A.I.ware

Pascal Baillehache
Pascal@BayashiInJapan.net

July 28, 2017

Contents

Introduction

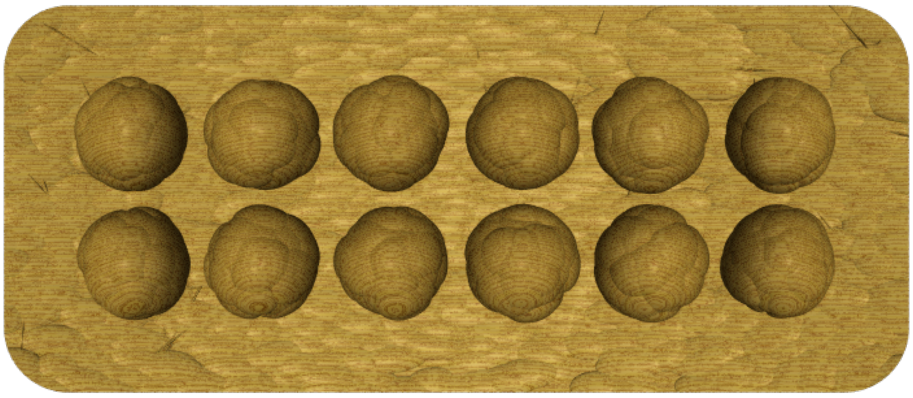
The game of Oware is a game among the Mancala family board games. It has many variants and many names as it has been played all around the world since hundreds of years. A.I.ware is an online version of this game where the human player can play against an artifical intelligence.

The game is availalbe to play here:
http://www.BayashiInJapan.net/A.I.ware/

1 Design

There are two main elements to the design of the game: the board and the stones. They are created with POV-Ray. Other elements of the interface are created with HTML and CSS.

