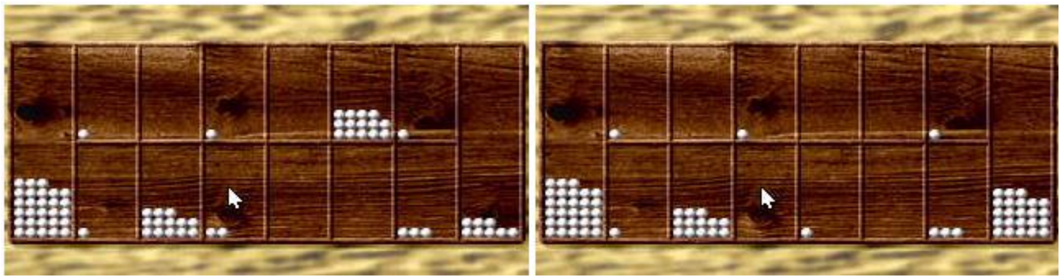
- Kalah is played by two players on a board with two rows of 6 holes facing each other.
- It has two KALAHS (Stores for each player).
- The stones are called beans. At the beginning of the game, there are 6 beans in every hole. The KALAHS are empty.
- The object of Kalah is to get as many beans into your own kalah by distributing them.
- A player moves by taking all the beans in one of his 6 holes and distributing them counterclockwise, by putting one bean in every hole including his, but excluding the opponent's kalah.

- **There are two special moves:**

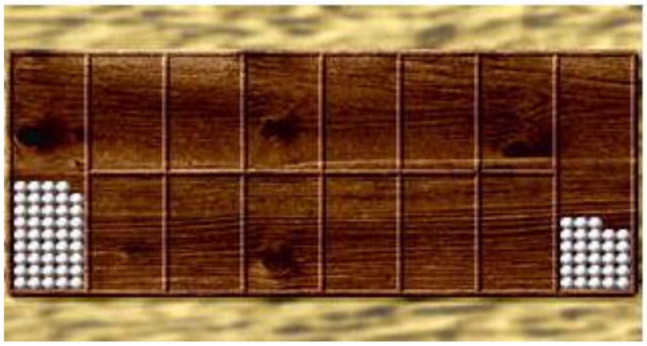
- **<u>Extra move:</u>** If the last bean is distributed into his own kalah, the player may move again. He has to move again even if he does not want to.



- **Capture:** If the last bean falls into an empty hole of the player and the opponent's hole above (or below) is not empty, the player puts his last bean and all the beans in his opponent's hole into his kalah. He has won all those beans.



- The game ends if all 6 holes of one player become empty (no matter if it is this player's move or not). The beans in the other player's holes are given into this player's kalah. The player who won more beans (at least 37) becomes the winner.



1.4 THE PROBLEM SOLUTION

As we have seen, the Kalah game is conceptually very simple, but is strategy based. The strategy of the Kalah Game can be very challenging. This very aspect makes it well suited for the analysis of the Min-Max AB and Alpha Beta Search algorithms. In this project, we deal with the different ways we could create a computer to computer Kalah game. We have based the game on MinMax-AB and Alpha-Beta Search algorithm and are trying to see which one yields better results. We have used three evaluation functions to get a better approximation of the values for the players that could be used to predict further moves. We use two depths, 2 and 4 and three evaluation functions. We mark the results from each of the combinations of the search algorithm, depth and evaluation function and draw conclusions about the performance of each algorithm. The results will be tabulated and presented for better understanding.

2 DISTRIBUTION OF WORK

2.1 TEAM MEMBERS

- Jain, Akshatha Manohar
- Sundara Raj Sreenath, Sahana
- Nagabhushana, Vidhyashree

2.2 CONTRIBUTIONS

AKSHATHA MANOHAR JAIN

- Each member of the team did extensive research on Kalah Game, the rules of the game, MINMAX-AB Algorithm, Alpha-Beta Search Algorithm and also, understood the Evaluation functions better.
- Helped in giving base idea about approaching the implementation during all the phases
- Implemented KalahFlow class
- Discussed and implemented the Classes Kalah and KalahFlow
- Created Evaluation function 2 and 3
- Did the unit testing of her modules and Integration testing after all the modules were integrated.
- As integration was challenging, each member worked to make sure every function did its job the right way.

VIDHYASHREE NAGABHUSHANA

- Each member of the team did extensive research on Kalah Game, the rules of the game, MINMAX-AB Algorithm, Alpha-Beta Search Algorithm and also, understood the Evaluation functions better.
- Helped in giving base idea about approaching the implementation during all the phases
- Discussed and implemented the Classes Kalah and KalahFlow
- Implemented MinMaxAB search algorithm.
- Implemented the main function
- Created evaluation function 1
- Did the unit testing of my modules and Integration testing after all the modules were integrated.
- As integration was challenging, each member worked to make sure every function did its job the right way.
- Added comments for each function to make it readable and understandable. The comments describe the functionality of each function, making it easier to get a gist of the functions.

SAHANA SREENATH

- Each member of the team did extensive research on Kalah Game, the rules of the game, MINMAX-AB Algorithm, Alpha-Beta Search Algorithm and also, understood the Evaluation functions better.
- Helped in giving base idea about approaching the implementation during all the phases
- Implemented AlphaBetaSearch algorithm.
- Discussed and implemented the Classes Kalah and KalahFlow
- Created evaluation function 4
- Did the unit testing of her modules and Integration testing after all the modules were integrated.
- As integration was challenging, each member worked to make sure every function did its job the right way.

3 METHODOLOGY

3.1 DIFFERENT PHASES

The project progressed as five stages of software development, Requirement Gathering phase, Design and Analysis phase, Implementation phase, Integration phase and Testing phase. Each of the phases are discussed below.

- **Requirement Gathering Phase:**

Requirement gathering phase is the initial phase. In this phase, we discussed about the project question, discussed amongst ourselves about our understanding of what was to be done as a part of the project and how we would proceed in doing it.

Each member of the team did extensive research on Kalah Game, the rules of the game, MINMAX-AB Algorithm, Alpha-Beta Search Algorithm and also, understood the Evaluation functions better as it is very important to understand these in the initial steps before the implementation is done.

- **Design and Analysis Phase:**

After the requirement gathering phase, we progressed to the Design and Analysis phase. In this phase, Object Oriented Programming Style was finalized for the project. Accordingly, the different classes that would help building the project were discussed. The functionalities to fit into each class, the data members, the member functions to be used for various purposes, object creation criteria were discussed and finalized. Separate procedures for MINMAX-AB and Alpha-Beta Search were finalized for better reusability. The Move generator functions were formulated for both the Players. The next move for each player was decided based on the rules of the game. We had a clear picture about the basic idea of implementation of the project, the classes that we would need, the member functions and their uses. This helped us proceed to the next phase.

- **Implementation Phase:**

  After a thorough analysis of the project requirements and having a design idea ready, we proceeded to the implementation phase. This is one of the most important phases as this phase is the core of the Project. In this phase, the classes were implemented. MinMax-AB and Alpha Beta Search algorithms were implemented. As mentioned earlier, Object oriented approach was followed, making it easier to build the connectivity between the objects and methods.

- **Integration Phase:**

  After the Implementation phase, all the necessary Classes were ready, the MinMax-AB and Alpha-Beta Search Algorithm implementations were ready. The integration phase acts as a very important phase as the code can break at any point during integration. A main function was used as a driver to integrate all the classes and procedures. Challenging phase in the project implementation was the integration as a lot of errors cropped up after integrating the different modules and these errors had to be fixed.

- **Testing Phase:**

  The final phase in the project implementation was the testing phase. Often neglected, yet the most important testing phase was when we could realize a few mistakes that had not come into light in the initial stages and were identified. We tried testing the code multiple times with three different evaluation functions, 2 depths, 2 algorithms. Results were noted down each time to make sure the program was not breaking. Each run was tested for the display, the correct player moves, to see if a player eligible was given extra moves or not, also, if the capture of opponent's beans and his own happened as and when the players last bean fell in an empty house of his. The number of nodes generated, the path length, the total time were monitored in each run to make sure the correct results were displayed.

3.2 MINMAX-AB ALGORITHM

Ever since the advent of Artificial Intelligence (AI), game playing has been one of the most interesting applications of AI. MinMax is a recursive algorithm which is used to choose an optimal move for a player assuming that the other player is also playing optimally. It is similar to how we think when we play a game: "if I make this move, then my opponent can only make only these moves," and so on. Minimax is called so because it helps in minimizing the loss when the other player chooses the strategy having the maximum loss. The MinMax procedure is very simple. But its performance can be improved significantly with a few refinements. A modified strategy is used, which is the MinMax Algorithm with

Alpha Beta Pruning. If we apply alpha-beta pruning to a standard minimax algorithm, it returns the same move as the standard one, but it removes (prunes) all the nodes that are possibly not affecting the final decision.

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

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

For each clement SUCC of SUCCESSORS:

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

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

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

5. Return the structure
VALUE = Pass-Thresh
PATH = BEST-PATH

3.3 ALPHA BETA SEARCH

The problem with minimax search is that the number of game states it has to examine is exponential in the depth of the tree. Alpha–beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minmax algorithm in its search tree. It is an adversarial search algorithm used commonly for machine playing of two-player games It stops evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. 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.

The Alpha-Beta Search Algorithm is as shown below:

```
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
```

4 EVALUATION FUNCTION

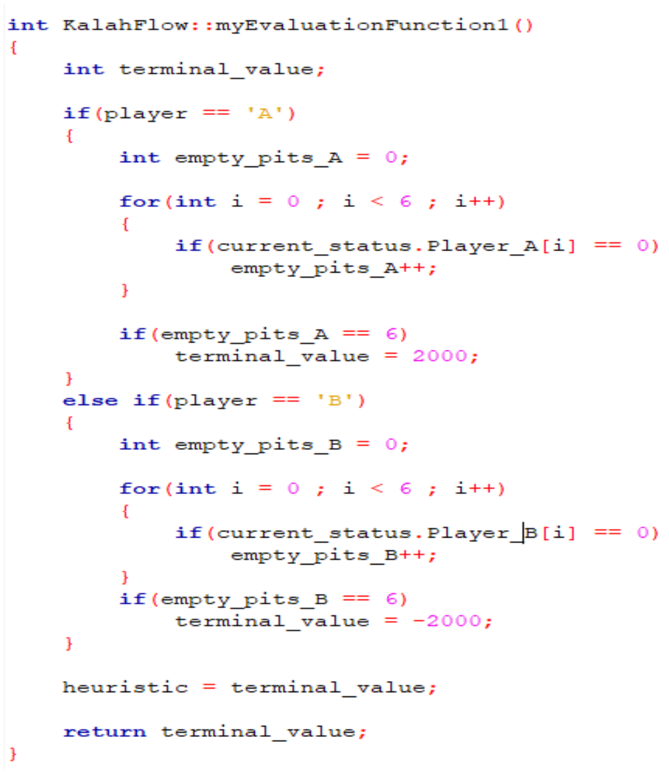
An evaluation function, also known as a heuristic evaluation function or static evaluation function, is a function used by game-playing computer programs to estimate the value or goodness of a position (usually at a leaf or terminal node) in a game tree. A tree of such evaluations is usually part of a minmax search paradigm which returns a particular node and its evaluation as a result of alternately selecting the most favorable move for the side on move at each ply of the game tree. An evaluation function returns an estimate of the expected utility of the game from a given position. It should be clear that the performance of a game-playing program depends strongly on the quality of its evaluation function. An inaccurate evaluation function will guide an agent toward positions that turn out to be lost. A good evaluation is designed as below:

- The evaluation function should order the terminal states in the same way as the true utility function, i.e., states that are wins must evaluate better than draws, which in turn must be better than losses.
- The computation must not take too long.
- For nonterminal states, the evaluation function should be strongly correlated with the actual chances of winning.

  Below are the evaluation functions used for the project:

EVALUATION FUNCTION 1

In the below evaluation function, we have considered the case where the pits of A or B are empty. If all pits in A is 0, we assign a value of 2000 to the terminal node. If all pits in B are empty, we assign -2000 to the terminal node. We are maximizing the chances of A winning and minimizing the chances of B winning.

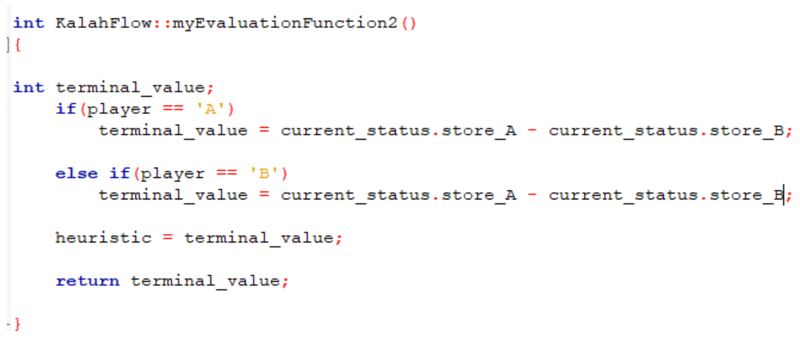


```
int KalahFlow::myEvaluationFunction1()
{
    int terminal_value;

    if(player == 'A')
    {
        int empty_pits_A = 0;

        for(int i = 0 ; i < 6 ; i++)
        {
            if(current_status.Player_A[i] == 0)
                empty_pits_A++;
        }

        if(empty_pits_A == 6)
            terminal_value = 2000;
    }
    else if(player == 'B')
    {
        int empty_pits_B = 0;

        for(int i = 0 ; i < 6 ; i++)
        {
            if(current_status.Player_B[i] == 0)
                empty_pits_B++;
        }
        if(empty_pits_B == 6)
            terminal_value = -2000;
    }

    heuristic = terminal_value;

    return terminal_value;
}
```

\

EVALUATION FUNCTION 2

This evaluation function checks whether Player A or player B is playing. Then it will calculate the terminal value for the respective players. Terminal value is the amount of difference between stones at store_A(PlayerA's total stones in the storeA till this move) and store_B(PlayerB's total stones in the storeB till this move). As our program always 1st gives chace to Player A, It is always guareented that PlayerA or the Player who plays 1st will win the game according to the algorithms. So, we assumed Player A's store will consists of more number of stones So we calculated a difference assuming that heuristic will get the best value. In this way heuristic will get the maximum possible value. Function returns the terminal value which is an interger.
Evaluation functions utilizes linear time complexity of O(1).



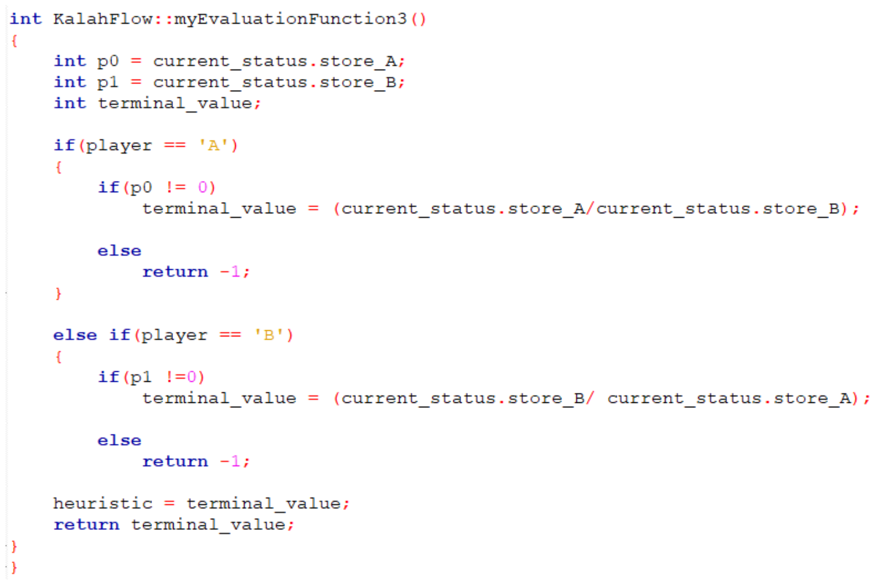
```
int KalahFlow::myEvaluationFunction2()
{

int terminal_value;
    if(player == 'A')
        terminal_value = current_status.store_A - current_status.store_B;

    else if(player == 'B')
        terminal_value = current_status.store_A - current_status.store_B;

    heuristic = terminal_value;

    return terminal_value;

}
```

EVALUATION FUNCTION 3

This evaluation function checks whether Player A or player B is playing. It has 2 Interger variables

1. P0 : which gets the number of stones at store_A
2. P1: which gets the number of stones at store_B

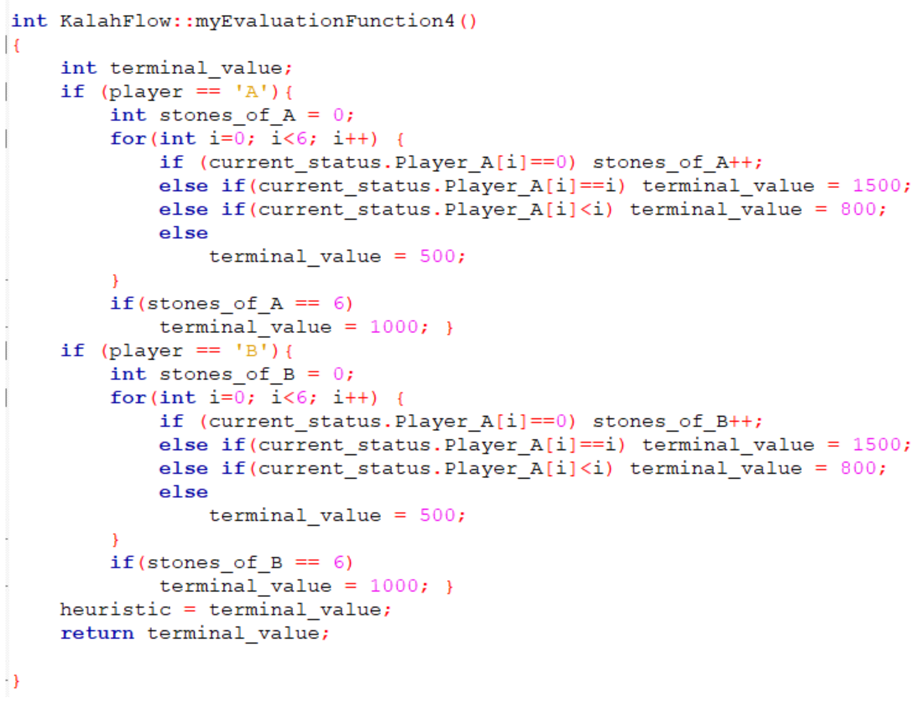
Function will check whether the po or p1 is not equal to zero(To explain this mathematically if the value of po or p1 is equal to 0. Then 0 divide by anything will be zero. So, zero cannot be the best value or maximum value. If this condition is satisfied it calculates the terminal value. Otherwise the evaluation function returns -1 and exits. Evaluation functions utilizes linear time complexity of O(1).



```
int KalahFlow::myEvaluationFunction3()
{
    int p0 = current_status.store_A;
    int p1 = current_status.store_B;
    int terminal_value;

    if(player == 'A')
    {
        if(p0 != 0)
            terminal_value = (current_status.store_A/current_status.store_B);

        else
            return -1;
    }

    else if(player == 'B')
    {
        if(p1 !=0)
            terminal_value = (current_status.store_B/ current_status.store_A);

        else
            return -1;

    heuristic = terminal_value;
    return terminal_value;
}
}
```

EVALUATION FUNCTION 4

The below evaluation function evaluates the current board, calculates efficiently and always returns the same value for the same board. Terminal node values are given precisely – maximal positive value for max player and maximal negative value to the opponent player. It also correlates the score for the current player to win with best moves – assuming the max player will not make any wrong moves or mistakes to let the opponent win(Ideal Situation)

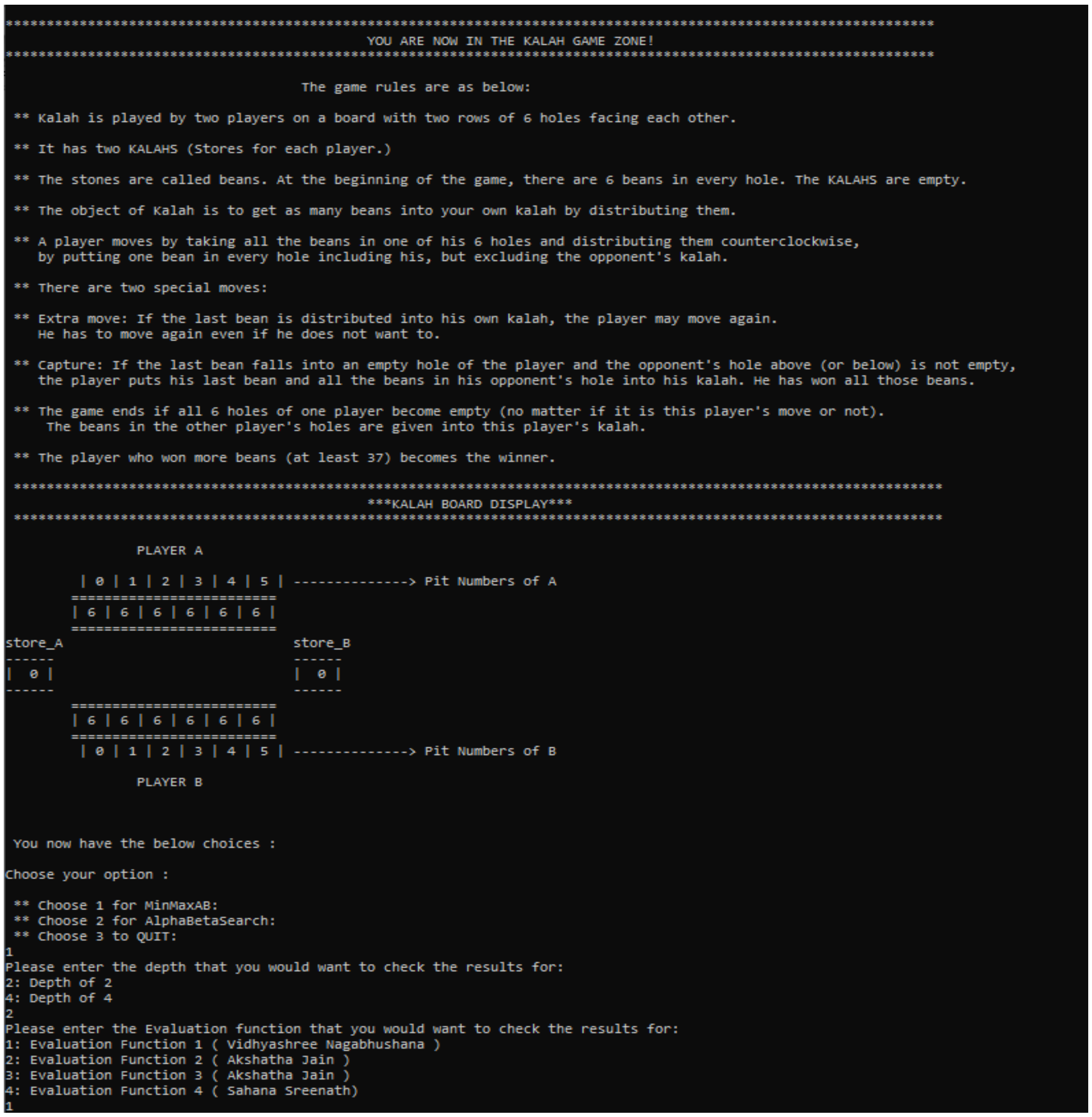


```
int KalahFlow::myEvaluationFunction4()
{
    int terminal_value;
    if (player == 'A'){
        int stones_of_A = 0;
        for(int i=0; i<6; i++) {
            if (current_status.Player_A[i]==0) stones_of_A++;
            else if(current_status.Player_A[i]==i) terminal_value = 1500;
            else if(current_status.Player_A[i]<i) terminal_value = 800;
            else
                terminal_value = 500;
        }
        if(stones_of_A == 6)
            terminal_value = 1000; }
    if (player == 'B'){
        int stones_of_B = 0;
        for(int i=0; i<6; i++) {
            if (current_status.Player_A[i]==0) stones_of_B++;
            else if(current_status.Player_A[i]==i) terminal_value = 1500;
            else if(current_status.Player_A[i]<i) terminal_value = 800;
            else
                terminal_value = 500;
        }
        if(stones_of_B == 6)
            terminal_value = 1000; }
    heuristic = terminal_value;
    return terminal_value;

}
```

5 SOURCE CODE IMPLEMENTATION

5.1 Main Function

The program begins with the execution of the main function. The control of the program begins with displaying the rules of the Kalah Game after which, the initial board is displayed for the player to get an idea about the game. Later, menu options are placed for the user to select from three choices, MinMaxAB, AlphaBetaSearch and to Quit the program. Once this is chosen, the user can select a depth of 2 or 4. The user is also given an option to select from the 4 available Evaluation function options. Below is a snapshot of the display screen initially:



```
*************************************************************************************************************
                                            YOU ARE NOW IN THE KALAH GAME ZONE!
*************************************************************************************************************

                                    The game rules are as below:

 ** Kalah is played by two players on a board with two rows of 6 holes facing each other.

 ** It has two KALAHS (Stores for each player.)

 ** The stones are called beans. At the beginning of the game, there are 6 beans in every hole. The KALAHS are empty.

 ** The object of Kalah is to get as many beans into your own kalah by distributing them.

 ** A player moves by taking all the beans in one of his 6 holes and distributing them counterclockwise,
    by putting one bean in every hole including his, but excluding the opponent's kalah.

 ** There are two special moves:

 ** Extra move: If the last bean is distributed into his own kalah, the player may move again.
    He has to move again even if he does not want to.

 ** Capture: If the last bean falls into an empty hole of the player and the opponent's hole above (or below) is not empty,
    the player puts his last bean and all the beans in his opponent's hole into his kalah. He has won all those beans.

 ** The game ends if all 6 holes of one player become empty (no matter if it is this player's move or not).
     The beans in the other player's holes are given into this player's kalah.

 ** The player who won more beans (at least 37) becomes the winner.

 *************************************************************************************************************
                                            ***KALAH BOARD DISPLAY***
 *************************************************************************************************************

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 6 | 6 | 6 | 6 | 6 | 6 |
        =========================
store_A                            store_B
------                             ------
|  0 |                             |  0 |
------                             ------
        =========================
        | 6 | 6 | 6 | 6 | 6 | 6 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


 You now have the below choices :

Choose your option :

 ** Choose 1 for MinMaxAB:
 ** Choose 2 for AlphaBetaSearch:
 ** Choose 3 to QUIT:
1
Please enter the depth that you would want to check the results for:
2: Depth of 2
4: Depth of 4
2
Please enter the Evaluation function that you would want to check the results for:
1: Evaluation Function 1 ( Vidhyashree Nagabhushana )
2: Evaluation Function 2 ( Akshatha Jain )
3: Evaluation Function 3 ( Akshatha Jain )
4: Evaluation Function 4 ( Sahana Sreenath)
1
```

Once the user selects the necessary options for the algorithm, the depth and the evaluation function choice, the necessary switch condition is executed.

Switch Case 1: MinMaxAB algorithm selection:

We initially start with assigning the player as ‘A’ to be the player that makes the first move. The clock time is started to note the time that the MinMaxAB algorithm takes.
We check if the result is ‘N’, i.e., neither A nor B has won the game. Until the result is ‘N’, is true, we perform repetitive steps.

In the loop that continues until the result is ‘N’, we create an object of type KalahFlow and copy the current player status into the object. We call the MinMaxAB function with the parameters : the object of the KalahFlow class, 0 as the initial depth, Player A as the initial player, Use Threshold as 2000 and Pass Threshold as -2000, selected depth of 2 if the user chooses to run the program with a depth 2 and 4 if the user chooses 4, the choice of the evaluation function.

Once the MinMaxAB function is called, the heuristic value is set in the function. We get the value of the heuristic and store it to call the move_generator function for A if the player is A and for B if player is B and pass the heuristic value as the pit from where the player has to make a move. The result is stored. We meanwhile keep a track of the game path length and increment it each time the control enters the loop while.

Once A or B is returned as the result, we stop the clock, compute the time, memory requirements and print the result for the user.

Switch Case 2: AlphaBetaSearch algorithm selection:

We initially start with assigning the player as ‘A’ to be the player that makes the first move. The clock time is started to note the time that the AlphaBetaSearch algorithm takes.
We check if the result is ‘N’, i.e., neither A nor B has won the game. Until the result is ‘N’, is true, we perform repetitive steps.

In the loop that continues until the result is ‘N’, we create an object of type KalahFlow and copy the current player status into the object. We call the AlphaBetaSearch function with the parameters : the object of the KalahFlow class, 0 as the initial depth, Player A as the initial player, alpha as 2000 and beta as -2000, selected depth of 2 if the user chooses to run the program with a depth 2 and 4 if the user chooses 4, the choice of the evaluation function.

Once the function is called, the heuristic value is set in the function. We get the value of the heuristic and store it to call the move_generator function for A if the player is A and for B if player is B and pass the heuristic value as the pit from where the player has to make a move. The result is stored. We meanwhile keep a track of the game path length and increment it each time the control enters the loop while.

Once A or B is returned as the result, we stop the clock, compute the time, memory requirements and print the result for the user.

Switch Case 3: QUIT the program

When user chooses this option, the control is returned.

5.2 Class Kalah

Class Kalah gives details about the representation of the Kalah Board. We have considered two players, Player_A and Player_B as an array of integers that can hold 6 values. Each array element Player_A[i] and Player_B[i] correspond to the pit of the players. Each player has 6 pits. Initially, we have initialized all the pits of Players A and B to 6, i.e., each pit has 6 stones in them. That is the initial board representation of the Kalah game. We have two stores, one for player A and one for player B. Initially, both the stores will be empty. As and when the game progresses, the stores are updated. In the class, we have constructors to initialize the data members. A default constructor, which initializes an object of type Kalah when created and a copy constructor. Copy constructor is used here to initialize one object with the values of other object of the same type Kalah. We have a display() function that is used to display the board each time. We have called the display function in multiple places in the program to show the step by step game procedure. We have a function to overload the '=' operator to compare two objects of type Kalah. We have move generator functions for both players. We also have defined a result function which decides as to which player has to win the game based on the current board status and the stone counts in each of the player's store.

Move_generator() function:

We have considered two move_generator functions. Move_generator_A() function helps to generate moves for Player A and move_generator_B() function helps to generate moves for Player B. The move generator functions based on the rules of the game, as follows:
Initially, we consider the current position from where the player starts moving stones. The current position is obtained from the function parameter hole_num. Once we get the hole/pit number, we check for the number of stones in that pit. As long as there are 1 or

more than one stones in the pit from where the player has to start the game, we continue with deciding what move the player has to make. We consider two special moves in the move generator function:

- If the last stone lands in the player’s store, the player gets another chance.
- If a player’s stone lands in an empty hole in its own set of pits, the player captures all the opponent player's stone from the opposite hole.
- If the above two special scenarios are not encountered, the moves are normal, meaning, player moves the stones from the pit where he starts to other pits counterclockwise.

The move_generator function works in a similar manner for player A and player B.

Below is a code snippet from the move_generator_B(hole_num) function that specifies the move for player B when last stone lands in B’s store:

```
while(stoneCount > 0)
{
    //last stone lands in Player B's store
    if(stoneCount >= 1 && current_Position == 6)
    {
        stoneCount--;
        store_B++;

        if(stoneCount == 0)
            return 'B'; // giving B another chance as last stone falls in B's store
    }
```

Result() function:

The result function is defined to return if player A has won or player B has won. It considers certain conditions to return the result as mentioned below:
It keeps a track of the number of empty pits of A and B. If A has no more stones to play, i.e., if A has 6 empty pits, we collect all the remaining stones in Player B’s pits to the store of Player B. If B has no more stones to play, i.e., if Player B has 6 empty pits, we collect all the remaining stones in Player A’s pits to the store of Player A. We now compare to see if store of Player A has more than 36 stones or if store of Player B has more than 36 stones. If Player A has more stones in its store, player A is returned as the winner. If Player B has more stones in its store, player B is returned as the winner.

Below is a code snippet from the function result() that returns if the winner is A or B based on the number of stones in the store of each player:

```
//if Player A has more than 36 stones, Player A wins
if(store_A > maxWinNumber)
    return 'A';

//if Player B has more than 36 stones, Player B wins
else if(store_B > maxWinNumber)
    return 'B';

else
    return 'N';
```

5.3 Class KalahFlow

We have implemented Class KalahFlow to deal with creating successors, to check if it is deep enough for the algorithms to return the actual best value, set heuristic value and to keep a track of the successor count. The data members of the class are player, an object of type Kalah which stores the current status of the game, heuristic value, successors and the count of how many successors were generated. The variable heuristic takes care of updating the heuristic value as and when it is changed during the program execution. A parameterized constructor is used to initialize the data members of the class. Initially, the player is initialized as ‘A’ and the successors are initialized to NULL.

The **current_player_status()** function takes an object of type Kalah and copies the current status of the player into the current_status object of type Kalah in the KalahFlow class.

The **is_deep_enough()** function checks if the depth is 2 or 4, which gives true for is_deep_enough for the algorithms. If it is not deep enough, we generate one more ply of the tree by calling the generate_successors() function. Below is a code snippet of the is_deep_enough() function:

```
bool KalahFlow::is_deep_enough(int depth, int selected)
{
    if(heuristic != -2000)
        return heuristic;

    if(depth >= selected || current_status.result() != 'N' )
        return true;

    else
    {
        expanded_node_count++;
        generate_successors();
        return false;
    }
}
```

Here, depth is the current depth and selected is the maximum depth that the user chooses, it can be 2 or 4 based on the user input.

The **generate_successors()** function generates successors(child node) when the deep enough function is satisfied to evaluate the future moves of the opponent based on the present state of the board. This root node has child branches which ends in the heuristic value of the end leaf node. Here the leaf node is the value of number of stones in the pit at a given board state.

The class KalahFlow is where the evaluation functions discussed above are defined too. The evaluation function sets the value of the terminal node, the heuristic value.

5.4 MINMAX-AB Function

One of the main functions of this program implementation is the MinMaxAB function. The minmax search procedure is a depth- first, depth-limited search procedure. We have defined the MinMaxAB function with the following parameters: an object of type KalahFlow, the current depth which is 0 during the initial call, the current player, which was ‘A’ initially, Use-Threshold, Pass-Threshold value, selectedDepth, which can be 2 or 4, the maximum depth, evaluation_choice-1, 2 or 3 – to select from three possible evaluation function options.

The idea is to start at the current position and use the plausible-move generator to generate the set of possible successor positions. We can apply the static evaluation function to those positions and simply choose the best one. The MinMax algorithm creates all possible states of the player in the game. This is the depth of the recursion. At the lowest level is the leaf or the terminal node which has a value, the value obtained by the relevant evaluation function. When applied to a standard minmax tree, it prunes away branches that cannot possibly influence the final decision. MINMAX-A-B uses two values, USE-THRESH and PASS-THRESH. USE-THRESH is used to compute cutoffs. PASS-THRESH is merely passed to the next level as its USE-THRESH. USE-THRESH must also be passed to the next level, but it will be passed as PASS-THRESH so that it can be passed to the third level down as USE-THRESH again, and so forth. The thresholds must be negated just like the values are negated each time it is passed. This is necessary so that, regardless of the level of the search, a test for greater than will determine whether a threshold has been crossed. If the algorithm reaches the terminal node, the evaluation function is calculated and the value is passed up to the root node. The opposite is also true if the root node is a min player. The premise of Minmax algorithm is for the system to calculate the next best move by evaluating the evaluation function of the next possible moves for both Min and max in the next future turns. We always assume the system should consider that the opponent will always choose the best possible move, minimizing the other player's chances to win. We have recursive function calls to check if its min or max turn. The algorithm will first check the configuration of the board if it is a terminal node or if it has reached the depth limit. If the condition is false it will calculate the Utility Function and set min and max value and check if utility is greater than or less than the current value. The recursive loop breaks only if someone has won the game or the depth limit is reached.

5.5 AlphaBetaSearch function

Alpha–beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minmax algorithm in its search tree. It is an adversarial search algorithm used commonly for machine playing of two-player games It stops evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. 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. We have defined the AlphaBetaSearch function with the following parameters: an object of type KalahFlow, the current depth which is 0 during the initial call to the function, the player, which is initially 'A', alpha, beta, selected depth, which can be 2 or 4, the maximum depth, evaluation_choice-1, 2 or 3 – to select from three possible evaluation function options.
Alpha is updated during MAX's turn which checks the terminal node value. Similarly, beta can be updated only when it's MIN's turn. The best value for both will change according to the terminal value when compared to the present best value. The value is passed to the root node for both alpha and beta. We also check for condition if alpha > beta for pruning. We compare the alpha and beta value with best value as below, as if the condition satisfies set the new value as pass threshold and use threshold value. This helps to prune a node if already we have a best value before traversal to the other child node.

6 SOURCE CODE

```
#include<ctime>
#include <stdio.h>
#include <iostream>
#include <iomanip>
#include<ctime>
#include<cstdlib>

using namespace std;

int total_nodes,expanded_node_count,game_path_length;
const int SIZE = 6;

class Kalah
{
    public:
    int Player_A[SIZE],
        Player_B[SIZE],
        store_A,
        store_B,
        slots,
        maxWinNumber;

    int *pointer;

    Kalah();
    Kalah(Kalah *);
    void display();
    void operator=(Kalah);
    char move_decider(int, char);
    char move_generator_A(int);
    char move_generator_B(int);
    char result();

};

/*******************************************************************
***********
Kalah(): A default constructor that initializes all the data members
of the Kalah Class. Each pit of Player A and Player B are
initialized to having 6 stones stores of both players initialized to
0
max number of stones that a person needs to be considered as having
a winning chance is assigned to 36
```

```
parameters used: no
returns : constructor does not have a return type.
*******************************************************************
***********/

Kalah::Kalah()
{
    for(int i = 0 ; i < SIZE; i++)
    {
        Player_A[i] = 6;
        Player_B[i] = 6;
    }

    store_A = store_B = 0;
    slots = 5;

    pointer = NULL;
    maxWinNumber = 36;
}


/******************************************************************
***********
Kalah(): A copy constructor that is used to initialize an object of
class Kalah with another object. Copies the kalah value from one
object to another
parameters used: no
returns : constructor does not have a return type.
*******************************************************************
***********/

Kalah::Kalah(Kalah *b)
{
    for(int i = 0; i < SIZE; i++)
    {
        this->Player_A[i] = b->Player_A[i];
        this->Player_B[i] = b->Player_B[i];
    }

    this->store_A = b->store_A;
    this->store_B = b->store_B;
}
```

```
/*******************************************************************
***********
dislay(): This function displays the Kalah board
parameters used: No parameters used
returns : Returns no values
********************************************************************
***********/

void Kalah::display()
{
    cout << endl;
    cout << "\t\tPLAYER A";
    cout << endl << endl;
    cout << " \t | ";

    for(int i = 0 ; i < SIZE; i++)
        cout  << i << setw(2) << "|" << " ";

    cout << "--------------> Pit Numbers of A" << endl;
    cout << "\t=========================";
    cout << endl;
    cout << "\t" << "|";
    cout << setw(2);


    for(int i = (SIZE - 1) ; i >= 0 ; i--)
        cout << Player_A[i] << setw(2) << "|" << " ";

    cout << endl;
    cout << "\t=========================";

    cout << endl;
    cout << "store_A" << "\t\t\t\t   store_B" <<endl;
    cout << "------"  << "\t\t\t\t   ------";
    cout << endl;
    cout <<"|" << setw(3) << store_A << " |"; // Fourth Line
    cout <<"\t\t\t\t   |" << setw(3) << store_B << " |" << endl;
    cout << "------"  << "\t\t\t\t   ------";
    cout << endl;
    cout << "\t=========================";
    cout << endl;
    cout << "\t" << "|";
    cout << setw(2);

    for(int i = 0 ; i < SIZE; i++)
```

```
        cout << Player_B[i] << setw(2) << "|" << " ";

    cout << endl;
    cout << "\t========================";
    cout << endl;
    cout << " \t | ";

     for(int i = 0 ; i < SIZE; i++)
        cout  << i << setw(2) << "|" << " ";

    cout << "---------------> Pit Numbers of B";
    cout << endl << endl;
    cout << "\t\tPLAYER B";
    cout << endl << endl << endl << endl;
}

/*******************************************************************
***********
operater=(Kalah k): This is an operator overloading function that
overloads the '='
parameters used: Object of type Kalah
returns : Returns no values
********************************************************************
***********/

void Kalah::operator=(Kalah k)
{
    for(int i = 0; i < SIZE; i++)
    {
        Player_A[i] = k.Player_A[i];
        Player_B[i] = k.Player_B[i];

    }

    store_A = k.store_A;
    store_B = k.store_B;
    slots = 5;
    pointer = NULL;
    maxWinNumber = 36;

}

char Kalah::move_decider(int hole_num,char player)
{
    if(player == 'A')
```

```
        return (move_generator_A(hole_num));
    else
        return (move_generator_B(hole_num));
}


/*******************************************************************
***********
move_generator_A(): This function helps generate moves for Player A
                    If last stone lands in Player A's store, Player
A gets another chance. If player A's stone lands in an empty hole in
Player A's holes, player A captures all player B's stone from the
opposite hole. If the above two special scenarios are not
encountered,
the moves are normal.
parameters used: Hole number from which the player should start
playing
returns : Returns the next player that gets a chance after A's move.
          In most cases, B gets a turn. When A's stone lands in its
store, A
          is returned, meaning, A gets an extra move
********************************************************************
***********/

char Kalah::move_generator_A(int hole_num)
{
    int current_Position = hole_num;
    pointer = Player_A;

    int stoneCount = pointer[current_Position];

    pointer[current_Position] = 0;
    int opPosition, opStones;

    current_Position++;

    while(stoneCount > 0)
    {
        //last stone lands in Player A's store
        if(stoneCount >= 1 && current_Position == 6 )
        {
            stoneCount--;
            store_A ++;

            if(stoneCount == 0)
```

```
                return 'A'; // giving A another chance as last stone
falls in A's store
        }

        else if(current_Position >= 0 && current_Position <= 5)
        {
            if(stoneCount == 1)
            {
                stoneCount--;

                opPosition = slots - current_Position;

                if(pointer[current_Position] == 0 &&
Player_B[opPosition] > 0)
                {
                    pointer = Player_B;
                    opStones = pointer[opPosition];
                    pointer[opPosition] = 0;

                    store_A += opStones + 1;

                    if(stoneCount == 0)
                        return 'B';
                }

                else
                {
                    pointer[current_Position]++;

                    if(stoneCount == 0)
                        return 'B';
                }
            }

            else if(stoneCount > 1)
            {
                stoneCount--;

                pointer[current_Position]++;
            }
        }

        else if(current_Position > 6 && current_Position <= 12)
        {
            pointer = Player_B;
            stoneCount--;

            pointer[current_Position-7]++;

            if(stoneCount == 0)
                return 'B';
        }

        //skip Player B's store
        else if(current_Position >= 12)
        {
            current_Position = -1;
            pointer = Player_A;
        }

        current_Position++;
    }
    return 'B';
}

/*******************************************************************
***********
move_generator_B(): This function helps generate moves for Player B
                    If last stone lands in Player B's store, Player
B gets another chance. If player B's stone lands in an empty hole in
Player B's holes, player B captures all player A's stone from the
opposite hole. If the above two special scenarios are not
encountered,
the moves are normal.
parameters used: Hole number from which the player should start
playing
returns : Returns the next player that gets a chance after B's move.
          In most cases, A gets a turn. When B's stone lands in its
store, B is returned, meaning, B gets an extra move
*******************************************************************
***********/
char Kalah::move_generator_B(int hole_num)
{
    int current_Position = hole_num;
    pointer = Player_B;

    int stoneCount = pointer[current_Position];

    pointer[current_Position] = 0;
    int opPosition, opStones;
    current_Position++;

    while(stoneCount > 0)
    {
        //last stone lands in Player B's store
        if(stoneCount >= 1 && current_Position == 6)
        {
            stoneCount--;
            store_B++;

            if(stoneCount == 0)
                return 'B'; // giving B another chance as last stone
falls in B's store
        }

        else if(current_Position >= 0 && current_Position <= 5)
        {
            if(stoneCount == 1)
            {
                stoneCount--;

                opPosition = slots - current_Position;

                if(pointer[current_Position] == 0)
                {
                    pointer = Player_A;
                    opStones = pointer[opPosition];
                    pointer[opPosition] = 0;

                    store_B += opStones + 1;

                    if(stoneCount == 0)
                        return 'A';
                }

                else
                {
                    pointer[current_Position]++;

                    if(stoneCount == 0)
                        return 'A';
                }
            }

            else if(stoneCount > 1)
```

```
                {
                    stoneCount--;

                    pointer[current_Position]++;
                }
            }

            else if(current_Position > 6 && current_Position <= 12)
            {
                pointer = Player_A;
                stoneCount--;

                pointer[current_Position-7]++;

                if(stoneCount == 0)
                    return 'A';
            }

            else if(current_Position >= 12)
            {
                current_Position = -1;
                pointer = Player_B;
            }

            current_Position++;
        }
        return 'A';
}


/*******************************************************************
***********
result(): This function checks the result of the game. If A has more
than 36 stones, A wins. If B has more than 36 stones, B wins.
parameters used: No parameters
returns : Returns A if Player A wins, B if player B wins, N if no
win yet.
********************************************************************
***********/

char Kalah::result()
{
    int empty_pits_A = 0,
        empty_pits_B = 0;
    for(int i = 0; i < SIZE; i++)
    {
        if(Player_A[i]==0)
            empty_pits_A++;

        if(Player_B[i]==0)
            empty_pits_B++;
    }

    if(empty_pits_A == 6)
    {
        for(int i = 0 ; i < SIZE; i++)
        {
            store_B += Player_B[i];
            Player_B[i] = 0;
        }

        cout << "Player A ran out of all the stones! All the pits in
A are empty! " << endl;
    }

    if( empty_pits_B == 6)
    {
        for(int i = 0 ; i < SIZE ; i++)
        {
            store_A += Player_A[i];
            Player_A[i] = 0;
        }

        cout << "Player B ran out of all the stones! All the pits in
B are empty! " << endl;
    }

    //if Player A has more than 36 stones, Player A wins
    if(store_A > maxWinNumber)
        return 'A';

    //if Player B has more than 36 stones, Player B wins
    else if(store_B > maxWinNumber)
        return 'B';

    else
        return 'N';
}
```

```
class KalahFlow
{

public:

    char player;
    Kalah current_status;
    int heuristic;
    KalahFlow * successors[SIZE];
    int successor_count;

    KalahFlow(char);
    void current_player_status(Kalah);
    bool is_deep_enough(int, int);
    void generate_successors();
    int myEvaluationFunction1();
    int myEvaluationFunction2();
    int myEvaluationFunction3();
    int myEvaluationFunction4();

};

/*******************************************************************
***********
KalahFlow(): A parameterized constructor that is used to initialize
an object of class KalahFlow.
parameters used: player
returns : constructor does not have a return type.
********************************************************************
***********/

KalahFlow::KalahFlow(char player)
{
    this->player = player;
    heuristic = -2000;
    successor_count = 0;

    for(int i = 0; i < SIZE; i++)
        successors[i] = NULL;
}
```

```
/*******************************************************************
***********
current_player_status(): copies the current status of the player
into the current_status object of type Kalah
parameters used: Kalah k
returns : returns nothing
*******************************************************************
***********/

void KalahFlow::current_player_status(Kalah k)
{
    for(int i = 0; i < SIZE; i++)
    {
        this->current_status.Player_A[i] = k.Player_A[i];
        this->current_status.Player_B[i] = k.Player_B[i];
    }

    this->current_status.store_A = k.store_A;
    this->current_status.store_B = k.store_B;
}

/*******************************************************************
***********
is_deep_enough(): This function checks if the depth is 2 or 4, which
gives true for is_deep_enough for the algorithms. If it is not deep
enough,we generate one more ply of the tree by calling the
generate_successors()function
parameters used: the current depth passed by the functions, the
selected maximum depth that the ply can be generated up to
returns : If the depth is greater than or equal to the selected
depth, the function returns true. Otherwise, false is returned
*******************************************************************
***********/

bool KalahFlow::is_deep_enough(int depth, int selected)
{
    if(heuristic != -2000)
        return heuristic;

    if(depth >= selected || current_status.result() != 'N' )
        return true;

    else
    {
        expanded_node_count++;
        generate_successors();
        return false;
    }
}

/*********************************************************************
***********
generate_successors(): generates successors(child node) when the
deep enough function is satisfied to evaluate the future moves of
the opponent based on the present state of the board. This root node
has child branches which ends in the heuristic value of the end leaf
node. Here the leaf node is the value of number of stones in the pit
at a given board state.
parameters used: no parameters used
returns : returns nothing
*********************************************************************
***********/

void KalahFlow::generate_successors()
{
    char player_temp;

    if(player == 'A')
        player_temp = 'B';

    else
        player_temp = 'A';

    for(int i = 0; i < SIZE; i++)
    {
        successor_count++;

        successors[i] = new KalahFlow(player_temp);

        if(current_status.Player_A[i]!=0 && player == 'A')
            successors[i]->current_status = current_status;

        else if(current_status.Player_B[i]!=0 && player == 'B')
            successors[i]->current_status = current_status;

        else
            successors[i] = NULL;

        if(successors[i]!=NULL)
        {
            total_nodes++;

            successors[i]-
>current_status.move_decider(i,successors[i]->player);
        }
    }
}

/*******************************************************************
***********
my_evaluation_function1():used to estimate the value or goodness of
a position at leaf node
parameters used: no parameters
returns : returns the terminal value
********************************************************************
***********/

int KalahFlow::myEvaluationFunction1()
{
    int terminal_value;

    if(player == 'A')
    {
        int empty_pits_A = 0;

        for(int i = 0 ; i < 6 ; i++)
        {
            if(current_status.Player_A[i] == 0)
                empty_pits_A++;
        }

        if(empty_pits_A == 6)
            terminal_value = 2000;
    }
    else if(player == 'B')
    {
        int empty_pits_B = 0;

        for(int i = 0 ; i < 6 ; i++)
        {
            if(current_status.Player_B[i] == 0)
                empty_pits_B++;
        }
        if(empty_pits_B == 6)
            terminal_value = -2000;
    }

    heuristic = terminal_value;

    return terminal_value;
}

/*******************************************************************
***********
my_evaluation_function2():used to estimate the value or goodness of
a position at leaf node
parameters used: no parameters
returns : returns the terminal value
********************************************************************
***********/

int KalahFlow::myEvaluationFunction2()
{

int terminal_value;
    if(player == 'A')
        terminal_value = current_status.store_A -
current_status.store_B;

    else if(player == 'B')
        terminal_value = current_status.store_A -
current_status.store_B;

    heuristic = terminal_value;

    return terminal_value;

}

/*******************************************************************
***********
my_evaluation_function3():used to estimate the value or goodness of
a position at leaf node
parameters used: no parameters
returns : returns the terminal value
********************************************************************
***********/

int KalahFlow::myEvaluationFunction3()
{
    int p0 = current_status.store_A;
    int p1 = current_status.store_B;
    int terminal_value;

    if(player == 'A')
    {
        if(p0 != 0)
            terminal_value =
(current_status.store_A/current_status.store_B);

        else
            return -1;
    }

    else if(player == 'B')
    {
        if(p1 !=0)
            terminal_value = (current_status.store_B/
current_status.store_A);

        else
            return -1;

    heuristic = terminal_value;
    return terminal_value;
}
}

/*******************************************************************
***********
my_evaluation_function4():used to estimate the value or goodness of
a position at leaf node
parameters used: no parameters
returns : returns the terminal value
********************************************************************
***********/

int KalahFlow::myEvaluationFunction4()
{
    int terminal_value;
    if (player == 'A'){
        int stones_of_A = 0;
        for(int i=0; i<6; i++) {
            if (current_status.Player_A[i]==0) stones_of_A++;
            else if(current_status.Player_A[i]==i) terminal_value =
1500;
            else if(current_status.Player_A[i]<i) terminal_value =
800;
            else
                terminal_value = 500;
        }
        if(stones_of_A == 6)
            terminal_value = 1000; }
    if (player == 'B'){
        int stones_of_B = 0;
        for(int i=0; i<6; i++) {
            if (current_status.Player_A[i]==0) stones_of_B++;
            else if(current_status.Player_A[i]==i) terminal_value =
1500;
            else if(current_status.Player_A[i]<i) terminal_value =
800;
            else
                terminal_value = 500;
        }
        if(stones_of_B == 6)
            terminal_value = 1000; }
    heuristic = terminal_value;
    return terminal_value;

}


/********************************************************************
***********
AlphaBetaSearch(): Alpha Beta search algorithm implementation for
Kalah Game
parameters used: an object of type KalahFlow, the current depth
which is 0 during the initial call,the current player, which was 'A'
initially, alpha, beta, selectedDepth, which can be 2 or 4, the
maximum depth, evaluation_choice 1, 2 or 3 select from three
possible evaluation function options.
returns : returns the best value
********************************************************************
***********/

int AlphaBetaSearch(KalahFlow *alphabeta, int depth, char player,
int alpha, int beta, int selected, int evaluation_choice)
{
    int bestValue = -100;
    int value;

    if(alphabeta->is_deep_enough(depth, selected))
    {
        if(evaluation_choice == 1)
            return alphabeta->myEvaluationFunction1();

        else if (evaluation_choice == 2)
            return alphabeta->myEvaluationFunction2();

        else if (evaluation_choice == 3)
            return alphabeta->myEvaluationFunction3();

        else
            return alphabeta->myEvaluationFunction4();
    }

    if (player == 'A' )

    {

        for (int i=0; i < 6; i++ )

        {

            if(alphabeta->successors[i]== NULL)

            continue;

            value = AlphaBetaSearch(alphabeta->successors[i],
depth+1, alphabeta->player, alpha, beta, selected,
evaluation_choice);

            if (bestValue > value)

            bestValue = max(bestValue,value);

            alpha=max(alpha,bestValue);

            if (beta <= alpha)

            break;  }

        alphabeta->heuristic = bestValue;
        return bestValue;

    }


    else

    {

        int bestValue = +100;

        int value;



        for(int i=0; i < 6; i++)

        {

            if(alphabeta->successors[i]== NULL)

            continue;

            value = AlphaBetaSearch(alphabeta->successors[i],
depth+1, alphabeta->player, alpha, beta, selected,
evaluation_choice);

            if (bestValue < value)

            bestValue = min( bestValue, value);

            beta = min( beta, bestValue);

            if (beta <= alpha)

            break;

        }

        alphabeta->heuristic = bestValue;

        return bestValue;

    }
```

```
}

/*******************************************************************
***********
MinMaxAB(): MinMAxAB search algorithm implementation for Kalah Game
parameters used: an object of type KalahFlow, the current depth
which is 0 during the initial call,the current player, which was 'A'
initially, Use-Threshold, Pass-Threshold value, selectedDepth, which
can be 2 or 4, the maximum depth, evaluation_choice-1, 2 or 3 - to
select from three possible evaluation function options.
returns : returns the best value
*******************************************************************
***********/

int MinMaxAB(KalahFlow *min_max,int depth, char player, int
use_Threshold, int pass_Threshold, int selectedDepth, int
evaluation_choice)
{
    int structure;
    int new_value;
    char new_Player;
    int result_successor = 0;

    if(min_max->is_deep_enough(depth, selectedDepth))
    {

        if(evaluation_choice == 1)
            structure = min_max->myEvaluationFunction1();

        else if (evaluation_choice == 2)
            structure = min_max->myEvaluationFunction2();

        else if (evaluation_choice == 3)
            structure = min_max->myEvaluationFunction3();

        else
            structure = min_max->myEvaluationFunction4();

        if(player == 'B')
            structure = -structure;

        min_max->heuristic = structure;

        return structure;
    }

    for(int i = 0 ; i < SIZE; i++)
    {
        if(min_max->successors[i] == NULL)
            continue;

        if(player == 'A')
            new_Player = 'B';

        else
            new_Player = 'A';

        result_successor = MinMaxAB(min_max-
>successors[i],depth+1,new_Player,-pass_Threshold,-use_Threshold,
selectedDepth, evaluation_choice);

        new_value = -result_successor;

        if(new_value > pass_Threshold)
        {
            min_max->heuristic = i;
            pass_Threshold = new_value;
        }

        if(pass_Threshold >= use_Threshold)
        {
            result_successor = pass_Threshold;
            return result_successor;
        }
    }

    result_successor = pass_Threshold;

    return result_successor;
}
```

```
/*******************************************************************
***********
winner(): Prints the result of the winner, the computation time,
memory consumed, number of nodes generated, number of nodes expanded
and the total game path length
parameters used: result if A won the game or B, total time , object
of type Kalah
returns : returns nothing
********************************************************************
***********/

void winner(char result,int total_time, Kalah k)
{
    if(result == 'A')
        cout << "Player A won the GAME with " << k.store_A << "
stones! " << endl << endl;

    else if(result == 'B')
        cout << "Player B won the GAME with " << k.store_B << "
stones! " << endl << endl;

    cout <<
"*******************************************************************
***********************************************" << endl;
    cout << "
***GAME OVER***                                                  "
<< endl ;
    cout <<
"*******************************************************************
***********************************************" << endl << endl <<
endl;

    cout << "The program took " << "'" <<
(total_time)/double(CLOCKS_PER_SEC)<< "'" << " seconds of Execution
time for the completed game! " << endl << endl;
    cout << "1 node takes 81 bytes of Memory "<<endl << endl;
    cout << "The algorithm takes : "<< "'" << (81* total_nodes) <<
"'" << " bytes = " << "'" << (81* total_nodes) / (1024) << "'" << "
kb" << " of memory!" << endl << endl;
    cout << "The program generated " << "'" << total_nodes << "'" <<
" nodes for the completed game!" << endl << endl;
    cout << "The program expanded " << "'" << expanded_node_count <<
"'" << " nodes for the completed game!" << endl << endl;
```