1.1 Board



1.1.1 board.pov

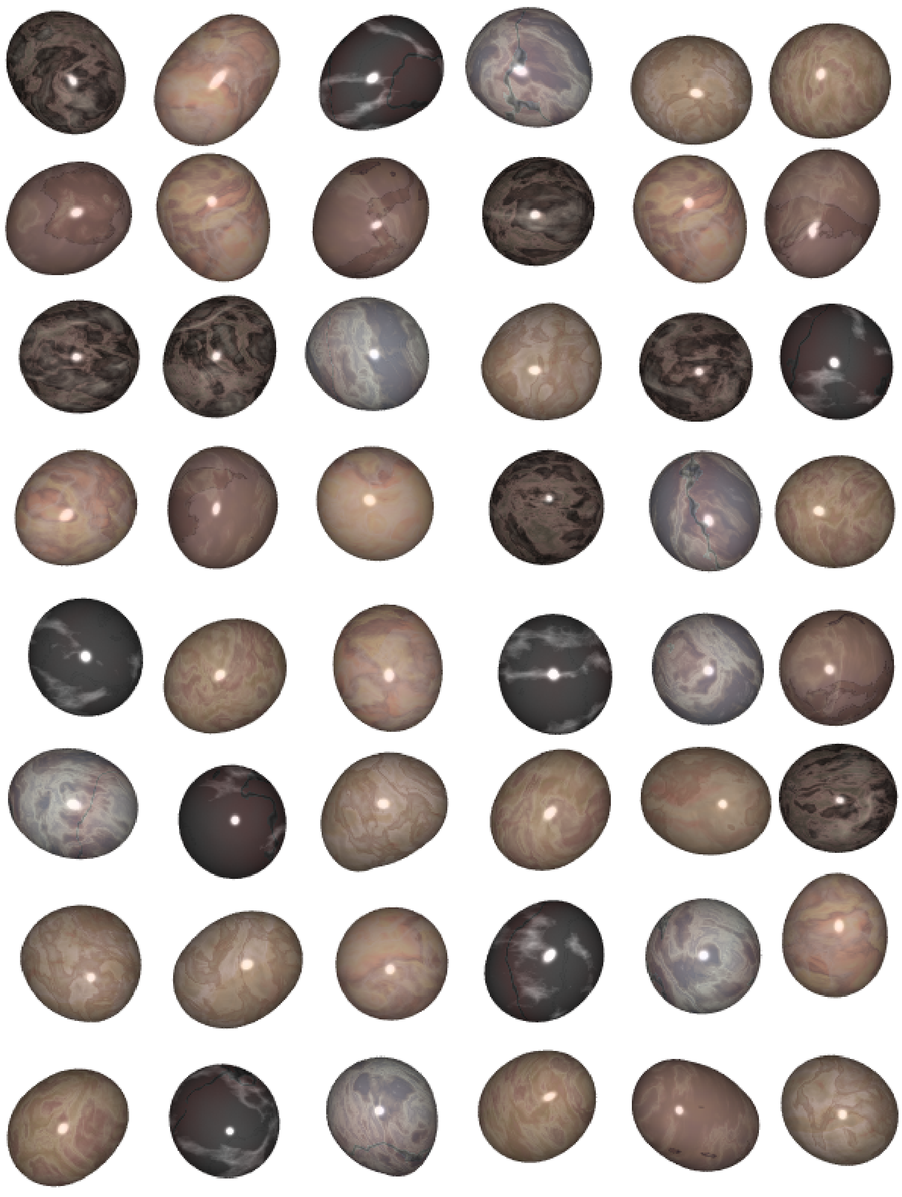
```
#include "colors.inc"
#include "woods.inc"
#include "textures.inc"
// unit is centimeter
```

```
#declare _texNeutral = texture {
  pigment { color rgb <0.75, 0.75, 0.75> }
  finish { ambient 0.1 diffuse 0.6 phong 0.0}
}

#declare _texBoard = texture {
  Yellow_Pine
  rotate y * 90.0
}

#declare _texVarnish = texture {
  pigment { color rgbft <1.0,1.0,1.0,1.0> }
  finish {
    phong 0.1
    phong_size 50.0
    reflection 0.025
  }
}

#declare _tex = texture {
  pigment { color White }
}

#declare RndSeed = seed(10); //30
#declare RndSeedScratch = seed(3);
#declare _posCamera = <0.0,30.0,0.0>;
#declare _lookAt = <0.0,0.0,0.0>;

#declare _nbHolePlayer = 6;
#declare _holeSize = 4.0;
#declare _boardWidth = _holeSize * 3.0;
#declare _boardLength = _holeSize * (_nbHolePlayer + 1);
#declare _boardHeight = _holeSize * 1.5;

camera {
  orthographic
  location    _posCamera
  sky z
  right x * (_boardLength + .1)
  up z * (_boardWidth + .1)
  look_at     _lookAt
}

light_source {
  _posCamera
  color rgb 1.0
  area_light <-0.5 * _boardLength, 0, -0.5 * _boardWidth>,
    <0.5 * _boardLength, 0, 0.5 * _boardWidth>, 6, 5
  adaptive 1
  jitter
}

background { color rgbft <1.0, 1.0, 1.0, 1.0, 1.0> }

global_settings { ambient_light 0 radiosity {brightness 0.5}}

#declare _boardBase = union {
  box {
    <-3.0 * _holeSize, 0.0, -1.5 * _holeSize>
    <3.0 * _holeSize, _boardHeight, 1.5 * _holeSize>
  }
  box {
    <-3.5 * _holeSize, 0.0, -1.0 * _holeSize>
    <3.5 * _holeSize, _boardHeight, 1.0 * _holeSize>
  }
  cylinder {
    <-3.0 * _holeSize, 0.0, -1.0 * _holeSize>
    <-3.0 * _holeSize, _boardHeight, -1.0 * _holeSize>
    0.5 * _holeSize
  }
  cylinder {
    <3.0 * _holeSize, 0.0, -1.0 * _holeSize>
    <3.0 * _holeSize, _boardHeight, -1.0 * _holeSize>
    0.5 * _holeSize
  }
  cylinder {
    <-3.0 * _holeSize, 0.0, 1.0 * _holeSize>
    <-3.0 * _holeSize, _boardHeight, 1.0 * _holeSize>
    0.5 * _holeSize
  }
  cylinder {
    <3.0 * _holeSize, 0.0, 1.0 * _holeSize>
    <3.0 * _holeSize, _boardHeight, 1.0 * _holeSize>
    0.5 * _holeSize
  }
}

#declare _boardCarveSurface = union {
  #local _nbCarve = 1000;
  #local _iCarve = 0;
  #local _depthCarve = 0.2;
  #local _sizeCarve = 0.5;
  #while (_iCarve < _nbCarve)
    sphere {
      0.0, 1.0
      scale <_sizeCarve * (1.5 + rand(RndSeed)),
        _depthCarve * (rand(RndSeed) * 0.5 + 0.5),
        _sizeCarve>