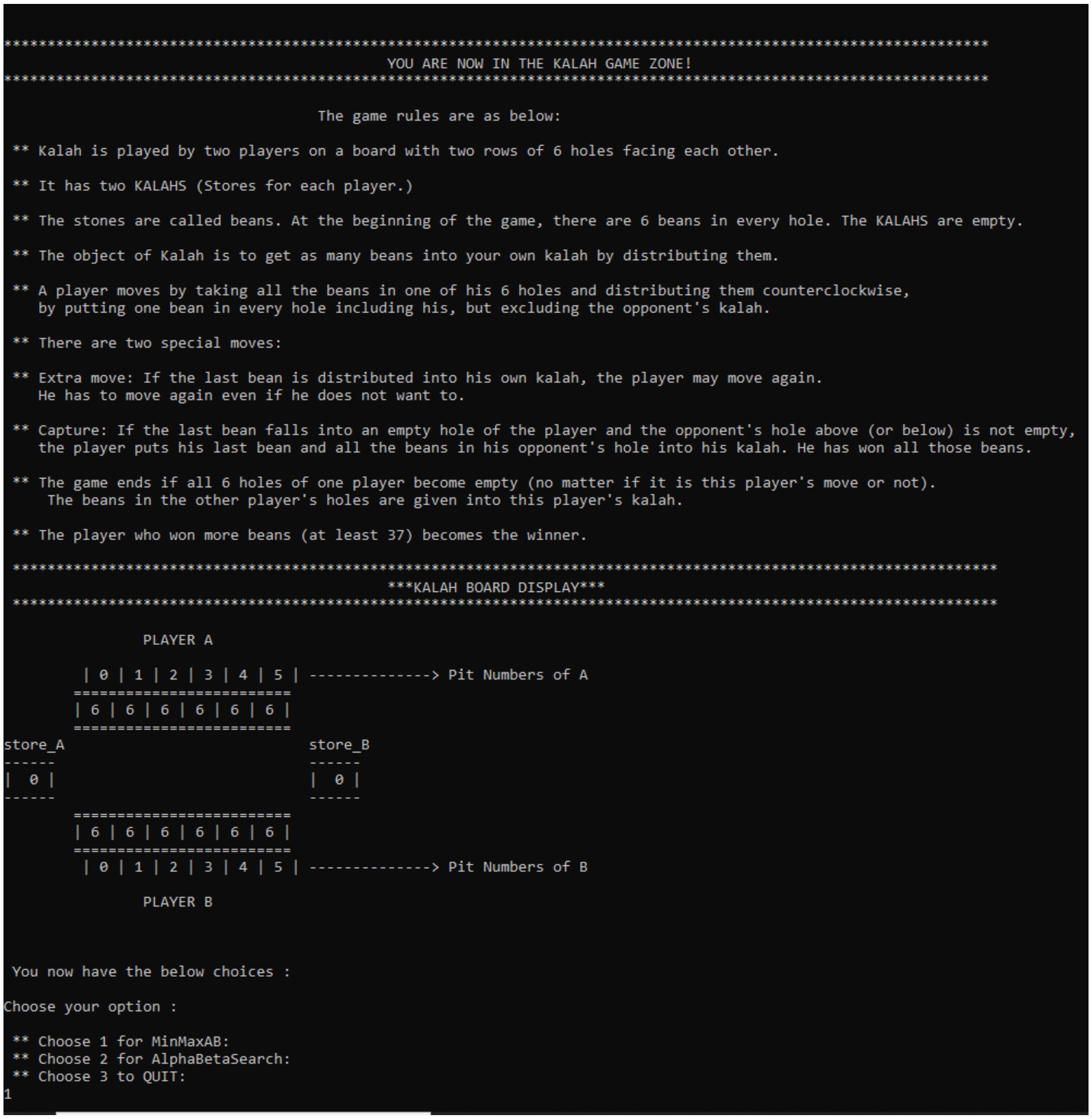
```
    cout << "The total length of the Game path is " << "'" <<
game_path_length << "'" << " for the completed game! " << endl <<
endl;

}

/*******************************************************************
***********
main():The program starts execution from this function main. It is
the driver to integrate all the classes and functions defined above
parameters used: no parameters
********************************************************************
***********/
int main()
{
    int choice, selectedDepth;
    int evaluation_choice;
    Kalah *kalah = new Kalah();


        cout << endl << endl;
        cout <<
"*******************************************************************
**********************************************" << endl;
        cout << "                                                 YOU ARE
NOW IN THE KALAH GAME ZONE!                              " <<
endl ;
        cout <<
"*******************************************************************
**********************************************" << endl;
        cout << endl;
        cout << "                                         The game rules
are as below:                               " << endl << endl;
        cout << " ** Kalah is played by two players on a board with
two rows of 6 holes facing each other." << endl << endl;
        cout << " ** It has two KALAHS (Stores for each player.)" <<
endl << endl;
        cout << " ** The stones are called beans. At the beginning
of the game, there are 6 beans in every hole. The KALAHS are empty."
<< endl << endl;
        cout << " ** The object of Kalah is to get as many beans
into your own kalah by distributing them." << endl << endl;
        cout << " ** A player moves by taking all the beans in one
of his 6 holes and distributing them counterclockwise," << endl;
```

```
        cout << "    by putting one bean in every hole including
his, but excluding the opponent's kalah." << endl << endl;
        cout << " ** There are two special moves:" << endl << endl;
        cout << " ** Extra move: If the last bean is distributed
into his own kalah, the player may move again. " << endl
             << "    He has to move again even if he does not want
to." << endl << endl;
        cout << " ** Capture: If the last bean falls into an empty
hole of the player and the opponent's hole above (or below) is not
empty," << endl
             << "    the player puts his last bean and all the beans
in his opponent's hole into his kalah. He has won all those beans."
<< endl << endl;
        cout << " ** The game ends if all 6 holes of one player
become empty (no matter if it is this player's move or not)." <<
endl
             <<"     The beans in the other player's holes are given
into this player's kalah." << endl << endl;
        cout << " ** The player who won more beans (at least 37)
becomes the winner." << endl << endl;


        cout << "
********************************************************************
*********************************************" << endl;
        cout << "
***KALAH BOARD DISPLAY***
" << endl ;
        cout << "
********************************************************************
*********************************************" << endl;

        kalah->display();

        cout << " You now have the below choices :" << endl << endl
             << "Choose your option :" << endl << endl;

        cout << " ** Choose 1 for MinMaxAB: " << endl;
        cout << " ** Choose 2 for AlphaBetaSearch: " << endl;
        cout << " ** Choose 3 to QUIT:" << endl;
        cin >> choice;

        cout << "Please enter the depth that you would want to check
the results for: " << endl;
        cout << "2: Depth of 2" << endl;
        cout << "4: Depth of 4" << endl;
        cin >> selectedDepth;

        while(selectedDepth !=2 && selectedDepth !=4)
        {
            cout << "Invalid Depth! Please enter depth value 2 or
4!" << endl;
            cin >> selectedDepth;
        }

        cout << "Please enter the Evaluation function that you would
want to check the results for: "<< endl;
        cout << "1: Evaluation Function 1 ( Vidhyashree Nagabhushana
) " << endl;
        cout << "2: Evaluation Function 2 ( Akshatha Jain ) " <<
endl;
        cout << "3: Evaluation Function 3 ( Akshatha Jain ) " <<
endl;
        cout << "4: Evaluation Function 4 ( Sahana Sreenath) " <<
endl;
        cin >> evaluation_choice;


        while(evaluation_choice < 1 || evaluation_choice > 4 )
        {
            cout << "Invalid selection! Please enter values 1, 2, 3
or 4!" << endl;
            cin >> evaluation_choice;
        }

        switch(choice)
        {
            case 1:
                {
                    cout << "Initial board " << endl;
                    kalah->display();

                    char result = kalah->result();

                    char player ='A';
                    int start_timer = clock();

                    while(result == 'N')
                    {
                        KalahFlow *min_max_ab = new
KalahFlow(player);

                        min_max_ab->current_player_status( kalah );

                        cout << "It is player " << player << "'s
turn!" << endl;

                        MinMaxAB(min_max_ab,0,player,2000,-
2000,selectedDepth,evaluation_choice);

                        int pit = min_max_ab->heuristic;

                        if(player == 'A')
                            player = kalah->move_generator_A(pit);

                        else
                            player = kalah->move_generator_B(pit);

                        kalah->display();

                        result = kalah->result();

                        game_path_length++;
                    }

                    int stop_timer = clock();
                    kalah->display();
                    int total_time = stop_timer - start_timer;
                    winner(result, total_time, kalah );
                }

                break;

            case 2:
                {
                    cout << "Initial board " << endl;
                    kalah->display();

                    char result = kalah->result();

                    char player = 'A';
                    int start_timer = clock();

                    while(result == 'N')
                    {
                        KalahFlow *alpha_beta_flow = new
KalahFlow(player);

                        alpha_beta_flow-
>current_player_status(kalah);

                        cout << "It is player " << player << "'s
turn!" << endl;
AlphaBetaSearch(alpha_beta_flow,0,player,2000,-2000,
selectedDepth,evaluation_choice);

                        int pit;

                        for(int i = 0; i < SIZE; i++)
                        {
                            if(alpha_beta_flow->successors[i] ==
NULL)
                                continue;

                            if(alpha_beta_flow->successors[i]-
>heuristic == alpha_beta_flow->heuristic)
                                pit = i;
                        }

                        if(player == 'A')
                            player = kalah->move_generator_A(pit);

                        else
                            player = kalah->move_generator_B(pit);

                        kalah->display();

                        result = kalah->result();

                        game_path_length++;
                    }

                    int stop_timer = clock();
                    kalah->display();
                    int total_time = stop_timer - start_timer;
                    winner(result,total_time, kalah);
                }

                break;
```

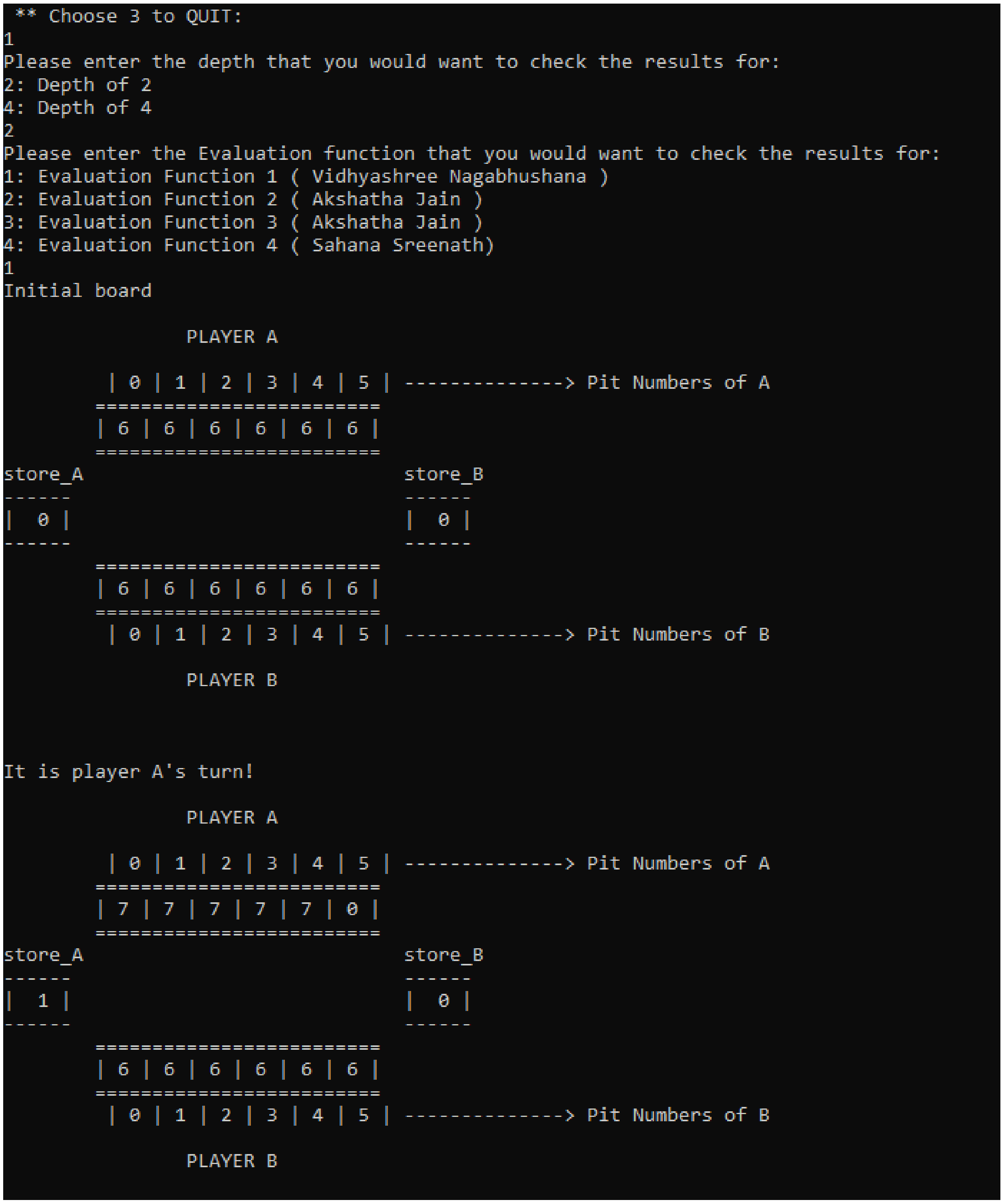
```
            case 3:
                cout <<"Thanks. See you again later........." <<
endl;
                return 0;
        }

    return 0;
}
```

7 SAMPLE PROGRAM RUN

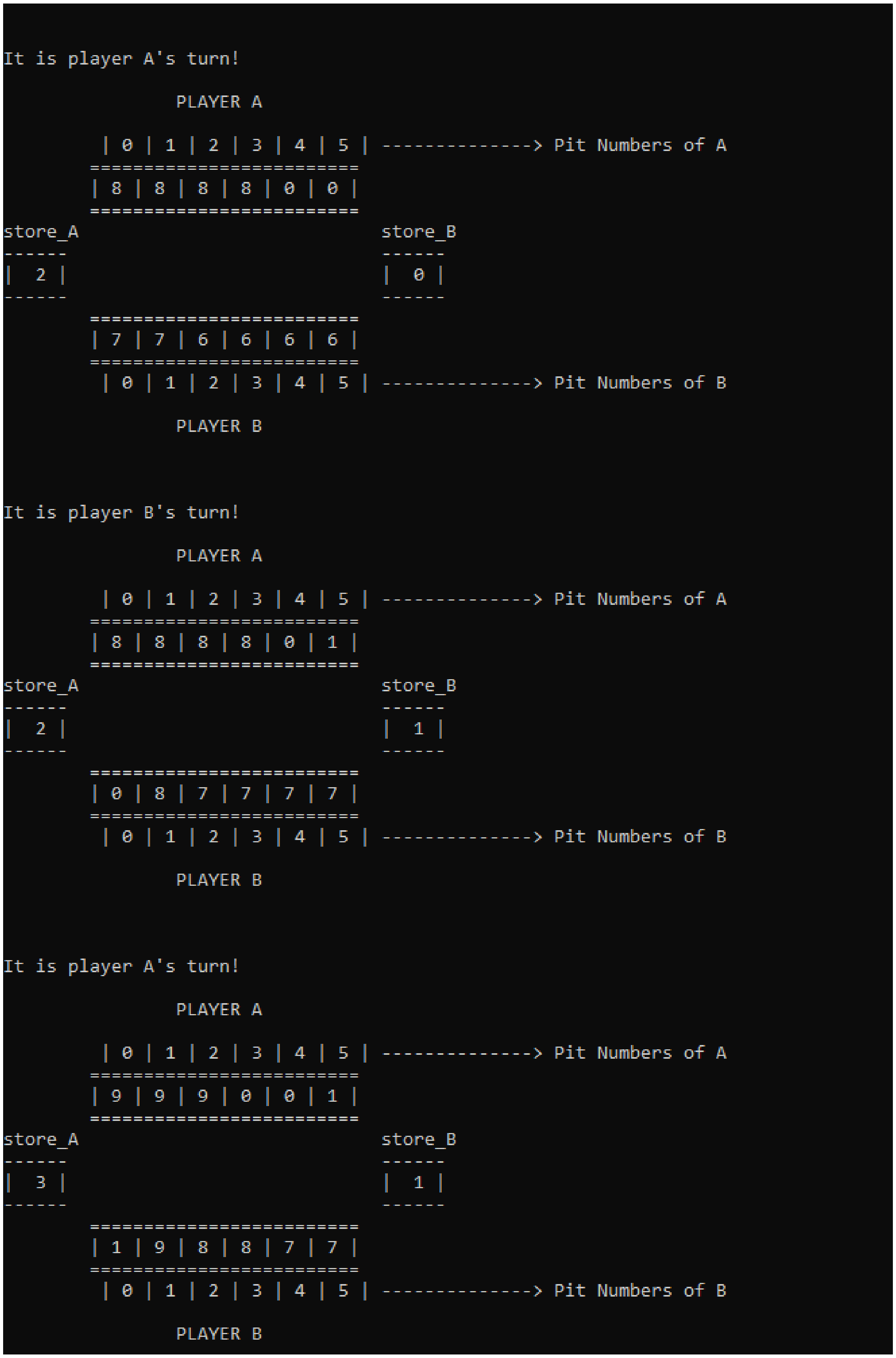
Below is the sample program run 1: Evaluation Function 1, Depth 2, MinMaxAB algorithm:



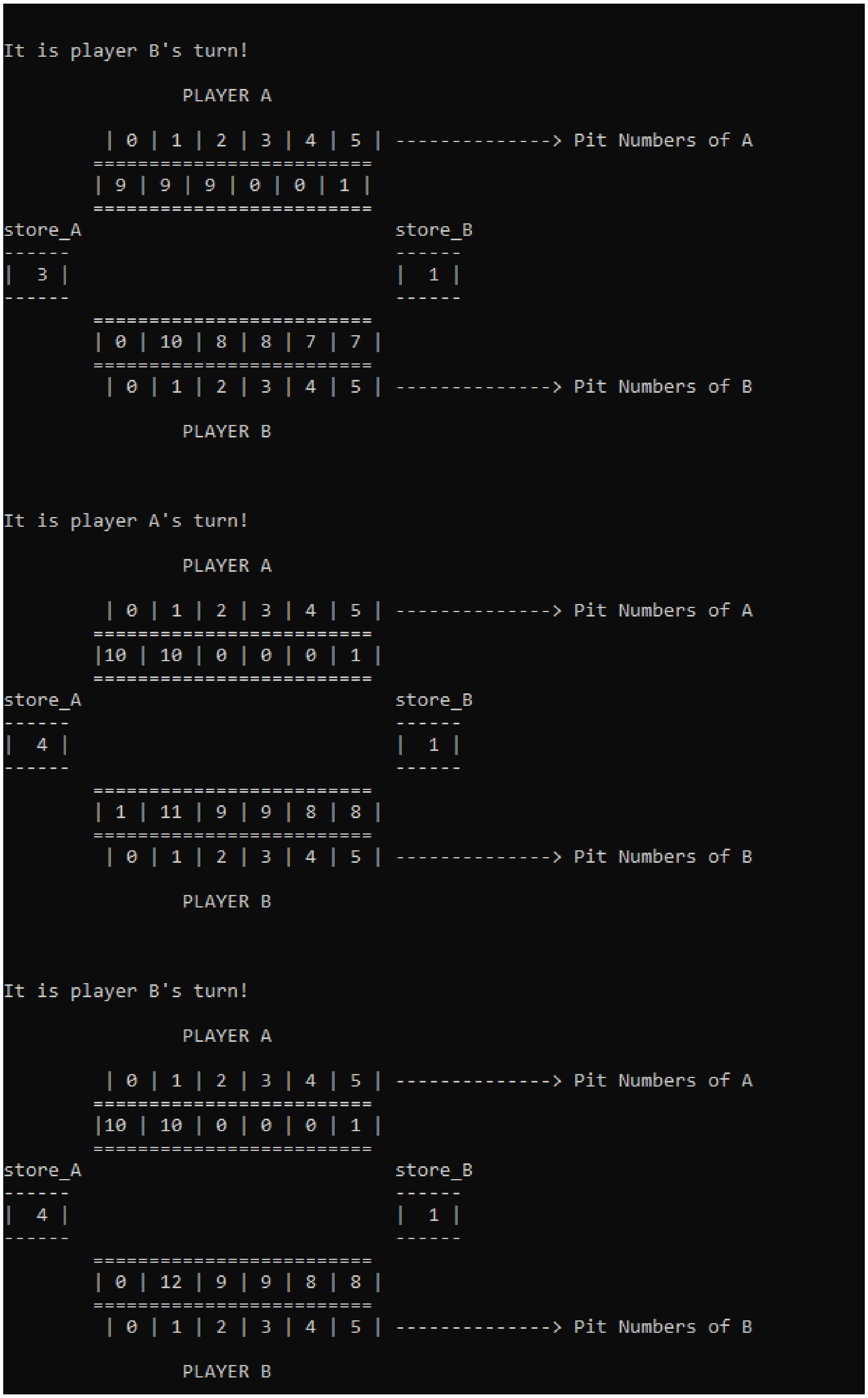
```
*****************************************************************************************************************
                                            YOU ARE NOW IN THE KALAH GAME ZONE!
*****************************************************************************************************************

                                    The game rules are as below:

 ** Kalah is played by two players on a board with two rows of 6 holes facing each other.

 ** It has two KALAHS (Stores for each player.)

 ** The stones are called beans. At the beginning of the game, there are 6 beans in every hole. The KALAHS are empty.

 ** The object of Kalah is to get as many beans into your own kalah by distributing them.

 ** A player moves by taking all the beans in one of his 6 holes and distributing them counterclockwise,
    by putting one bean in every hole including his, but excluding the opponent's kalah.

 ** There are two special moves:

 ** Extra move: If the last bean is distributed into his own kalah, the player may move again.
    He has to move again even if he does not want to.

 ** Capture: If the last bean falls into an empty hole of the player and the opponent's hole above (or below) is not empty,
    the player puts his last bean and all the beans in his opponent's hole into his kalah. He has won all those beans.

 ** The game ends if all 6 holes of one player become empty (no matter if it is this player's move or not).
     The beans in the other player's holes are given into this player's kalah.

 ** The player who won more beans (at least 37) becomes the winner.

 *****************************************************************************************************************
                                            ***KALAH BOARD DISPLAY***
 *****************************************************************************************************************

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 6 | 6 | 6 | 6 | 6 | 6 |
        =========================
store_A                            store_B
------                             ------
|  0 |                             |  0 |
------                             ------
        =========================
        | 6 | 6 | 6 | 6 | 6 | 6 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


 You now have the below choices :

Choose your option :

 ** Choose 1 for MinMaxAB:
 ** Choose 2 for AlphaBetaSearch:
 ** Choose 3 to QUIT:
1
```



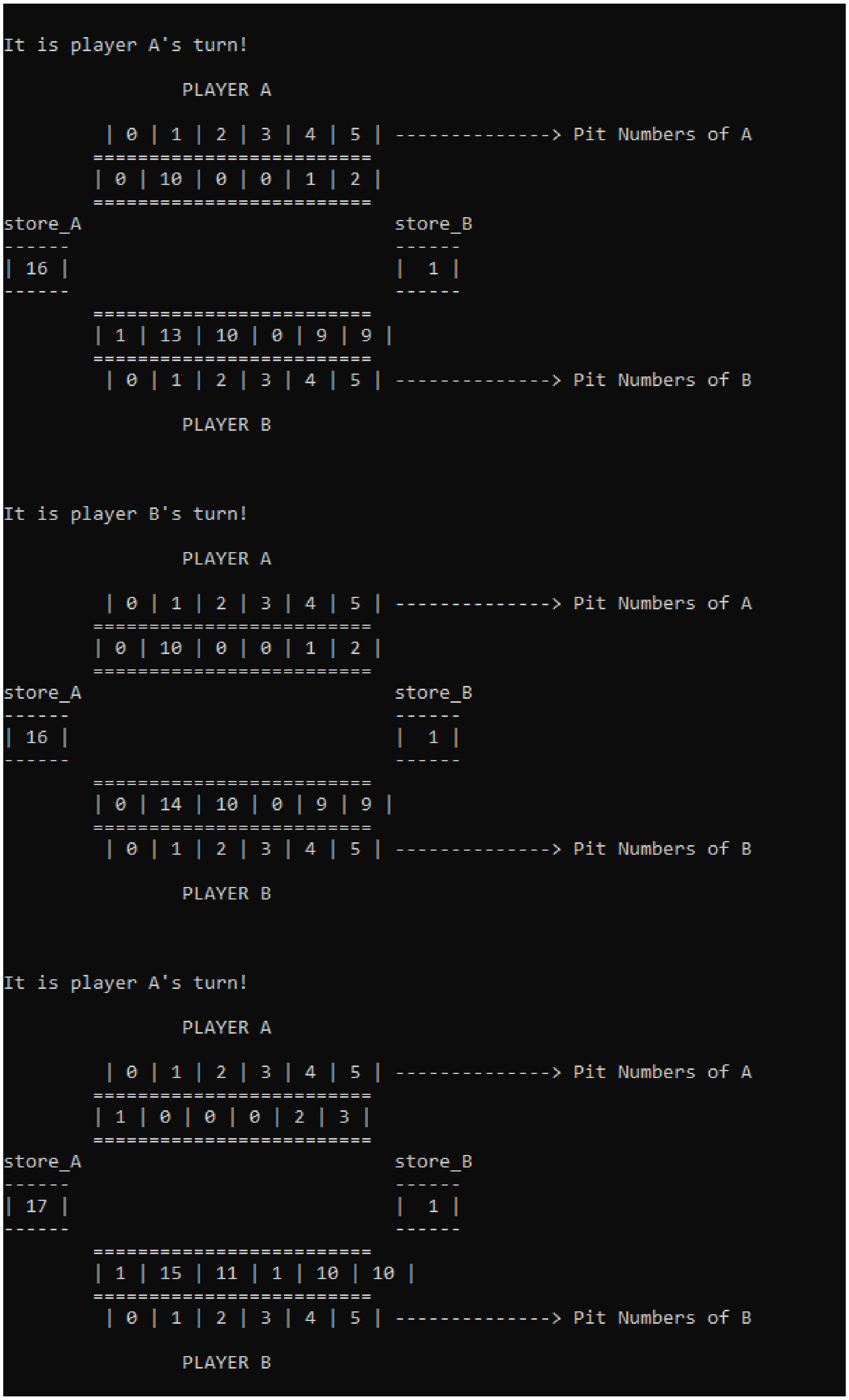
```
 ** Choose 3 to QUIT:
1
Please enter the depth that you would want to check the results for:
2: Depth of 2
4: Depth of 4
2
Please enter the Evaluation function that you would want to check the results for:
1: Evaluation Function 1 ( Vidhyashree Nagabhushana )
2: Evaluation Function 2 ( Akshatha Jain )
3: Evaluation Function 3 ( Akshatha Jain )
4: Evaluation Function 4 ( Sahana Sreenath)
1
Initial board

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 6 | 6 | 6 | 6 | 6 | 6 |
        =========================
store_A                            store_B
------                             ------
|  0 |                             |  0 |
------                             ------
        =========================
        | 6 | 6 | 6 | 6 | 6 | 6 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B



It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 7 | 7 | 7 | 7 | 7 | 0 |
        =========================
store_A                            store_B
------                             ------
|  1 |                             |  0 |
------                             ------
        =========================
        | 6 | 6 | 6 | 6 | 6 | 6 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B
```



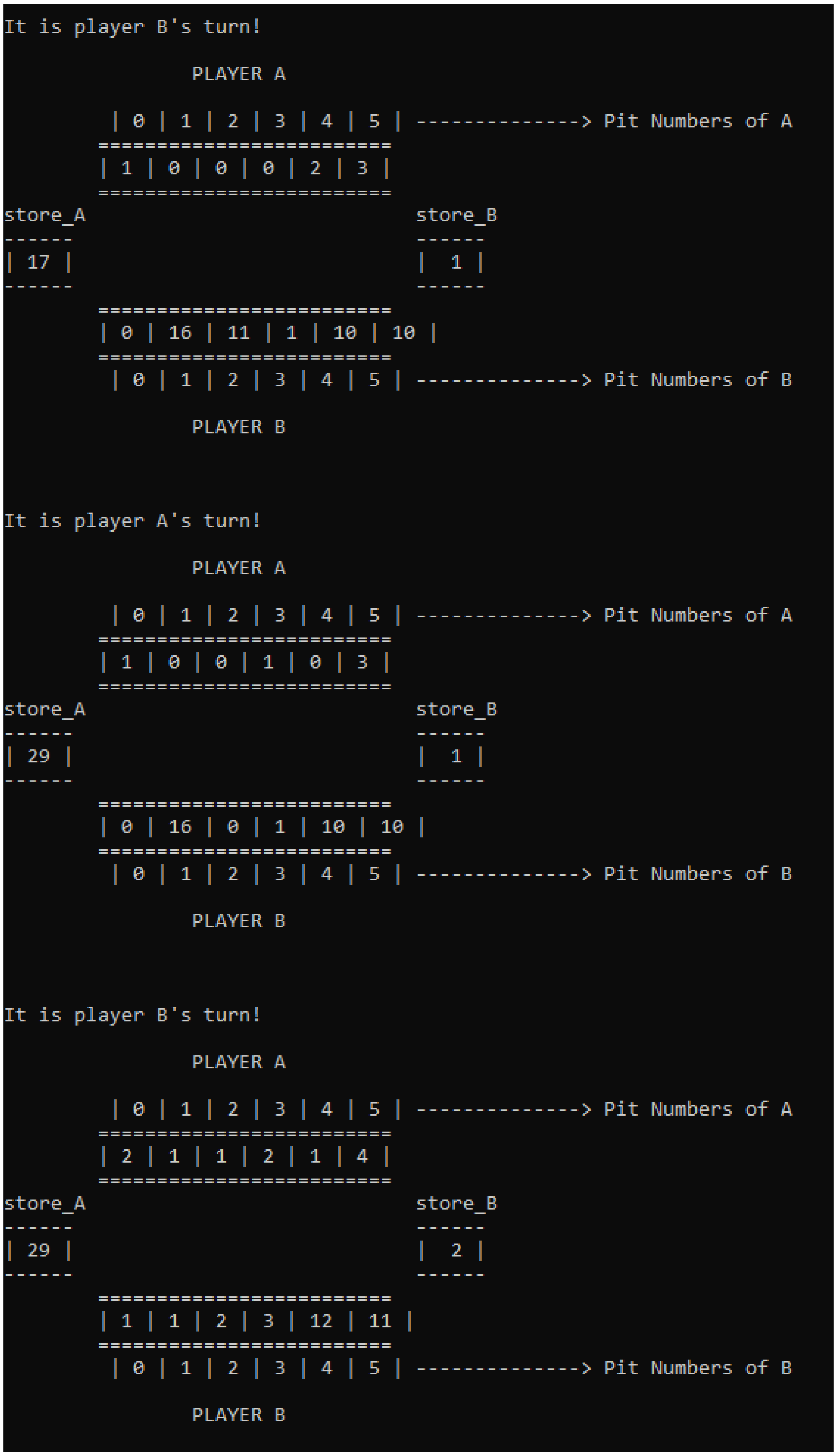
```
It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 8 | 8 | 8 | 8 | 0 | 0 |
        =========================
store_A                            store_B
------                             ------
|  2 |                             |  0 |
------                             ------
        =========================
        | 7 | 7 | 6 | 6 | 6 | 6 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 8 | 8 | 8 | 8 | 0 | 1 |
        =========================
store_A                            store_B
------                             ------
|  2 |                             |  1 |
------                             ------
        =========================
        | 0 | 8 | 7 | 7 | 7 | 7 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 9 | 9 | 9 | 0 | 0 | 1 |
        =========================
store_A                            store_B
------                             ------
|  3 |                             |  1 |
------                             ------
        =========================
        | 1 | 9 | 8 | 8 | 7 | 7 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B
```



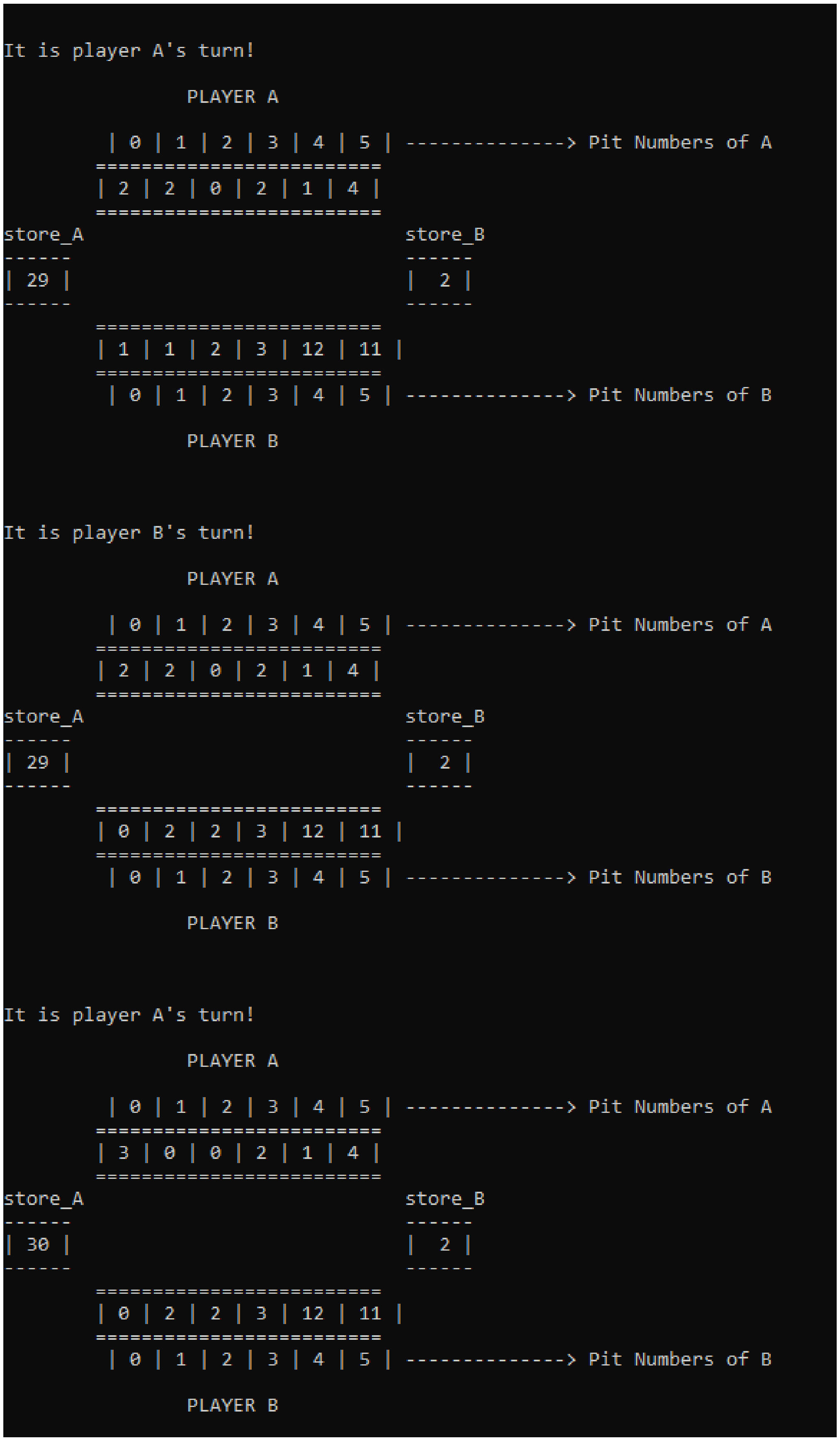
```
It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 9 | 9 | 9 | 0 | 0 | 1 |
        =========================
store_A                            store_B
------                             ------
|  3 |                             |  1 |
------                             ------
        =========================
        | 0 | 10 | 8 | 8 | 7 | 7 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B



It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        |10 | 10 | 0 | 0 | 0 | 1 |
        =========================
store_A                            store_B
------                             ------
|  4 |                             |  1 |
------                             ------
        =========================
        | 1 | 11 | 9 | 9 | 8 | 8 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B



It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        |10 | 10 | 0 | 0 | 0 | 1 |
        =========================
store_A                            store_B
------                             ------
|  4 |                             |  1 |
------                             ------
        =========================
        | 0 | 12 | 9 | 9 | 8 | 8 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B
```



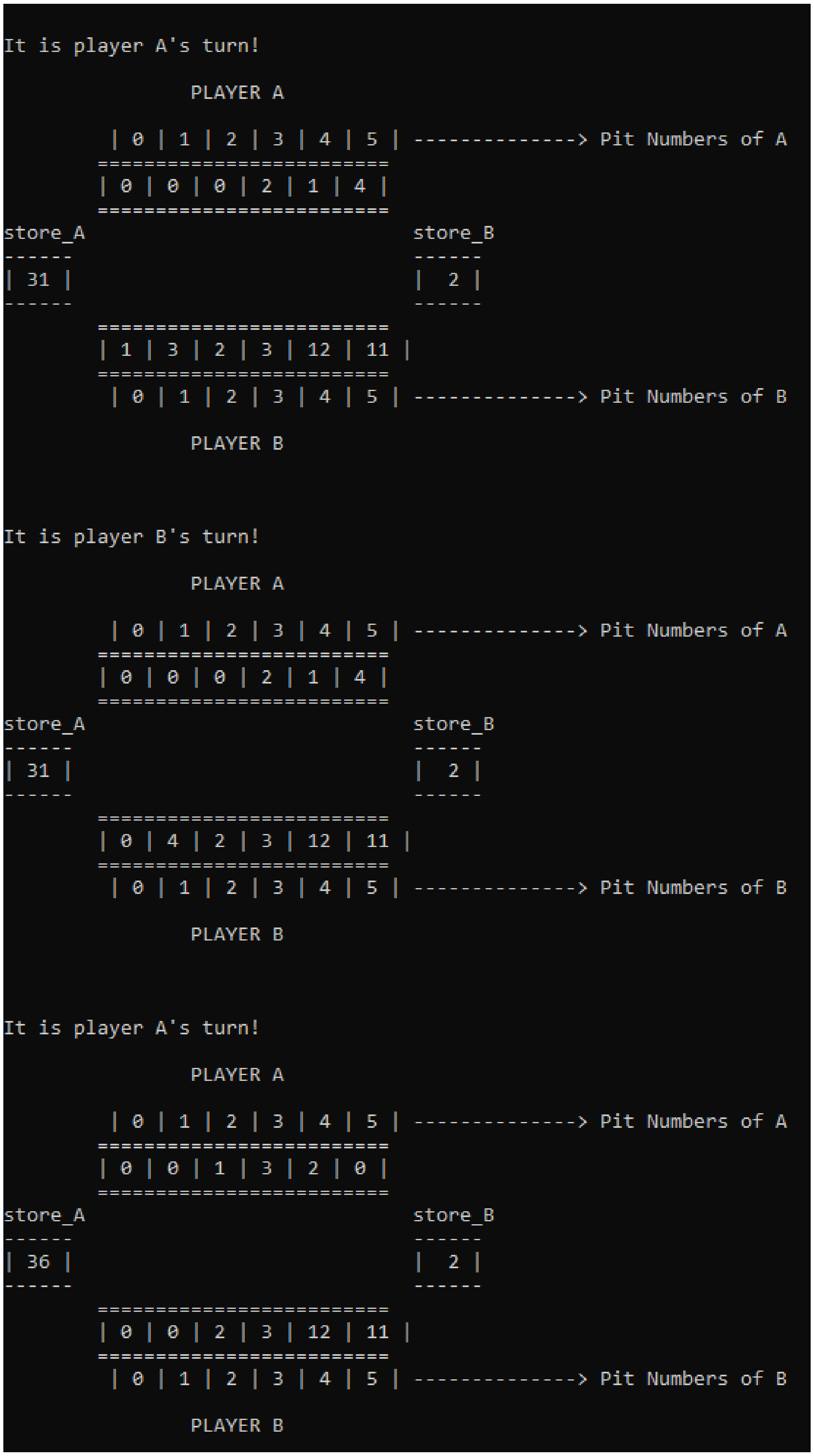
```
It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 0 | 10 | 0 | 0 | 1 | 2 |
        =========================
store_A                            store_B
------                             ------
| 16 |                             |  1 |
------                             ------
        =========================
        | 1 | 13 | 10 | 0 | 9 | 9 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 0 | 10 | 0 | 0 | 1 | 2 |
        =========================
store_A                            store_B
------                             ------
| 16 |                             |  1 |
------                             ------
        =========================
        | 0 | 14 | 10 | 0 | 9 | 9 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 1 | 0 | 0 | 0 | 2 | 3 |
        =========================
store_A                            store_B
------                             ------
| 17 |                             |  1 |
------                             ------
        =========================
        | 1 | 15 | 11 | 1 | 10 | 10 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B
```



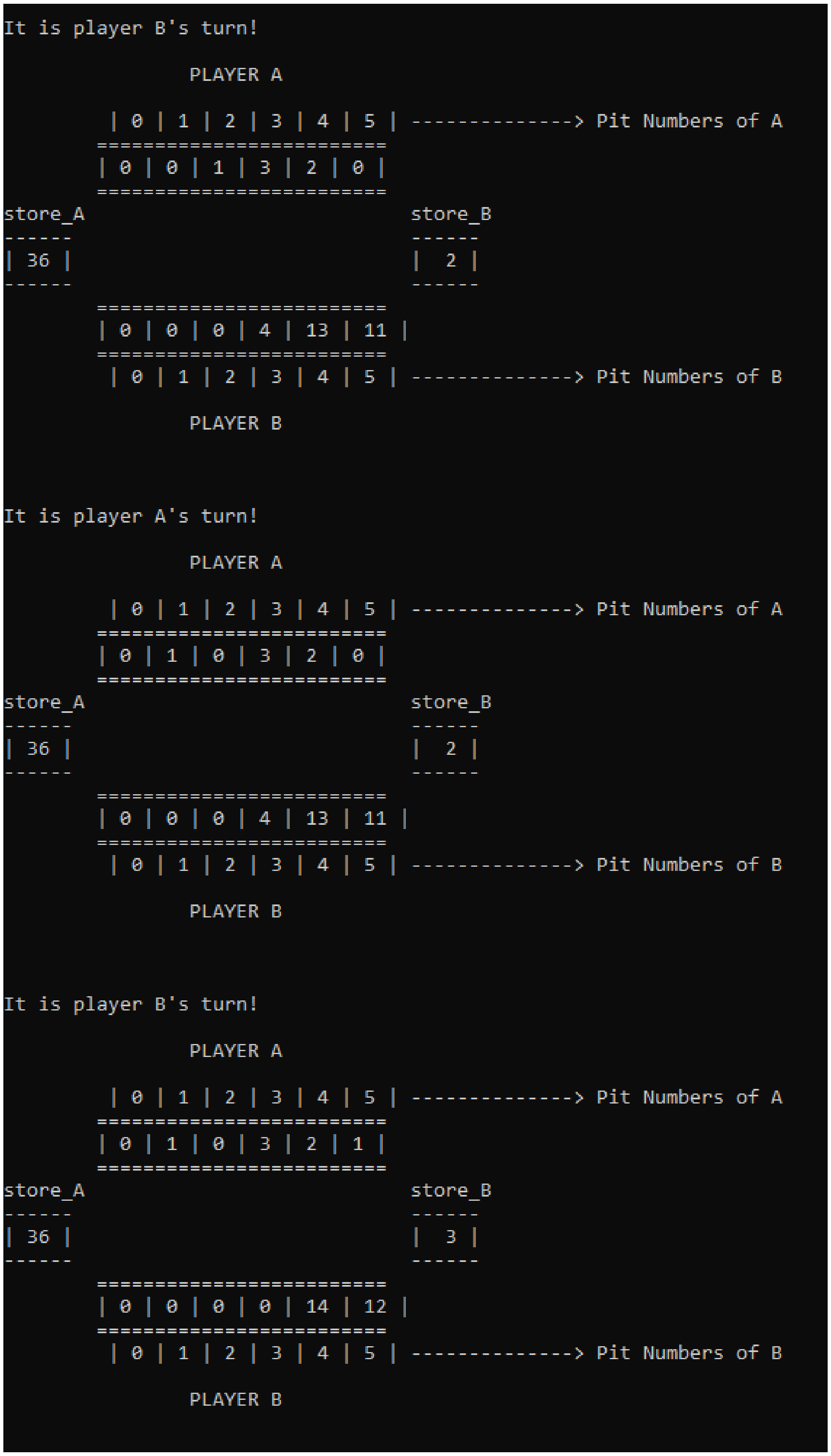
```
It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 1 | 0 | 0 | 0 | 2 | 3 |
        =========================
store_A                            store_B
------                             ------
| 17 |                             |  1 |
------                             ------
        =========================
        | 0 | 16 | 11 | 1 | 10 | 10 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 1 | 0 | 0 | 1 | 0 | 3 |
        =========================
store_A                            store_B
------                             ------
| 29 |                             |  1 |
------                             ------
        =========================
        | 0 | 16 | 0 | 1 | 10 | 10 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 2 | 1 | 1 | 2 | 1 | 4 |
        =========================
store_A                            store_B
------                             ------
| 29 |                             |  2 |
------                             ------
        =========================
        | 1 | 1 | 2 | 3 | 12 | 11 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B
```