      rotate y * (rand(RndSeed) - 0.5) * 20.0
      translate <(rand(RndSeed) - 0.5) * _boardLength,
        _boardHeight + _depthCarve * 0.25 * rand(RndSeed),
        (rand(RndSeed) - 0.5) * _boardWidth>
    }
    #declare _iCarve = _iCarve + 1;
  #end
}

#declare _boardScratchSurface = union {
  #local _nbCarve = 40; //20;
  #local _iCarve = 0;
  #local _depthCarve = 0.15;
  #local _sizeCarve = 2.0;
  #while (_iCarve < _nbCarve)
    sphere {
      0.0, 1.0
      scale <_sizeCarve * (0.5 + rand(RndSeedScratch)),
        _depthCarve * (rand(RndSeedScratch) * 0.5 + 0.5),
        0.025 + 0.025 * rand(RndSeedScratch)>
      rotate y * (rand(RndSeedScratch) - 0.5) * 180.0
      translate <(rand(RndSeedScratch) - 0.5) * _boardLength,
        _boardHeight// + _depthCarve * 0.3 * rand(RndSeedScratch),
        (rand(RndSeedScratch) - 0.5) * _boardWidth>
    }
    #declare _iCarve = _iCarve + 1;
  #end
}

#declare _boardCarveHoles = union {
  #local _relSizeHole = 0.35;
  #local _iPlayer = 0;
  #while (_iPlayer < 2)
    #local _iHole = 0;
    #while (_iHole < _nbHolePlayer)
      sphere {
        0.0, _holeSize * _relSizeHole
        translate <
          _holeSize * (_iHole + 0.5 - 0.5 * _nbHolePlayer),
          _boardHeight, _holeSize * (-0.5 + _iPlayer)>
      }

      #local _nbCarve = 200;
      #local _iCarve = 0;
      #local _sizeCarve = _holeSize * _relSizeHole * 0.6;
      #while (_iCarve < _nbCarve)
        sphere {
          0.0, 1.0
          #local _l = sqrt(_holeSize * _relSizeHole *
            _holeSize * _relSizeHole +
            _sizeCarve * _sizeCarve);
          #local _lp = _holeSize * _relSizeHole - _l;
          scale <_sizeCarve,
            _lp + (_sizeCarve - _lp) * rand(RndSeed) * 0.75,
            _sizeCarve>
          translate y * -0.8 * _l
          rotate x * (rand(RndSeed) - 0.5) * 200.0
          rotate y * rand(RndSeed) * 360.0
          translate <
            _holeSize * (_iHole + 0.5 - 0.5 * _nbHolePlayer),
            _boardHeight, _holeSize * (-0.5 + _iPlayer)>
        }
        #declare _iCarve = _iCarve + 1;
      #end

      #declare _iHole = _iHole + 1;
    #end
    #declare _iPlayer = _iPlayer + 1;
  #end
}

#declare _boardCarve = union {
  object { _boardCarveSurface }
  object { _boardScratchSurface }
  object { _boardCarveHoles }
}

#declare _board = difference {
  object { _boardBase }
  object { _boardCarve }
  //texture { _texNeutral }
  texture { _texBoard }
  texture { _texVarnish }
}

object {
  _board
}
```