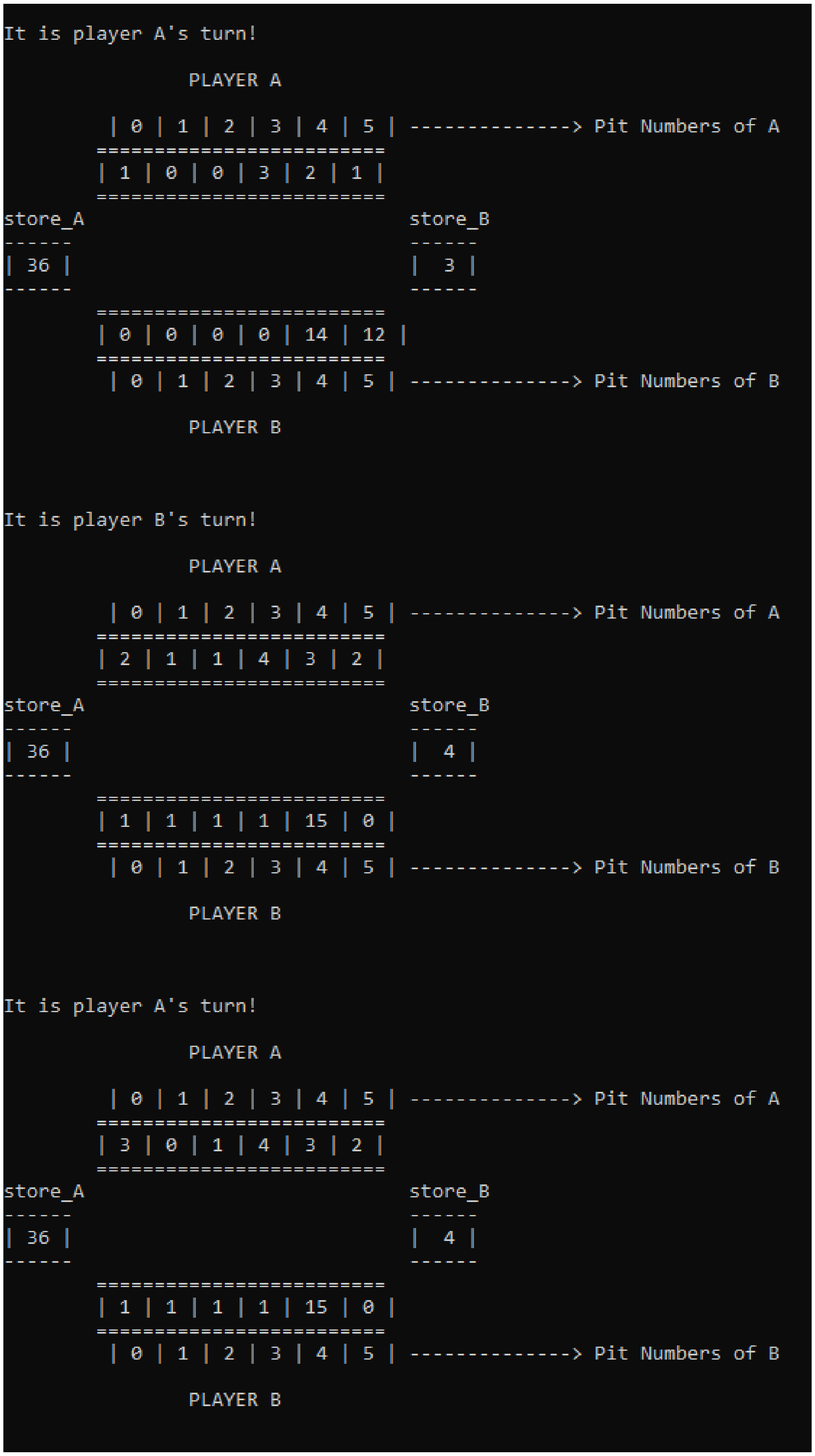


```
It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 2 | 2 | 0 | 2 | 1 | 4 |
        =========================
store_A                            store_B
------                             ------
| 29 |                             |  2 |
------                             ------
        =========================
        | 1 | 1 | 2 | 3 | 12 | 11 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 2 | 2 | 0 | 2 | 1 | 4 |
        =========================
store_A                            store_B
------                             ------
| 29 |                             |  2 |
------                             ------
        =========================
        | 0 | 2 | 2 | 3 | 12 | 11 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 3 | 0 | 0 | 2 | 1 | 4 |
        =========================
store_A                            store_B
------                             ------
| 30 |                             |  2 |
------                             ------
        =========================
        | 0 | 2 | 2 | 3 | 12 | 11 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B
```

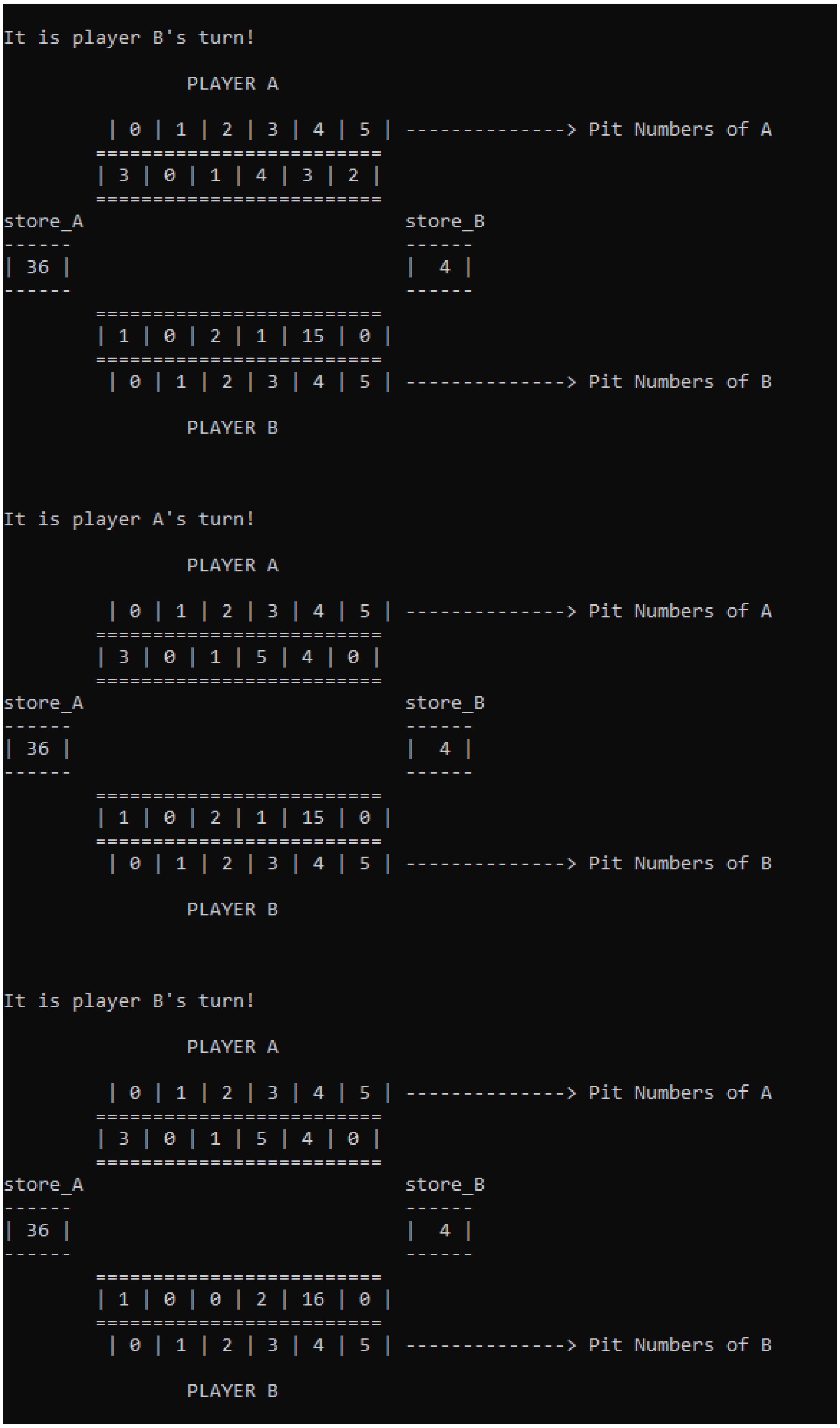


```
It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 0 | 0 | 0 | 2 | 1 | 4 |
        =========================
store_A                            store_B
------                             ------
| 31 |                             |  2 |
------                             ------
        =========================
        | 1 | 3 | 2 | 3 | 12 | 11 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 0 | 0 | 0 | 2 | 1 | 4 |
        =========================
store_A                            store_B
------                             ------
| 31 |                             |  2 |
------                             ------
        =========================
        | 0 | 4 | 2 | 3 | 12 | 11 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 0 | 0 | 1 | 3 | 2 | 0 |
        =========================
store_A                            store_B
------                             ------
| 36 |                             |  2 |
------                             ------
        =========================
        | 0 | 0 | 2 | 3 | 12 | 11 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B
```

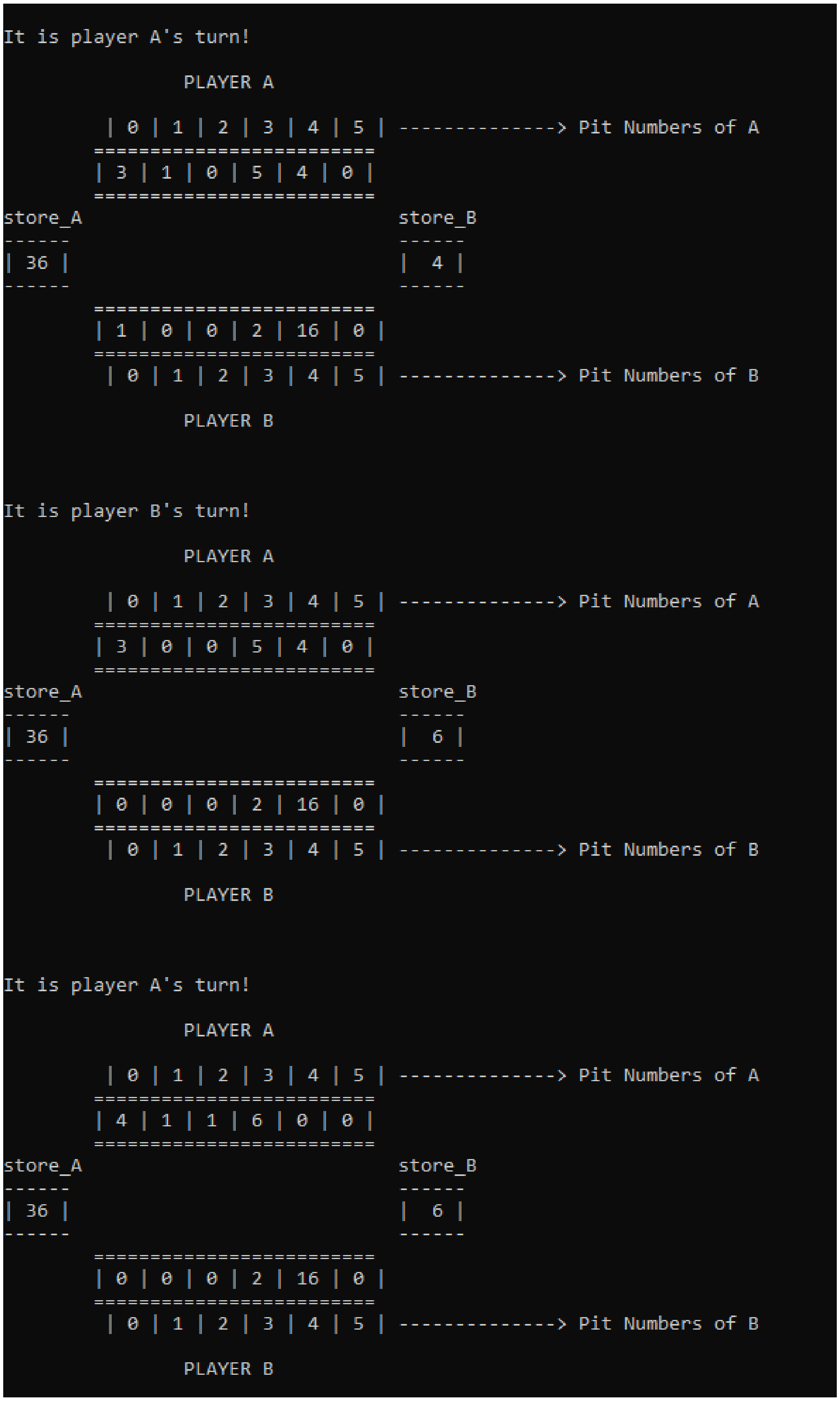

```
It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 0 | 0 | 1 | 3 | 2 | 0 |
        =========================
store_A                            store_B
------                             ------
| 36 |                             |  2 |
------                             ------
        =========================
        | 0 | 0 | 0 | 4 | 13 | 11 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 0 | 1 | 0 | 3 | 2 | 0 |
        =========================
store_A                            store_B
------                             ------
| 36 |                             |  2 |
------                             ------
        =========================
        | 0 | 0 | 0 | 4 | 13 | 11 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 0 | 1 | 0 | 3 | 2 | 1 |
        =========================
store_A                            store_B
------                             ------
| 36 |                             |  3 |
------                             ------
        =========================
        | 0 | 0 | 0 | 0 | 14 | 12 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B
```



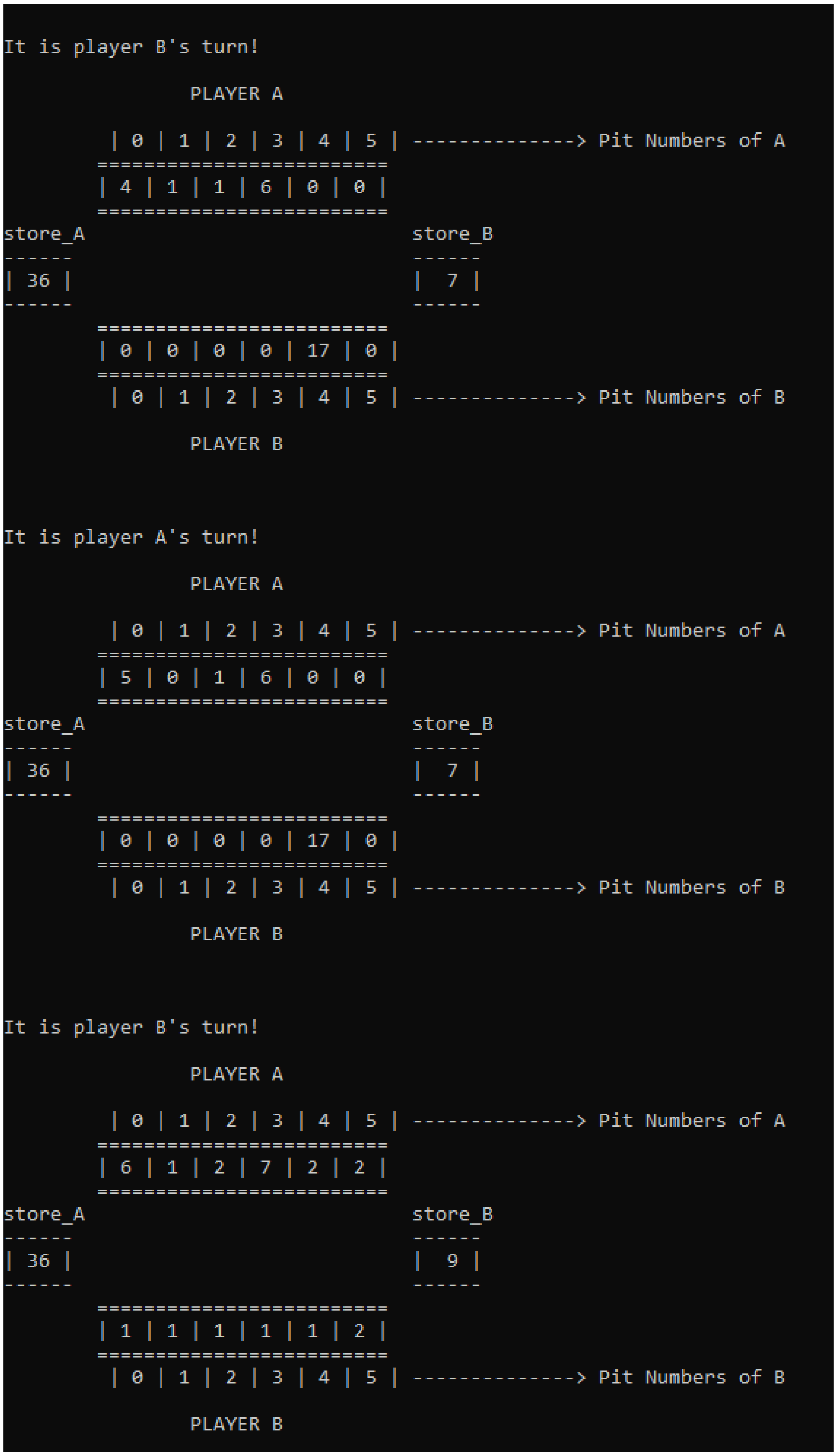
```
It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 1 | 0 | 0 | 3 | 2 | 1 |
        =========================
store_A                            store_B
------                             ------
| 36 |                             |  3 |
------                             ------
        =========================
        | 0 | 0 | 0 | 0 | 14 | 12 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 2 | 1 | 1 | 4 | 3 | 2 |
        =========================
store_A                            store_B
------                             ------
| 36 |                             |  4 |
------                             ------
        =========================
        | 1 | 1 | 1 | 1 | 15 | 0 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 3 | 0 | 1 | 4 | 3 | 2 |
        =========================
store_A                            store_B
------                             ------
| 36 |                             |  4 |
------                             ------
        =========================
        | 1 | 1 | 1 | 1 | 15 | 0 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B
```

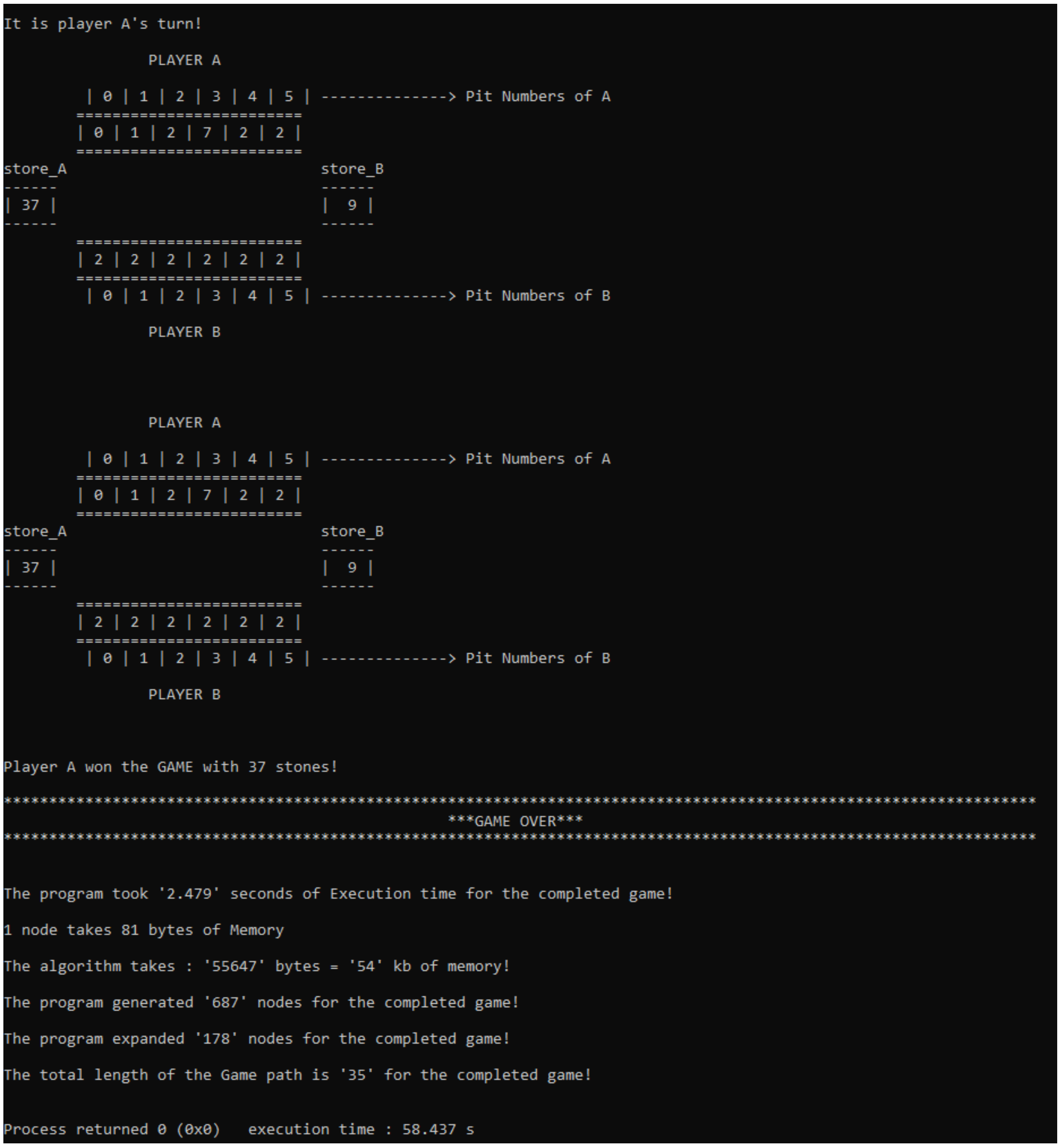


```
It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 3 | 0 | 1 | 4 | 3 | 2 |
        =========================
store_A                            store_B
------                             ------
| 36 |                             |  4 |
------                             ------
        =========================
        | 1 | 0 | 2 | 1 | 15 | 0 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 3 | 0 | 1 | 5 | 4 | 0 |
        =========================
store_A                            store_B
------                             ------
| 36 |                             |  4 |
------                             ------
        =========================
        | 1 | 0 | 2 | 1 | 15 | 0 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 3 | 0 | 1 | 5 | 4 | 0 |
        =========================
store_A                            store_B
------                             ------
| 36 |                             |  4 |
------                             ------
        =========================
        | 1 | 0 | 0 | 2 | 16 | 0 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B
```



```
It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 3 | 1 | 0 | 5 | 4 | 0 |
        =========================
store_A                            store_B
------                             ------
| 36 |                             |  4 |
------                             ------
        =========================
        | 1 | 0 | 0 | 2 | 16 | 0 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 3 | 0 | 0 | 5 | 4 | 0 |
        =========================
store_A                            store_B
------                             ------
| 36 |                             |  6 |
------                             ------
        =========================
        | 0 | 0 | 0 | 2 | 16 | 0 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 4 | 1 | 1 | 6 | 0 | 0 |
        =========================
store_A                            store_B
------                             ------
| 36 |                             |  6 |
------                             ------
        =========================
        | 0 | 0 | 0 | 2 | 16 | 0 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B
```

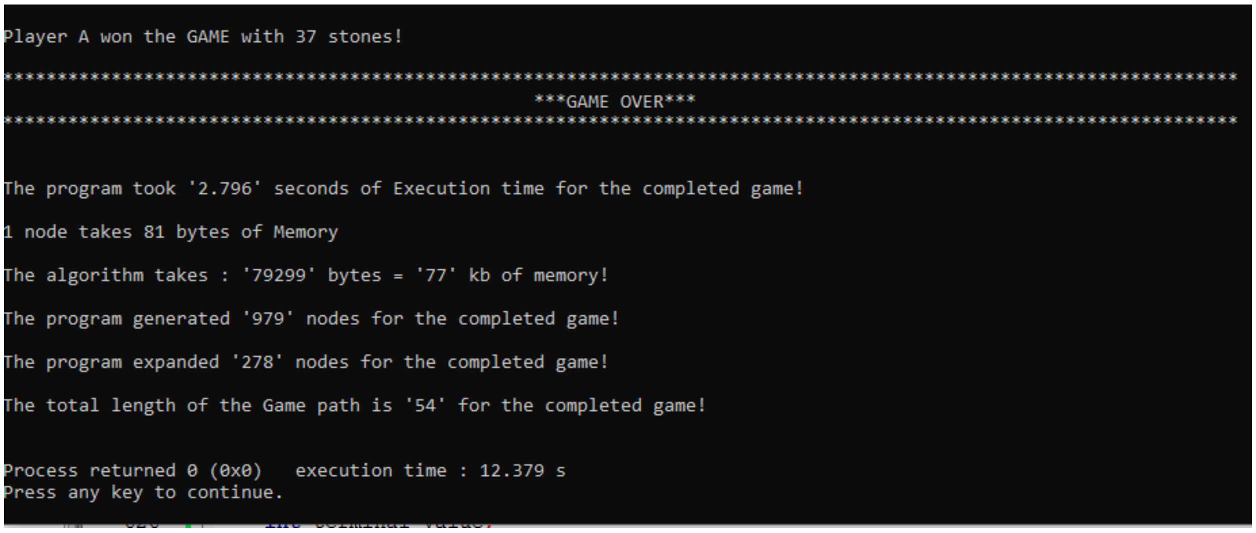

```
It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 4 | 1 | 1 | 6 | 0 | 0 |
        =========================
store_A                            store_B
------                             ------
| 36 |                             |  7 |
------                             ------
        =========================
        | 0 | 0 | 0 | 0 | 17 | 0 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 5 | 0 | 1 | 6 | 0 | 0 |
        =========================
store_A                            store_B
------                             ------
| 36 |                             |  7 |
------                             ------
        =========================
        | 0 | 0 | 0 | 0 | 17 | 0 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 6 | 1 | 2 | 7 | 2 | 2 |
        =========================
store_A                            store_B
------                             ------
| 36 |                             |  9 |
------                             ------
        =========================
        | 1 | 1 | 1 | 1 | 1 | 2 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B
```



```
It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 0 | 1 | 2 | 7 | 2 | 2 |
        =========================
store_A                            store_B
------                             ------
| 37 |                             |  9 |
------                             ------
        =========================
        | 2 | 2 | 2 | 2 | 2 | 2 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B



                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 0 | 1 | 2 | 7 | 2 | 2 |
        =========================
store_A                            store_B
------                             ------
| 37 |                             |  9 |
------                             ------
        =========================
        | 2 | 2 | 2 | 2 | 2 | 2 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


Player A won the GAME with 37 stones!
***********************************************************************************************************************
                                                 ***GAME OVER***
***********************************************************************************************************************


The program took '2.479' seconds of Execution time for the completed game!

1 node takes 81 bytes of Memory

The algorithm takes : '55647' bytes = '54' kb of memory!

The program generated '687' nodes for the completed game!

The program expanded '178' nodes for the completed game!

The total length of the Game path is '35' for the completed game!


Process returned 0 (0x0)   execution time : 58.437 s
```

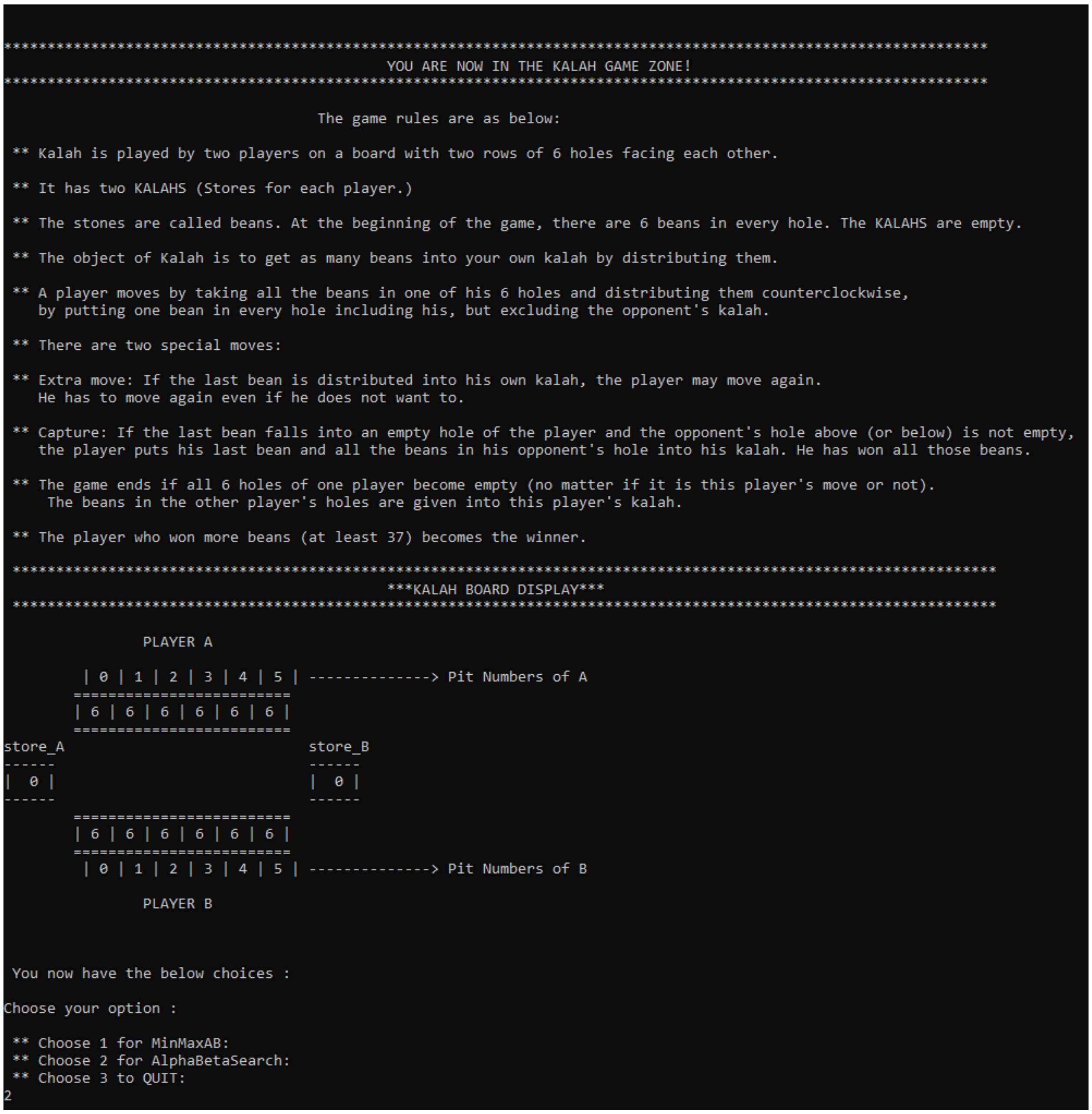
Below is the sample run 2: Evaluation function 2, Depth 4, MinMaxAB algorithm:

(Just last step shown)

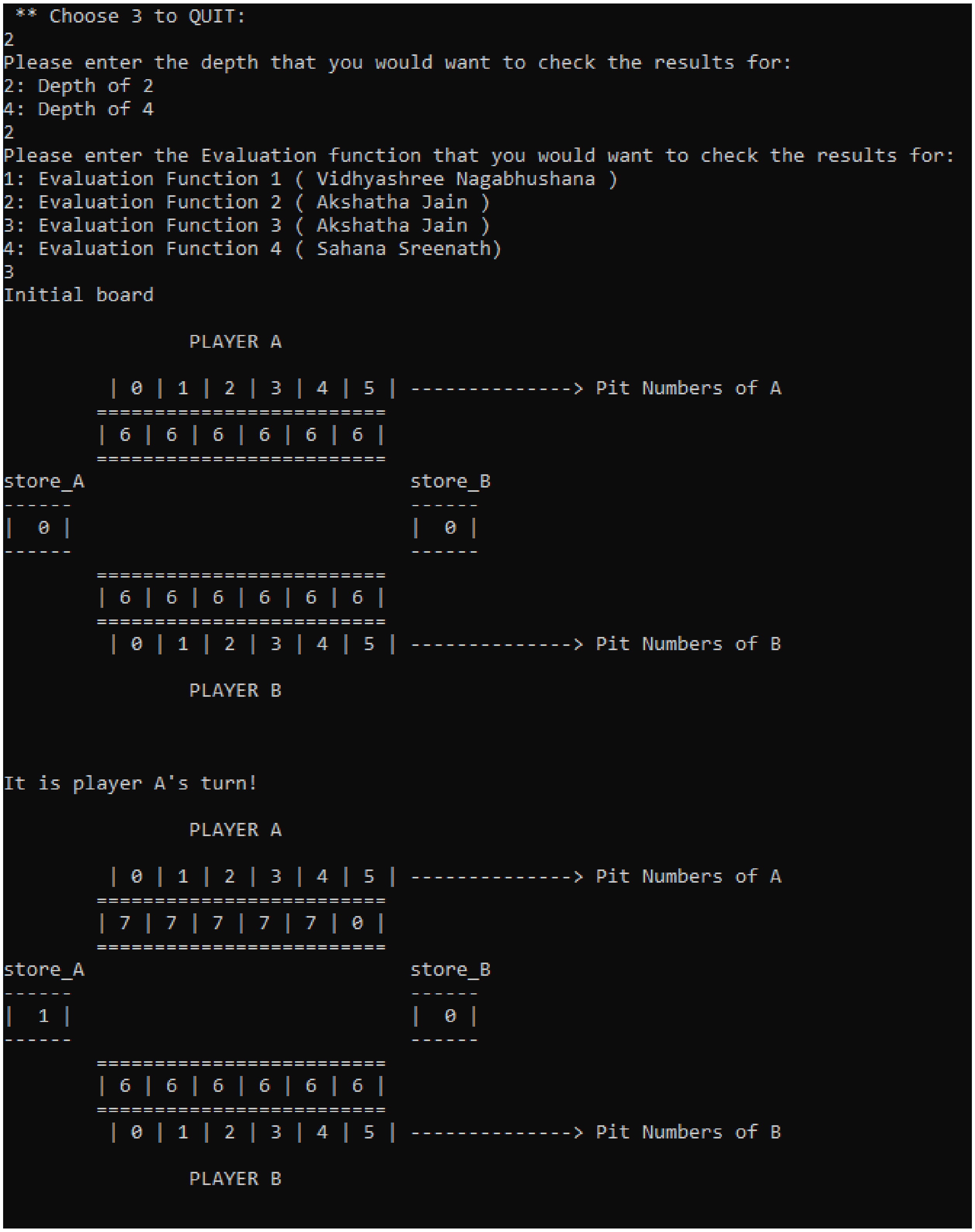


```
Player A won the GAME with 37 stones!
**************************************************************************************************************************
                                                    ***GAME OVER***
**************************************************************************************************************************


The program took '2.796' seconds of Execution time for the completed game!

1 node takes 81 bytes of Memory

The algorithm takes : '79299' bytes = '77' kb of memory!

The program generated '979' nodes for the completed game!

The program expanded '278' nodes for the completed game!

The total length of the Game path is '54' for the completed game!


Process returned 0 (0x0)   execution time : 12.379 s
Press any key to continue.
```

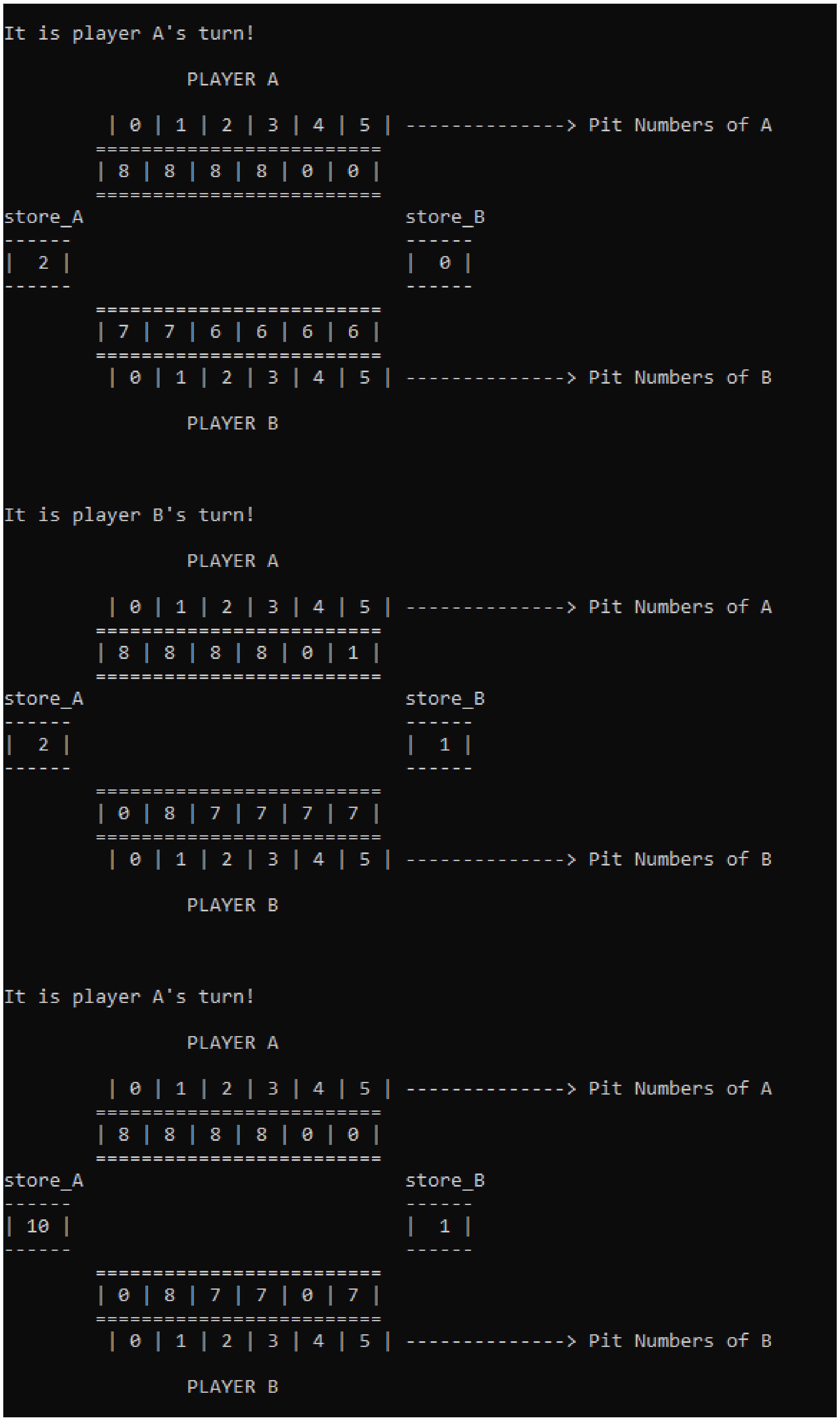
Below is the sample run 3: Evaluation function 3, Depth 2, AlphaBetaSearch algorithm:



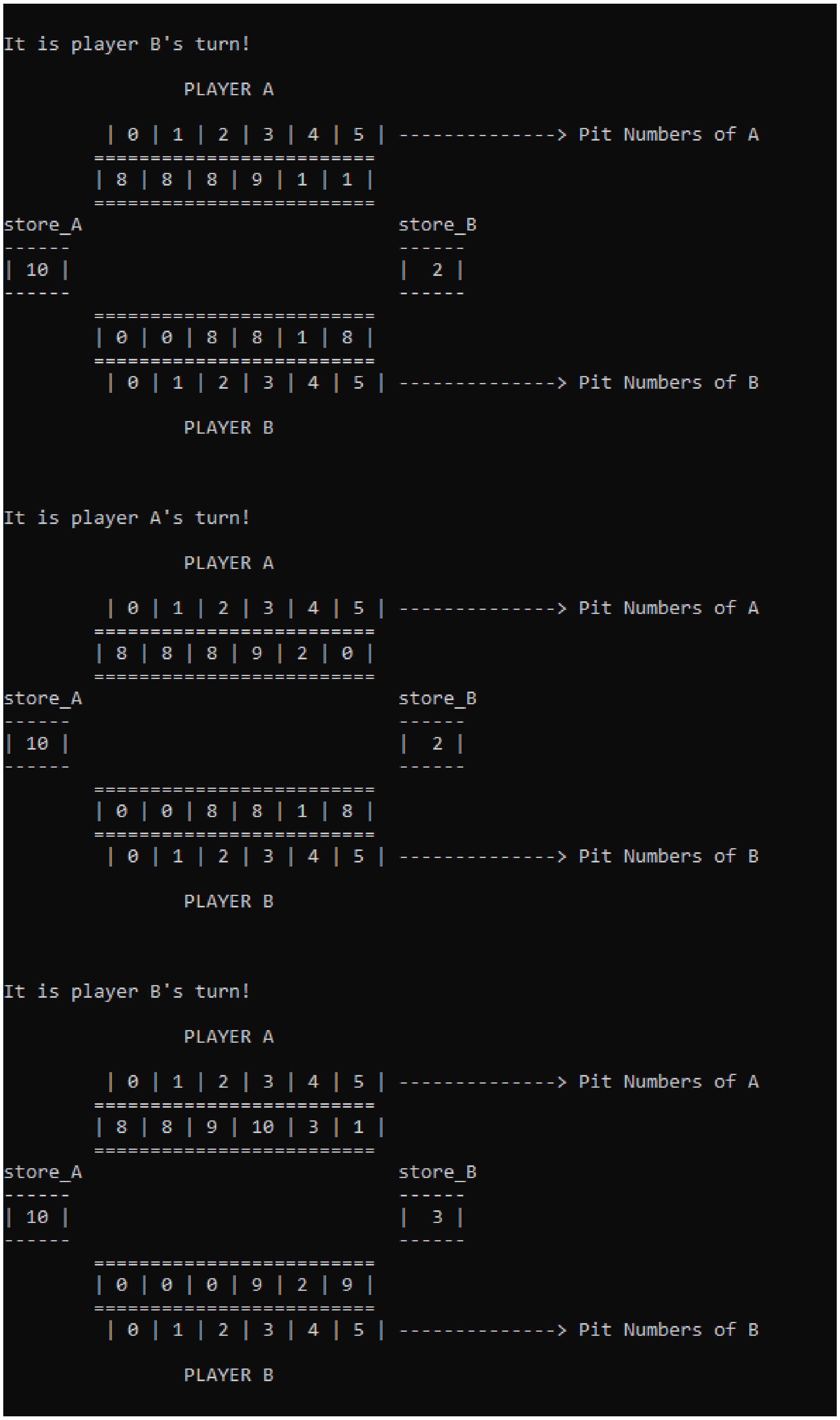
```
*******************************************************************************************************************
                                            YOU ARE NOW IN THE KALAH GAME ZONE!
*******************************************************************************************************************

                                    The game rules are as below:

 ** Kalah is played by two players on a board with two rows of 6 holes facing each other.

 ** It has two KALAHS (Stores for each player.)

 ** The stones are called beans. At the beginning of the game, there are 6 beans in every hole. The KALAHS are empty.

 ** The object of Kalah is to get as many beans into your own kalah by distributing them.

 ** A player moves by taking all the beans in one of his 6 holes and distributing them counterclockwise,
    by putting one bean in every hole including his, but excluding the opponent's kalah.

 ** There are two special moves:

 ** Extra move: If the last bean is distributed into his own kalah, the player may move again.
    He has to move again even if he does not want to.

 ** Capture: If the last bean falls into an empty hole of the player and the opponent's hole above (or below) is not empty,
    the player puts his last bean and all the beans in his opponent's hole into his kalah. He has won all those beans.

 ** The game ends if all 6 holes of one player become empty (no matter if it is this player's move or not).
     The beans in the other player's holes are given into this player's kalah.

 ** The player who won more beans (at least 37) becomes the winner.

 *******************************************************************************************************************
                                            ***KALAH BOARD DISPLAY***
 *******************************************************************************************************************

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 6 | 6 | 6 | 6 | 6 | 6 |
        =========================
store_A                            store_B
------                             ------
|  0 |                             |  0 |
------                             ------
        =========================
        | 6 | 6 | 6 | 6 | 6 | 6 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


 You now have the below choices :

Choose your option :

 ** Choose 1 for MinMaxAB:
 ** Choose 2 for AlphaBetaSearch:
 ** Choose 3 to QUIT:
2
```

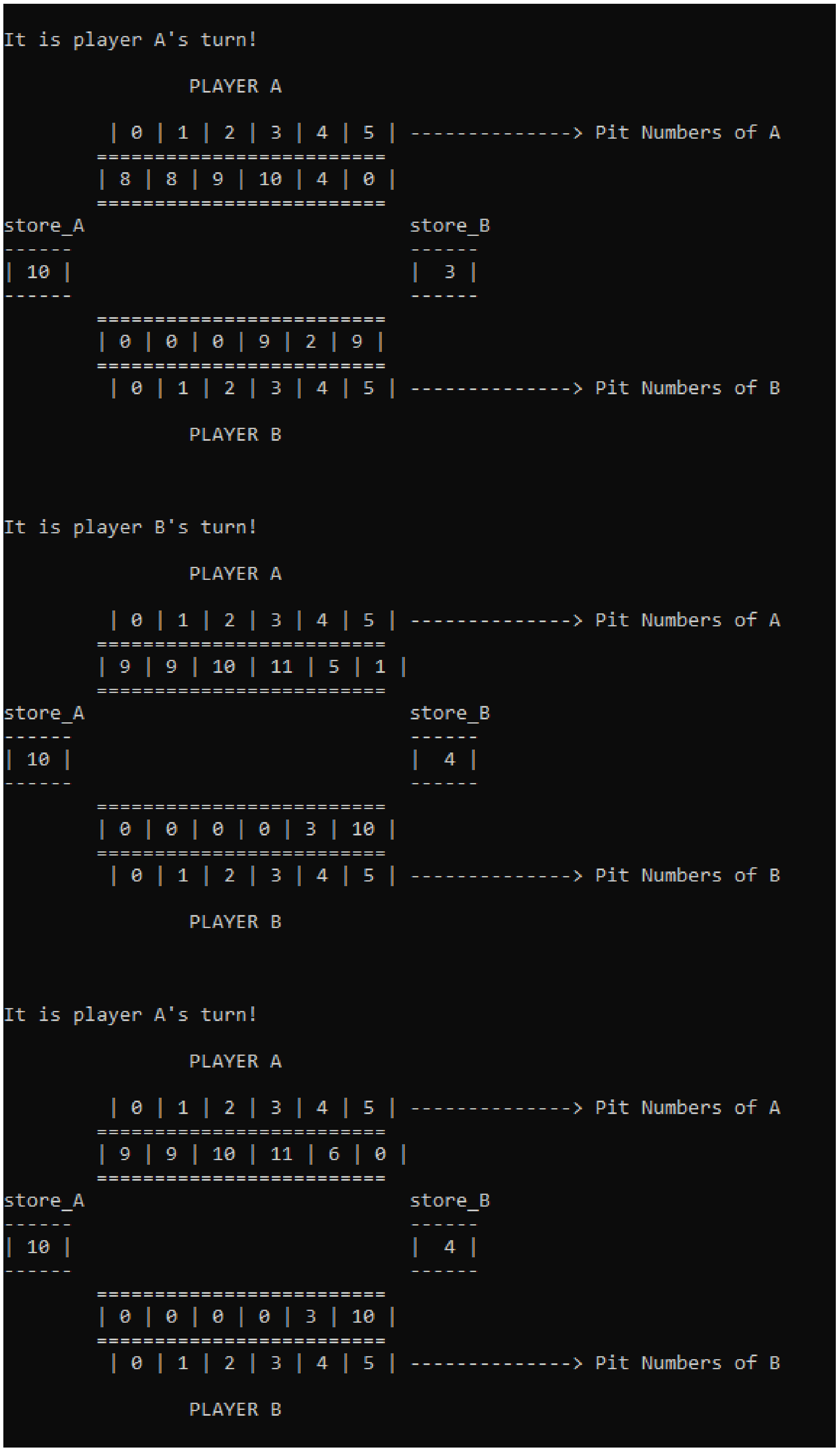


```
 ** Choose 3 to QUIT:
2
Please enter the depth that you would want to check the results for:
2: Depth of 2
4: Depth of 4
2
Please enter the Evaluation function that you would want to check the results for:
1: Evaluation Function 1 ( Vidhyashree Nagabhushana )
2: Evaluation Function 2 ( Akshatha Jain )
3: Evaluation Function 3 ( Akshatha Jain )
4: Evaluation Function 4 ( Sahana Sreenath)
3
Initial board

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 6 | 6 | 6 | 6 | 6 | 6 |
        =========================
store_A                            store_B
------                             ------
|  0 |                             |  0 |
------                             ------
        =========================
        | 6 | 6 | 6 | 6 | 6 | 6 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B



It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 7 | 7 | 7 | 7 | 7 | 0 |
        =========================
store_A                            store_B
------                             ------
|  1 |                             |  0 |
------                             ------
        =========================
        | 6 | 6 | 6 | 6 | 6 | 6 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B
```