1.1.2 Makefile

```
pov: board.pov
        povray -W700 -H300 -D -P -Q9 +A +UA -Iboard.pov
```

1.2 Stones



1.2.1 stones.pov

```
#include "colors.inc"
#include "stones.inc"
#include "textures.inc"
// unit is centimeter

#declare _texNeutral = texture {
  pigment { color rgb <0.75, 0.75, 0.75> }
  finish { ambient 0.1 diffuse 0.6 phong 0.0}
}

#declare RndSeed = seed(clock);
#declare _posCamera = <0.0,10.0,0.0>;
#declare _lookAt = <0.0,0.0,0.0>;

camera {
  orthographic
  location _posCamera
  sky z
  right x
  up z
  look_at _lookAt
}

light_source {
  _posCamera
  color rgb 1.0
  area_light <-0.5, 0, -0.5>, <0.5, 0, 0.5>, 3, 3
  adaptive 1
  jitter
}

background { color rgbft <1.0, 1.0, 1.0, 1.0, 1.0> }

global_settings { ambient_light 0 radiosity {brightness 0.5}}

#declare _stone = blob {
  threshold 1.0
  #local _v = 0.35;
  #local _r = 0.5;
  #local _d = 0.4 * _r;
  #local iSphere = 0;
  #while (iSphere < 4)
    #local _s = 1.0/((1.0-(_d/_r)*(_d/_r))*(1.0-(_d/_r)*(_d/_r)));
    sphere {
      <_v * (rand(RndSeed) - 0.5),
      _v * (rand(RndSeed) - 0.5),
      _v * (rand(RndSeed) - 0.5)>,
      _r, _s
    }
    #declare iSphere = iSphere + 1;
  #end
  texture {
    #local _tex = floor(rand(RndSeed) * 10.0);
    #switch (_tex)
    #case (0) T_Stone1 #break
    #case (1) T_Stone2 #break
    #case (2) T_Stone3 #break
    #case (3) T_Stone4 #break
    #case (4) T_Stone5 #break
    #case (5) T_Stone6 #break
    #case (6) T_Stone7 #break
    #case (7) T_Stone8 #break
    #case (8) T_Stone9 #break
    #case (9) T_Stone10 #break
    #else
      _texNeutral
    #end
    translate 10.0 * rand(RndSeed)
  }
}

object {
  _stone
}
```

1.2.2 stones.ini

```
Input_File_Name=stone.pov

Initial_Clock = 0.0
Final_Clock = 47.0

Initial_Frame = 0
Final_Frame = 47
```

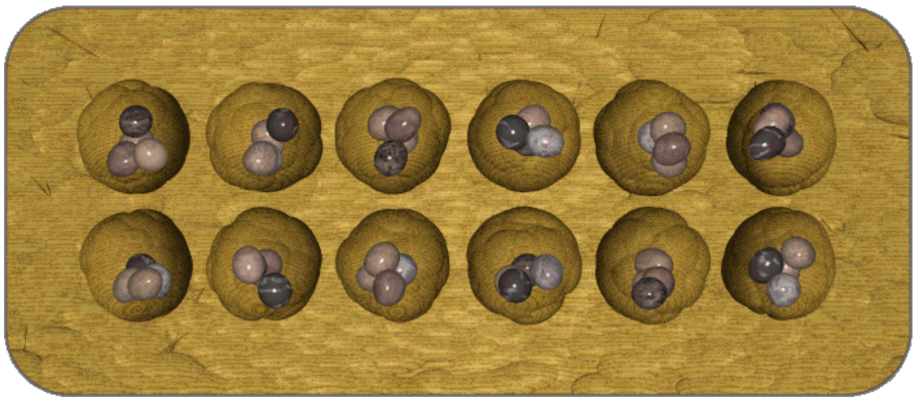
1.2.3 Makefile

```
pov: stone.pov
        povray -W100 -H100 -D -P -Q9 +A +UA +O./ stone.ini
```