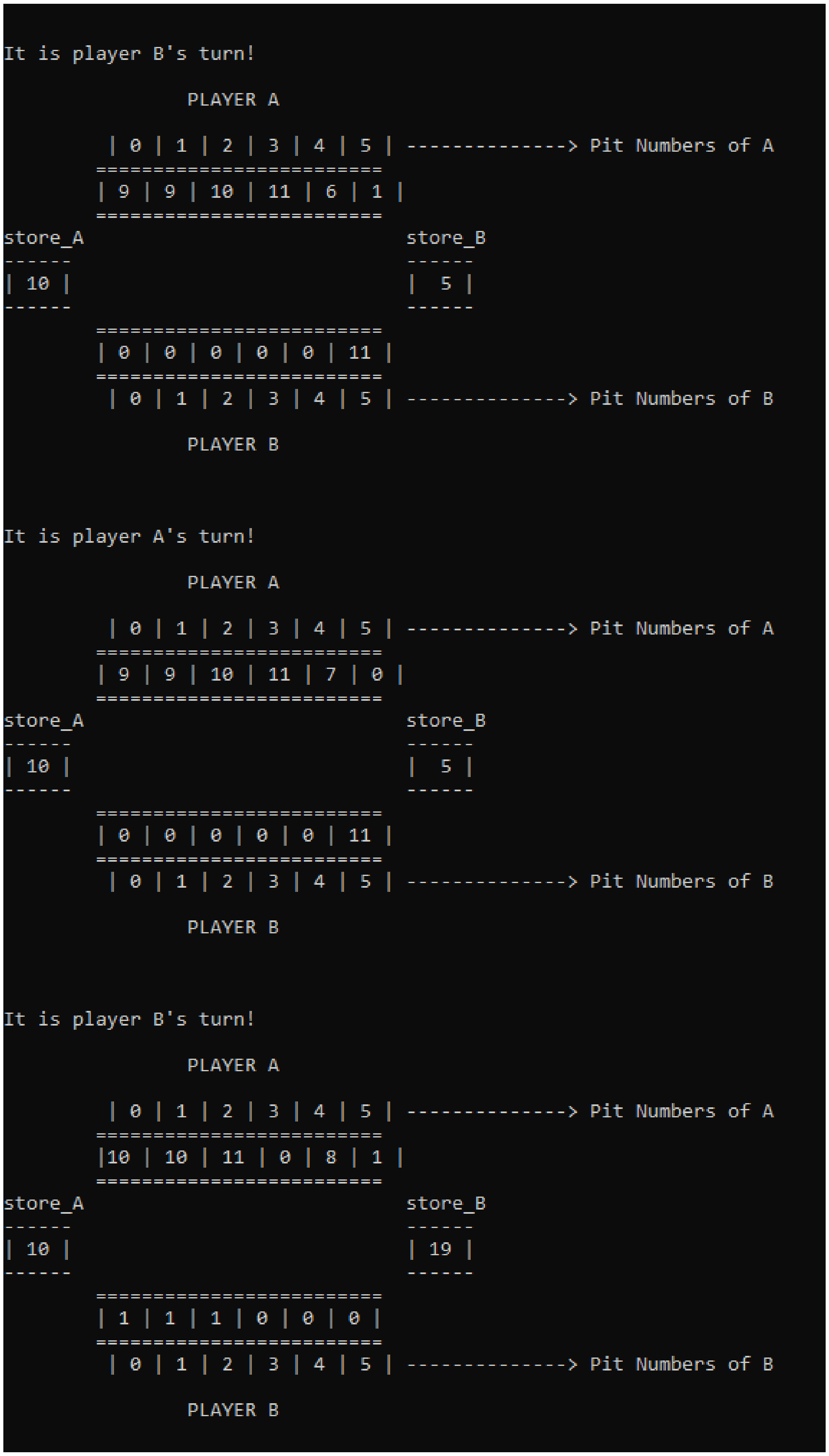


```
It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 8 | 8 | 8 | 8 | 0 | 0 |
        =========================
store_A                            store_B
------                             ------
|  2 |                             |  0 |
------                             ------
        =========================
        | 7 | 7 | 6 | 6 | 6 | 6 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 8 | 8 | 8 | 8 | 0 | 1 |
        =========================
store_A                            store_B
------                             ------
|  2 |                             |  1 |
------                             ------
        =========================
        | 0 | 8 | 7 | 7 | 7 | 7 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 8 | 8 | 8 | 8 | 0 | 0 |
        =========================
store_A                            store_B
------                             ------
| 10 |                             |  1 |
------                             ------
        =========================
        | 0 | 8 | 7 | 7 | 0 | 7 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B
```

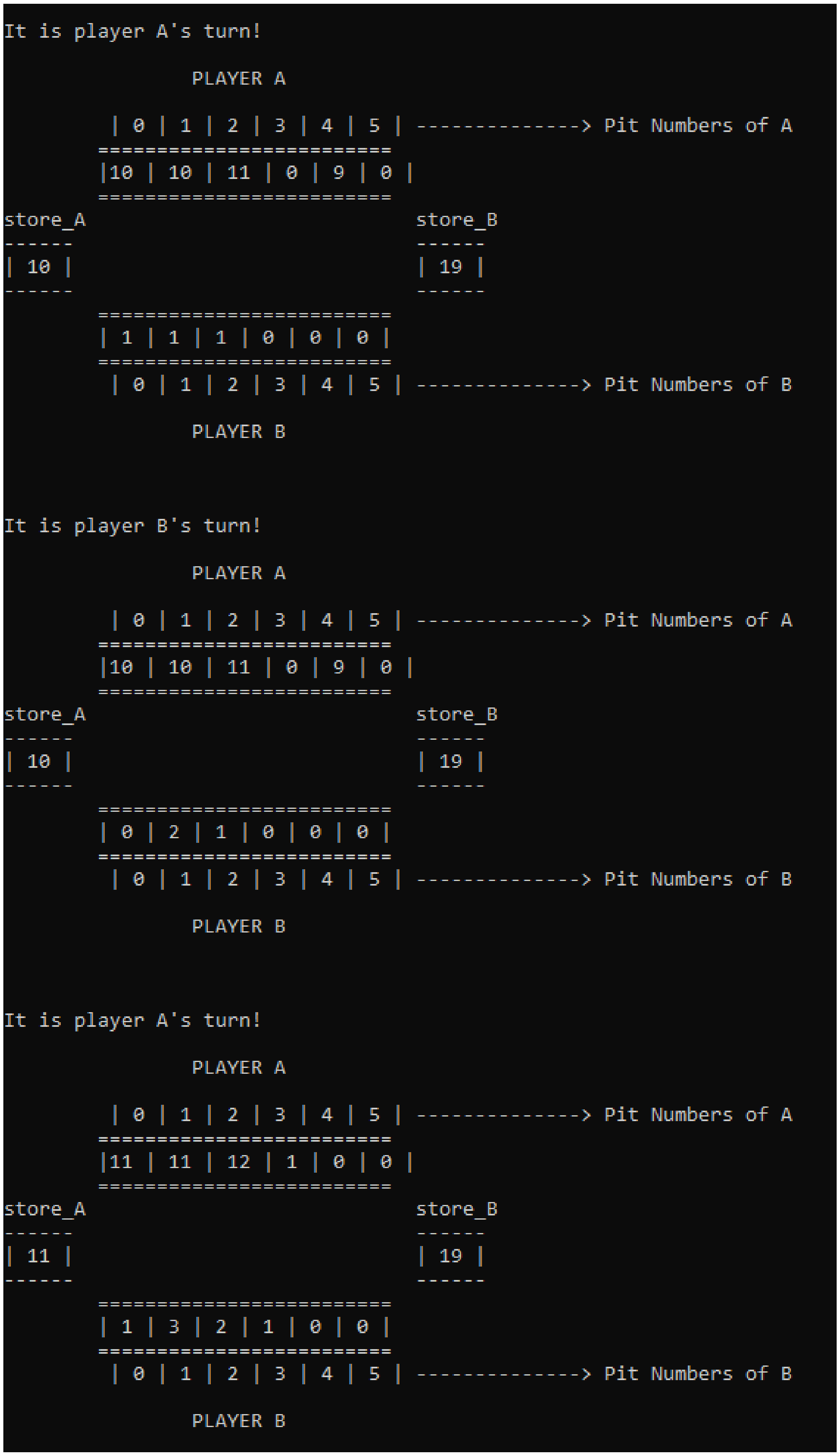

```
It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 8 | 8 | 8 | 9 | 1 | 1 |
        =========================
store_A                            store_B
------                             ------
| 10 |                             |  2 |
------                             ------
        =========================
        | 0 | 0 | 8 | 8 | 1 | 8 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 8 | 8 | 8 | 9 | 2 | 0 |
        =========================
store_A                            store_B
------                             ------
| 10 |                             |  2 |
------                             ------
        =========================
        | 0 | 0 | 8 | 8 | 1 | 8 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 8 | 8 | 9 | 10 | 3 | 1 |
        =========================
store_A                            store_B
------                             ------
| 10 |                             |  3 |
------                             ------
        =========================
        | 0 | 0 | 0 | 9 | 2 | 9 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B
```



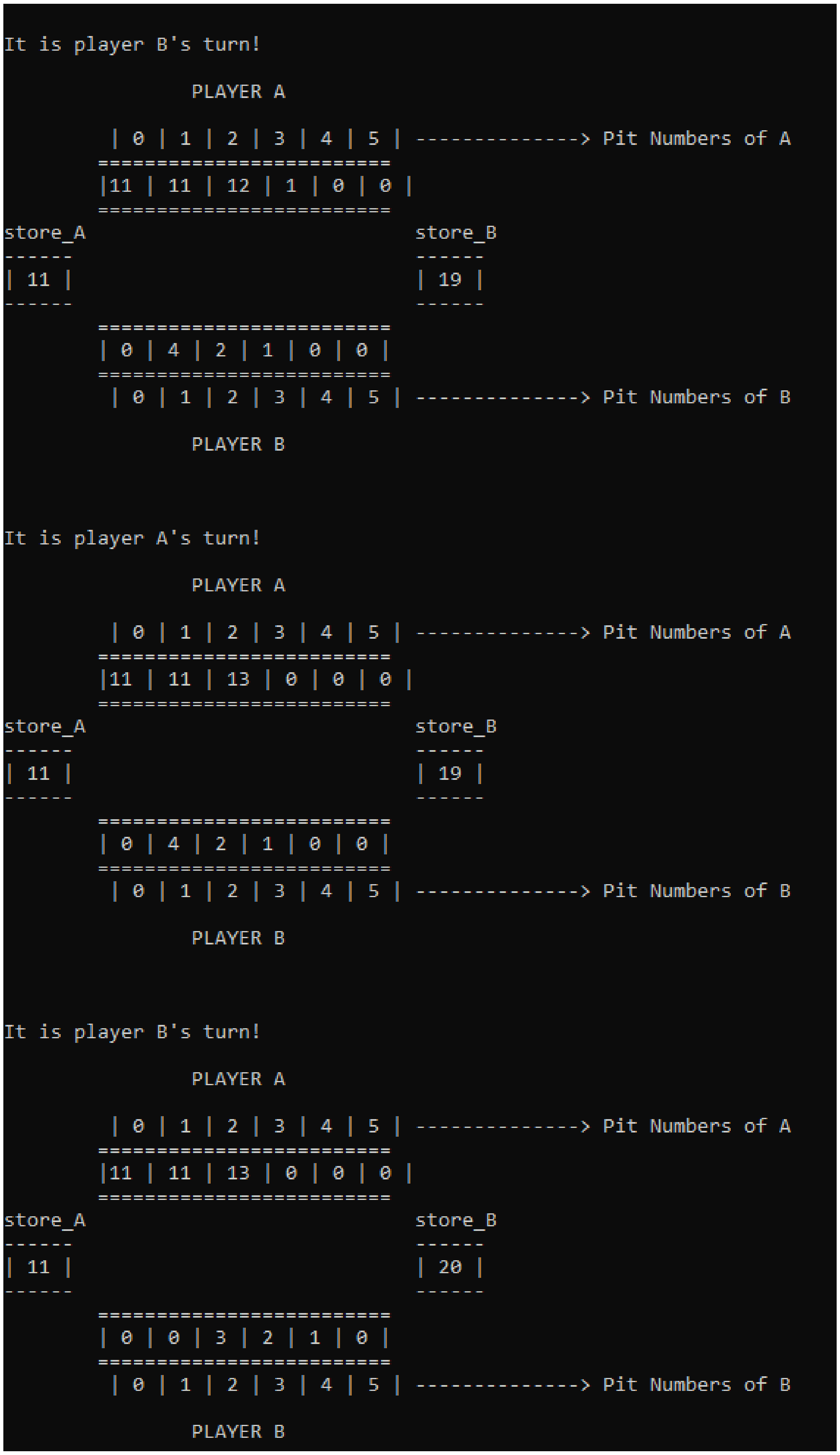
```
It is player A's turn!

               PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 8 | 8 | 9 | 10 | 4 | 0 |
        =========================
store_A                           store_B
------                            ------
| 10 |                            |  3 |
------                            ------
        =========================
        | 0 | 0 | 0 | 9 | 2 | 9 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

               PLAYER B


It is player B's turn!

               PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 9 | 9 | 10 | 11 | 5 | 1 |
        =========================
store_A                           store_B
------                            ------
| 10 |                            |  4 |
------                            ------
        =========================
        | 0 | 0 | 0 | 0 | 3 | 10 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

               PLAYER B


It is player A's turn!

               PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 9 | 9 | 10 | 11 | 6 | 0 |
        =========================
store_A                           store_B
------                            ------
| 10 |                            |  4 |
------                            ------
        =========================
        | 0 | 0 | 0 | 0 | 3 | 10 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

               PLAYER B
```



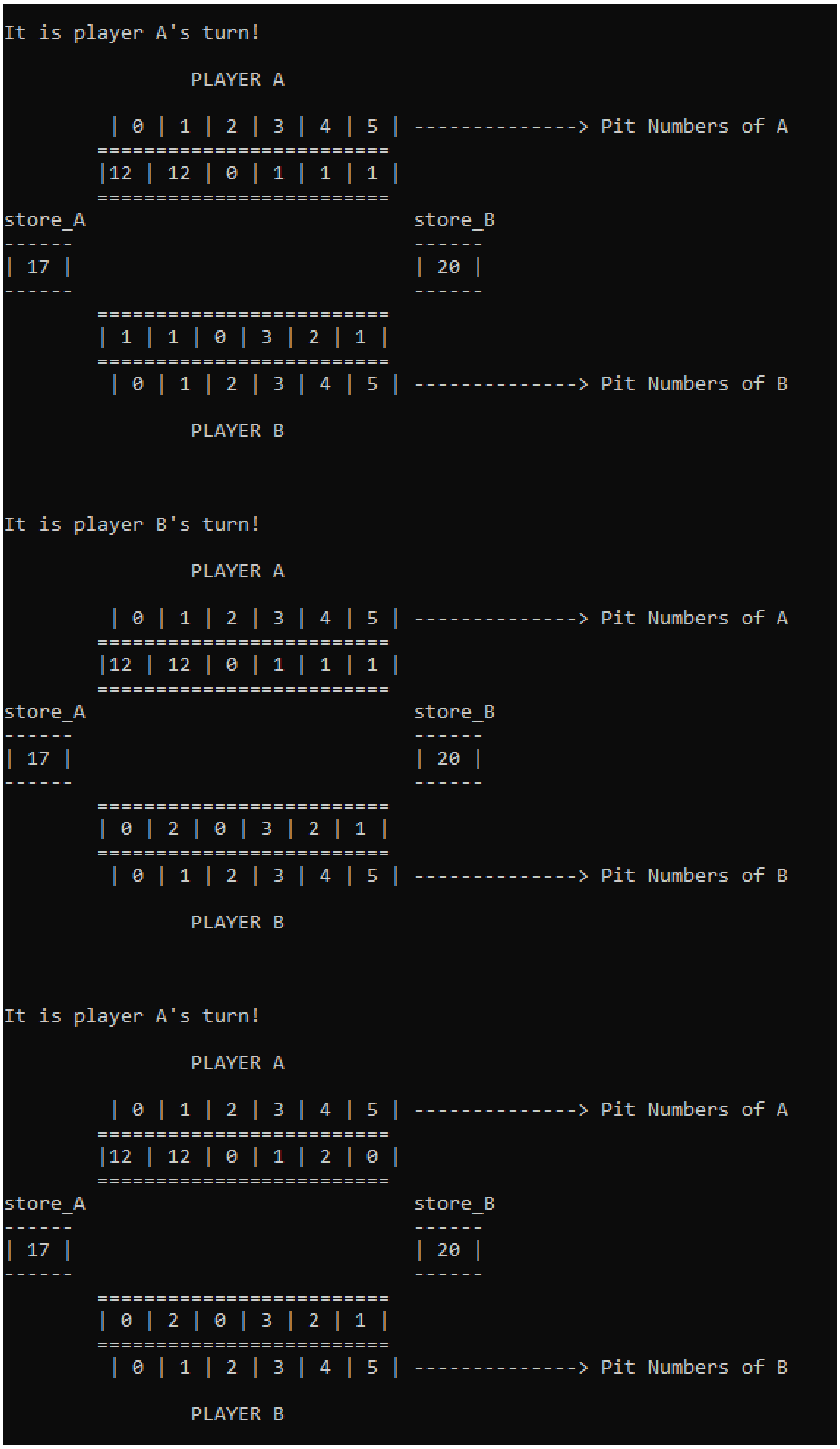
```
It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 9 | 9 | 10 | 11 | 6 | 1 |
        =========================
store_A                            store_B
------                             ------
| 10 |                             |  5 |
------                             ------
        =========================
        | 0 | 0 | 0 | 0 | 0 | 11 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 9 | 9 | 10 | 11 | 7 | 0 |
        =========================
store_A                            store_B
------                             ------
| 10 |                             |  5 |
------                             ------
        =========================
        | 0 | 0 | 0 | 0 | 0 | 11 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        |10 | 10 | 11 | 0 | 8 | 1 |
        =========================
store_A                            store_B
------                             ------
| 10 |                             | 19 |
------                             ------
        =========================
        | 1 | 1 | 1 | 0 | 0 | 0 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B
```

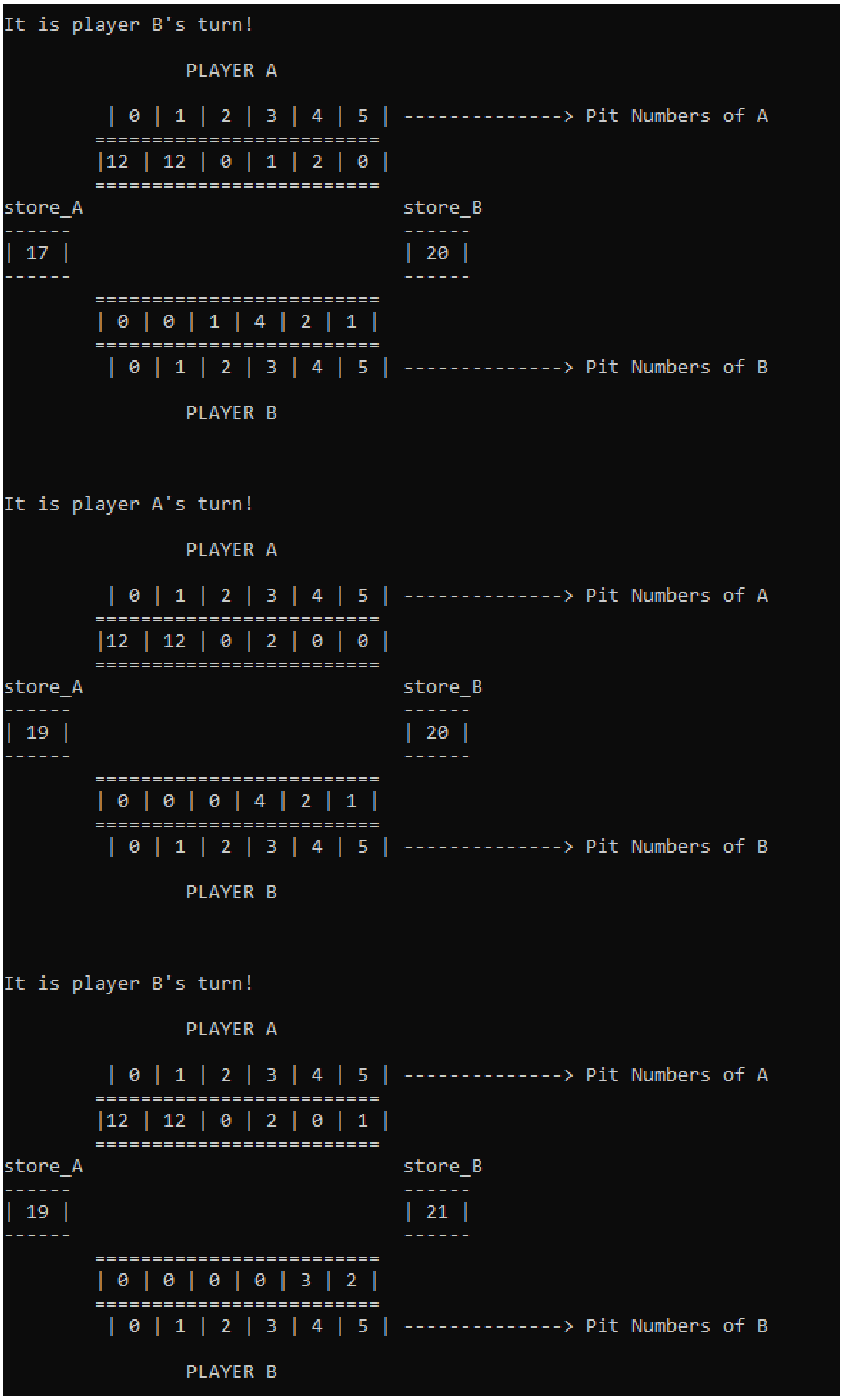


```
It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        |10 | 10 | 11 | 0 | 9 | 0 |
        =========================
store_A                          store_B
------                           ------
| 10 |                           | 19 |
------                           ------
        =========================
        | 1 | 1 | 1 | 0 | 0 | 0 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        |10 | 10 | 11 | 0 | 9 | 0 |
        =========================
store_A                          store_B
------                           ------
| 10 |                           | 19 |
------                           ------
        =========================
        | 0 | 2 | 1 | 0 | 0 | 0 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        |11 | 11 | 12 | 1 | 0 | 0 |
        =========================
store_A                          store_B
------                           ------
| 11 |                           | 19 |
------                           ------
        =========================
        | 1 | 3 | 2 | 1 | 0 | 0 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B
```

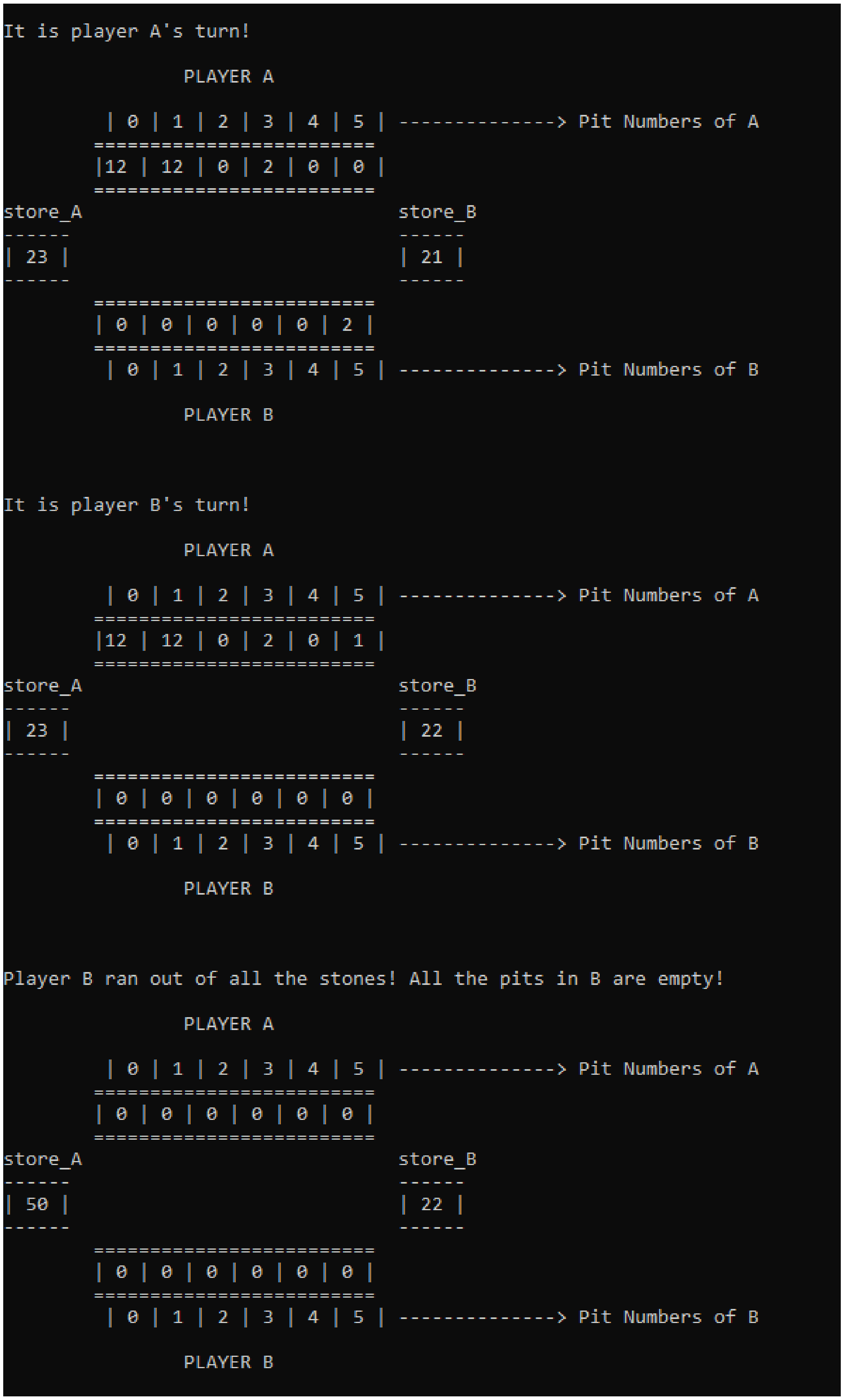


```
It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        |11 | 11 | 12 | 1 | 0 | 0 |
        =========================
store_A                             store_B
------                              ------
| 11 |                              | 19 |
------                              ------
        =========================
        | 0 | 4 | 2 | 1 | 0 | 0 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        |11 | 11 | 13 | 0 | 0 | 0 |
        =========================
store_A                             store_B
------                              ------
| 11 |                              | 19 |
------                              ------
        =========================
        | 0 | 4 | 2 | 1 | 0 | 0 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        |11 | 11 | 13 | 0 | 0 | 0 |
        =========================
store_A                             store_B
------                              ------
| 11 |                              | 20 |
------                              ------
        =========================
        | 0 | 0 | 3 | 2 | 1 | 0 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B
```

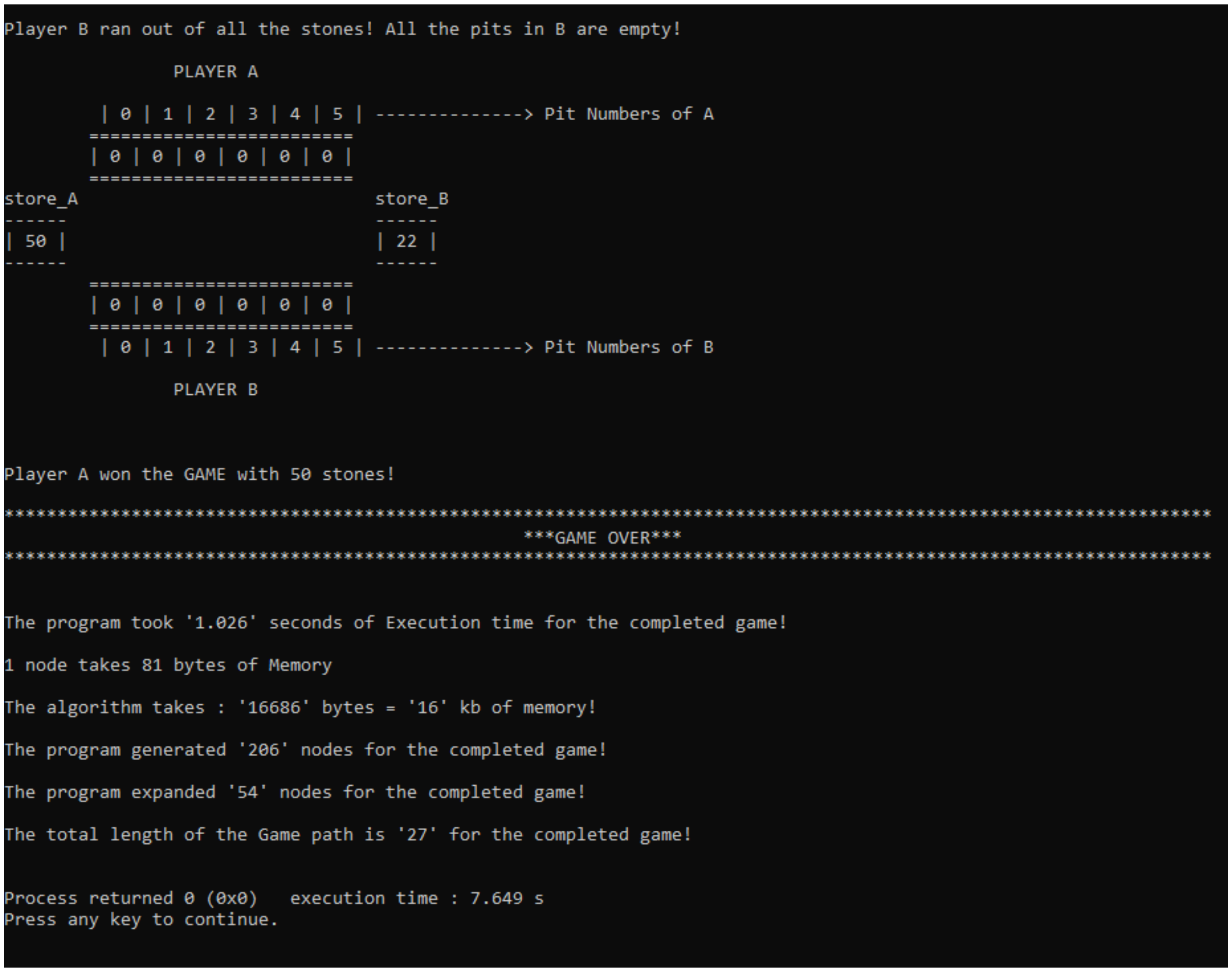

```
It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        |12 | 12 | 0 | 1 | 1 | 1 |
        =========================
store_A                            store_B
------                             ------
| 17 |                             | 20 |
------                             ------
        =========================
        | 1 | 1 | 0 | 3 | 2 | 1 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        |12 | 12 | 0 | 1 | 1 | 1 |
        =========================
store_A                            store_B
------                             ------
| 17 |                             | 20 |
------                             ------
        =========================
        | 0 | 2 | 0 | 3 | 2 | 1 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        |12 | 12 | 0 | 1 | 2 | 0 |
        =========================
store_A                            store_B
------                             ------
| 17 |                             | 20 |
------                             ------
        =========================
        | 0 | 2 | 0 | 3 | 2 | 1 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B
```



```
It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        |12 | 12 | 0 | 1 | 2 | 0 |
        =========================
store_A                            store_B
------                             ------
| 17 |                             | 20 |
------                             ------
        =========================
        | 0 | 0 | 1 | 4 | 2 | 1 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        |12 | 12 | 0 | 2 | 0 | 0 |
        =========================
store_A                            store_B
------                             ------
| 19 |                             | 20 |
------                             ------
        =========================
        | 0 | 0 | 0 | 4 | 2 | 1 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        |12 | 12 | 0 | 2 | 0 | 1 |
        =========================
store_A                            store_B
------                             ------
| 19 |                             | 21 |
------                             ------
        =========================
        | 0 | 0 | 0 | 0 | 3 | 2 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B
```



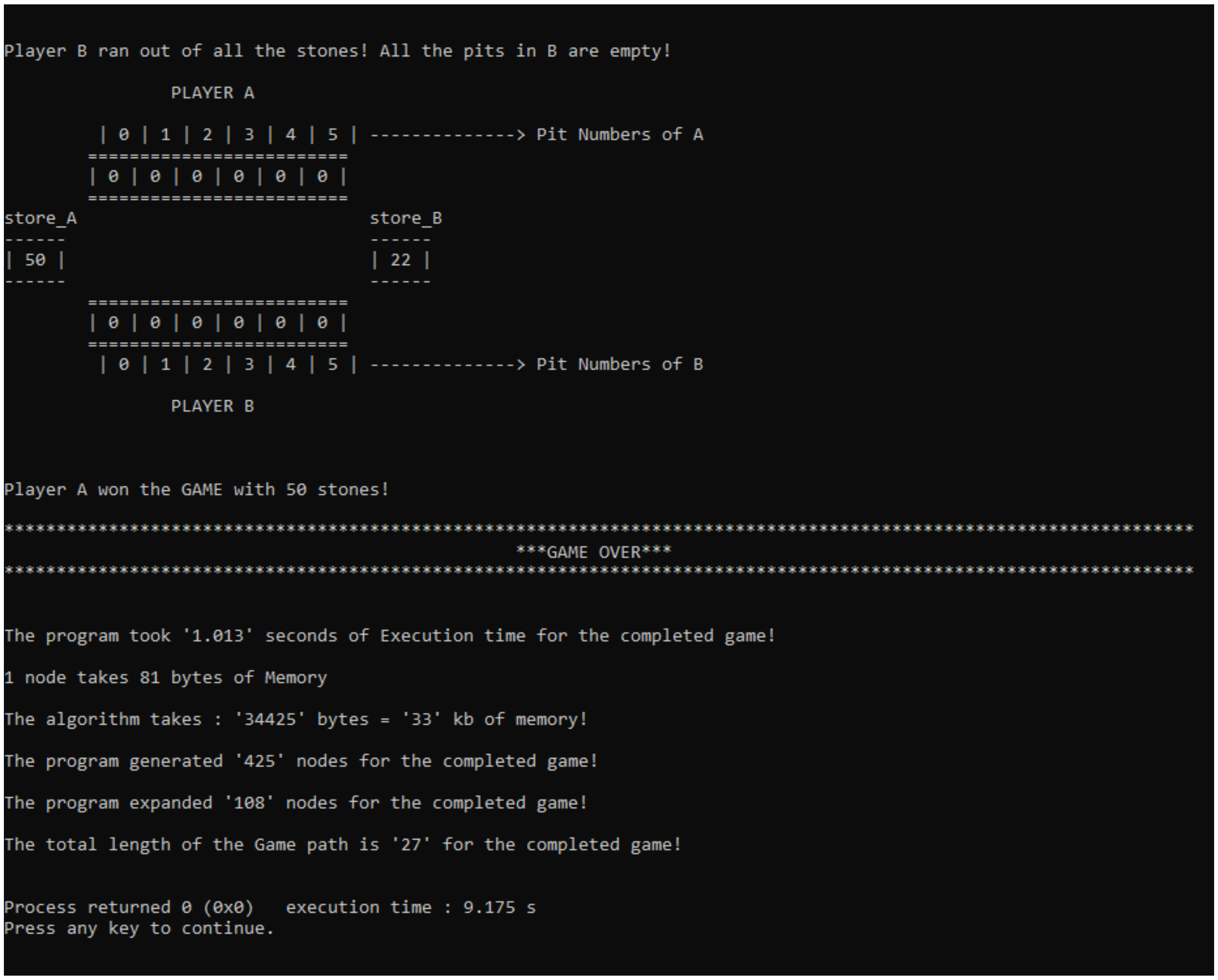
```
It is player A's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        |12 | 12 | 0 | 2 | 0 | 0 |
        =========================
store_A                            store_B
------                             ------
| 23 |                             | 21 |
------                             ------
        =========================
        | 0 | 0 | 0 | 0 | 0 | 2 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B



It is player B's turn!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        |12 | 12 | 0 | 2 | 0 | 1 |
        =========================
store_A                            store_B
------                             ------
| 23 |                             | 22 |
------                             ------
        =========================
        | 0 | 0 | 0 | 0 | 0 | 0 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B



Player B ran out of all the stones! All the pits in B are empty!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 0 | 0 | 0 | 0 | 0 | 0 |
        =========================
store_A                            store_B
------                             ------
| 50 |                             | 22 |
------                             ------
        =========================
        | 0 | 0 | 0 | 0 | 0 | 0 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B
```



```
Player B ran out of all the stones! All the pits in B are empty!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 0 | 0 | 0 | 0 | 0 | 0 |
        =========================
store_A                             store_B
------                              ------
| 50 |                              | 22 |
------                              ------
        =========================
        | 0 | 0 | 0 | 0 | 0 | 0 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


Player A won the GAME with 50 stones!

*************************************************************************************************************************
                                                  ***GAME OVER***
*************************************************************************************************************************


The program took '1.026' seconds of Execution time for the completed game!

1 node takes 81 bytes of Memory

The algorithm takes : '16686' bytes = '16' kb of memory!

The program generated '206' nodes for the completed game!

The program expanded '54' nodes for the completed game!

The total length of the Game path is '27' for the completed game!


Process returned 0 (0x0)   execution time : 7.649 s
Press any key to continue.
```

Below is the sample run 4: Evaluation function 4, Depth 4, AlphaBetaSearch algorithm:

(Just last step shown)

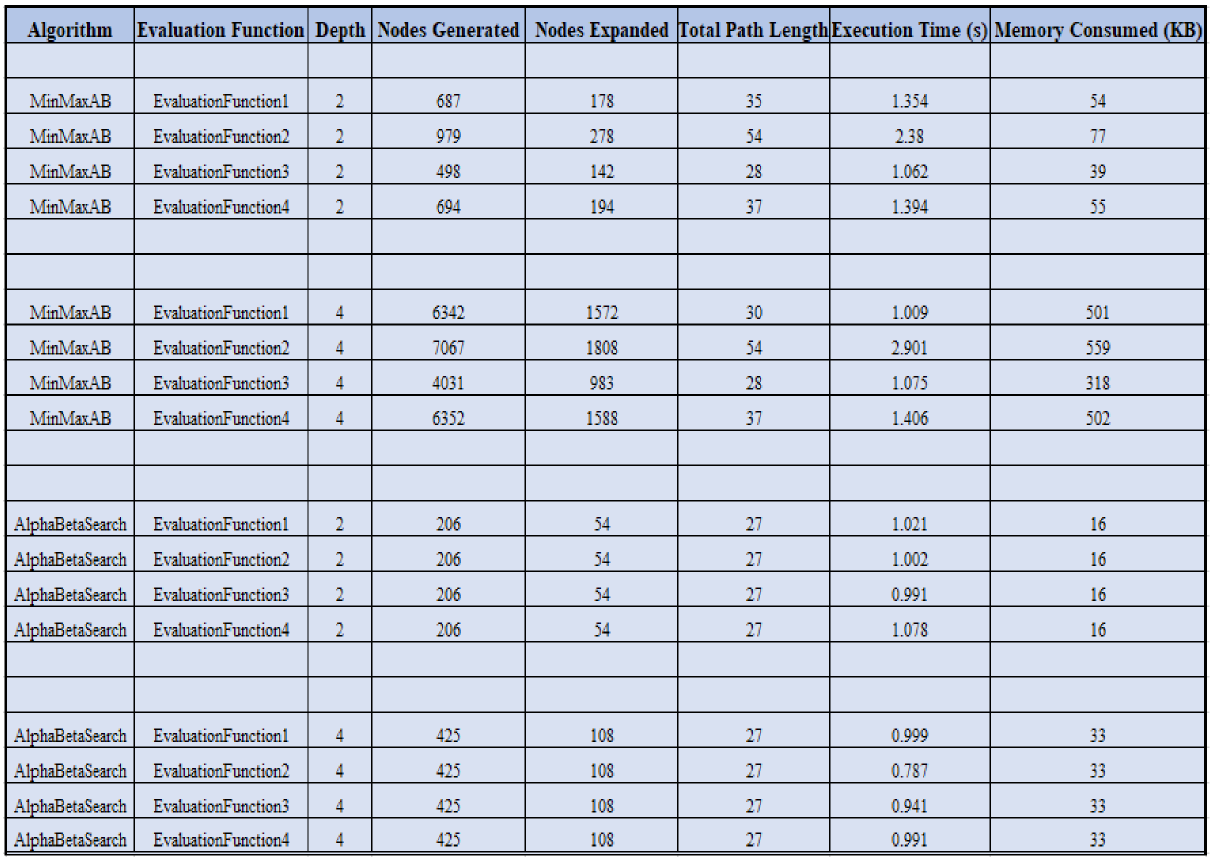


```
Player B ran out of all the stones! All the pits in B are empty!

                PLAYER A

         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of A
        =========================
        | 0 | 0 | 0 | 0 | 0 | 0 |
        =========================
store_A                            store_B
------                             ------
| 50 |                             | 22 |
------                             ------
        =========================
        | 0 | 0 | 0 | 0 | 0 | 0 |
        =========================
         | 0 | 1 | 2 | 3 | 4 | 5 | --------------> Pit Numbers of B

                PLAYER B


Player A won the GAME with 50 stones!

*******************************************************************************************************************
                                                 ***GAME OVER***
*******************************************************************************************************************


The program took '1.013' seconds of Execution time for the completed game!

1 node takes 81 bytes of Memory

The algorithm takes : '34425' bytes = '33' kb of memory!

The program generated '425' nodes for the completed game!

The program expanded '108' nodes for the completed game!

The total length of the Game path is '27' for the completed game!


Process returned 0 (0x0)   execution time : 9.175 s
Press any key to continue.
```

8 ANALYSIS OF THE PROGRAM AND RESULT

We have used Classes and objects to implement the project which is an Object Oriented Approach which is more secure than procedural method. We can use it to base real world. The object oriented approach made it easy for us to overload the operator '=' which made the code more efficient, as we could decide what had to be compared/assigned when we used '=' operator. Since we used this approach, adding functions in between was as easy as just editing the class and not the whole usage. This approach makes it easy to be reused, it is more flexible now. Adding separate classes for each functionality makes it readable and easy to understand. We have added comments to each function, which gives us a brief idea of what the function is doing, without having to go through the code in the function to understand its functionality. We have used evaluation functions in a way to make sure they are admissible and good heuristics, evaluating the moves of each player based on the opponent's move.

Below are the tabulated results of the different runs of the program with different depth, different evaluation function and different algorithm:



Algorithm	Evaluation Function	Depth	Nodes Generated	Nodes Expanded	Total Path Length	Execution Time (s)	Memory Consumed (KB)
MinMaxAB	EvaluationFunction1	2	687	178	35	1.354	54
MinMaxAB	EvaluationFunction2	2	979	278	54	2.38	77
MinMaxAB	EvaluationFunction3	2	498	142	28	1.062	39
MinMaxAB	EvaluationFunction4	2	694	194	37	1.394	55
MinMaxAB	EvaluationFunction1	4	6342	1572	30	1.009	501
MinMaxAB	EvaluationFunction2	4	7067	1808	54	2.901	559
MinMaxAB	EvaluationFunction3	4	4031	983	28	1.075	318
MinMaxAB	EvaluationFunction4	4	6352	1588	37	1.406	502
AlphaBetaSearch	EvaluationFunction1	2	206	54	27	1.021	16
AlphaBetaSearch	EvaluationFunction2	2	206	54	27	1.002	16
AlphaBetaSearch	EvaluationFunction3	2	206	54	27	0.991	16
AlphaBetaSearch	EvaluationFunction4	2	206	54	27	1.078	16
AlphaBetaSearch	EvaluationFunction1	4	425	108	27	0.999	33
AlphaBetaSearch	EvaluationFunction2	4	425	108	27	0.787	33
AlphaBetaSearch	EvaluationFunction3	4	425	108	27	0.941	33
AlphaBetaSearch	EvaluationFunction4	4	425	108	27	0.991	33

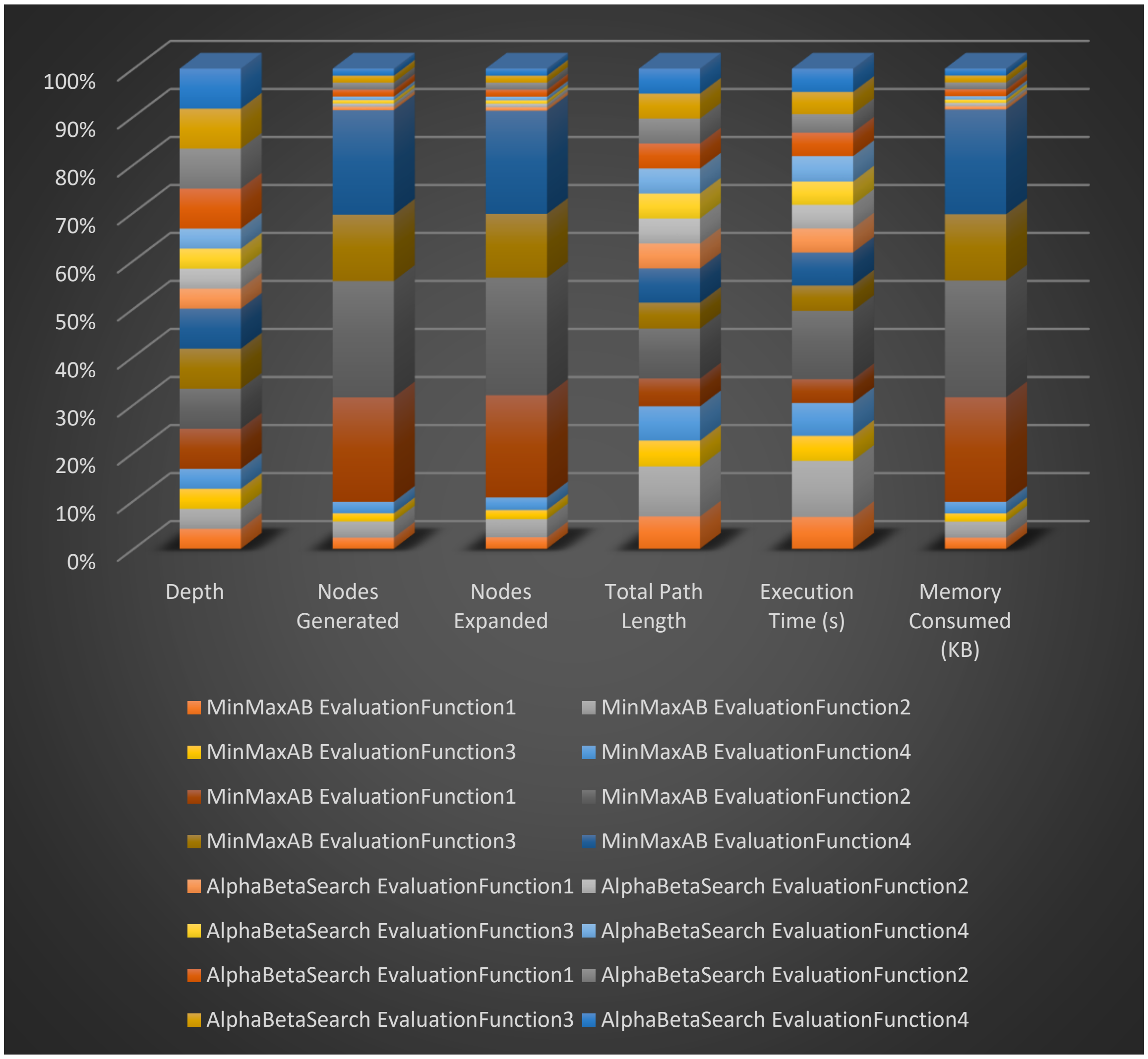
As mentioned earlier, we have 4 evaluation functions in this program and using each of those evaluation functions with different algorithms has given us varied results. We have considered a combination of depth, algorithm type, evaluation function and have landed with the above 16 set of results, 8 for each algorithm.

When we say MinMax-AB, both the players use MinMAxAB algorithm to play the game. Each time, the number of nodes generated, number of nodes expanded, and total length of the path were different. This was because of the different evaluation functions used and the different depths considered in each game. We ran the program 4 times for MinMax AB for depth 2 with 4 different evaluation functions and 4 times for MinMaxAB for depth 4 with 4 different evaluation functions. Choice 1 of our menu initiates the Computer vs Computer game using the MinMaxAB algorithm and the results are shown in above table.

When we say AlphaBetaSearch, both the players use AlphaBetaSearch algorithm to play the game. Each time, the number of nodes generated, the number of nodes expanded, and total length of the path were different. This was because of the different evaluation functions used and the different depths considered in each game. We ran the program 4 times for AlphaBetaSearch for depth 2 with 4 different evaluation functions and 4 times for AlphaBetaSearch for depth 4 with 4 different evaluation functions. Choice 2 of our menu initiates the Computer vs Computer game using the AlphaBetaSearch algorithm and the results are shown in above table

It is difficult to measure the performance of the two algorithms or to decide which algorithm is better. But looking at the collected data we can deduce few very important information:

We see that AlphaBetaSearch has a better performance than the MinMaxAB search algorithm. We can conclude this with an instance of result here: Consider a run for depth 2 and Evaluation Function 3 for both MinMaxAB and AlphaBetaSearch algorithms. We notice that, in this scenario, MinMaxAB generated 498 nodes, expanded 142 nodes and the total game path length was 28. For the same scenario, AlphaBetaSearch generated 206 nodes, expanded 54 nodes and the total game path length was 27. This makes us infer that AlphaBetaSearch was better.



From the above bar graph representation of the results, we can also comment on the performance of the Evaluation functions. We see that Evaluation function 3 yields the best result comparatively as it generated minimal nodes, expands minimal nodes and the game path length is minimal too for MinMaxAB. For AlphaBetaSearch, all the evaluation functions yield the same results, as, each time, the same number of unwanted nodes are pruned making it an efficient algorithm.

9 CONCLUSION

Implementing the MinMaxAB algorithm and AlphaBetaSearch algorithm using Kalah Game as an example to test the implementation was a learning experience. Over the course of completion of the project, I now have better understanding of MinMaxAB and AlphaBetaSearch algorithms. Having understood the concepts of these search algorithms prior to doing this project, this project gave me an opportunity to learn the implementation of the algorithms that we read to real life applications, Kalah Game in this case. Conversion of the understanding of concepts to a working project has been a path to learn the practical applications of these search algorithms. I was able to keep a track of how many nodes were generated for each algorithm and each depth, which helped me to practically analyze as to which algorithm performed better and why. Was able to write code that was more readable, flexible and got a hold of using objects better. Besides, this project was a platform to learn in a better way the rules of Kalah Game and the history of how the game evolved over years and generations. I also got a better understanding on the software development process and the design principles as we followed the approach in this project.

I would like to sincerely thank you for this opportunity that helped me in many ways to get a hold of practical implementation of a given problem.

10 REFERENCES

- Russell, Stuart, and Norvig, Peter. Artificial Intelligence, A Modern Approach. 3rd ed. Upper Saddle River, New Jersey, 2010

- https://en.wikipedia.org/wiki/Kalah#Kalah(6,6)

- Artificial intelligence, by E.Rich, K.Knight, K.B.Nair

- https://en.wikipedia.org/wiki/Mancala

- https://www.rose-hulman.edu/class/csse/archive/other-old/archive/winter99/kalah/KalahRules.html

- https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning

- https://www.hackerearth.com/blog/developers/minimax-algorithm-alpha-beta-pruning/

- https://en.wikipedia.org/wiki/Evaluation_function