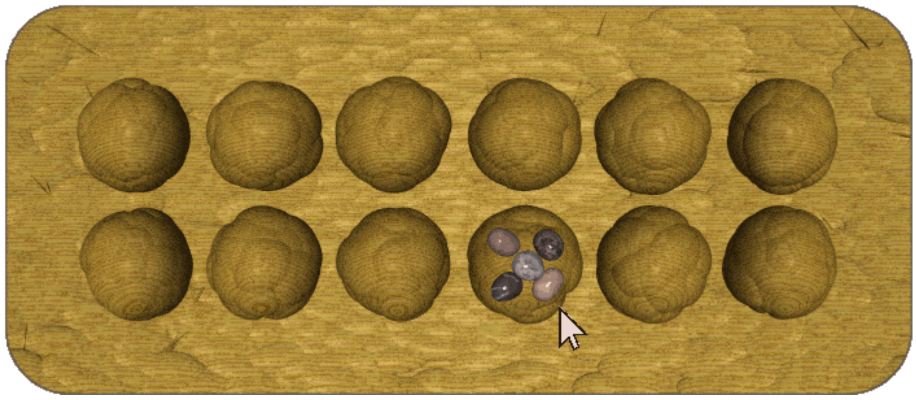
2 Rules

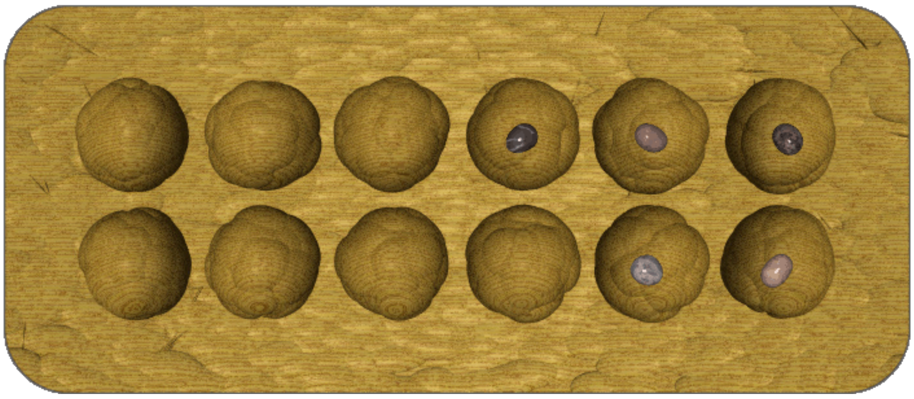
The game is played between two players (the Human and the A.I.) with a board and 48 stones. The board has 12 holes divided into 2 lines. The 6 holes on the top line are the A.I. territory. The 6 holes on the bottom line are the Human territory.

To start a new game, select the level of the A.I. (beginner, easy, intermediate, strong) and click "Start a new game". The 48 stones are distributed into the 12 holes, 4 stones per hole.

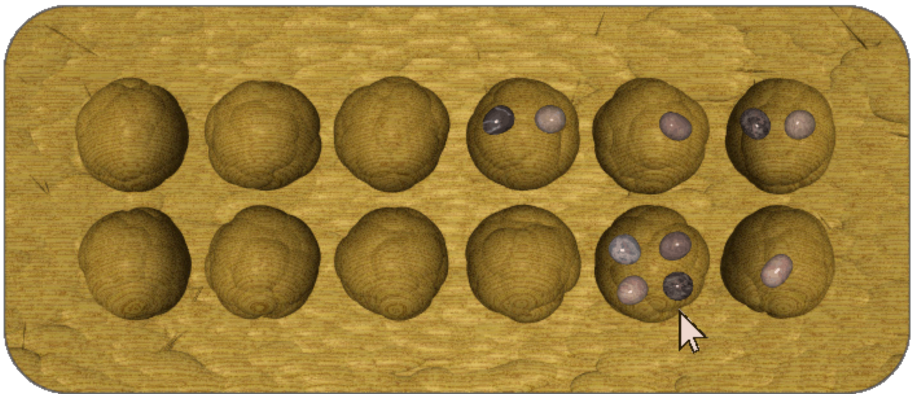


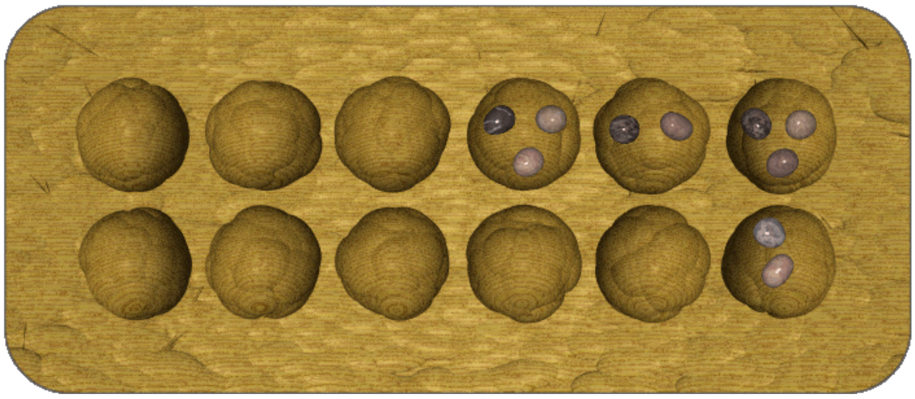
Players play one at a time. The current player turn is displayed below the board. The current player choose one hole containing stones in its territory. Stones in the choosen hole are moved in the following holes, one stone per hole, in a counter clockwise order. To select a hole, click on it and it will display below the board the number of stones inside. To play this hole, click on it a second time.

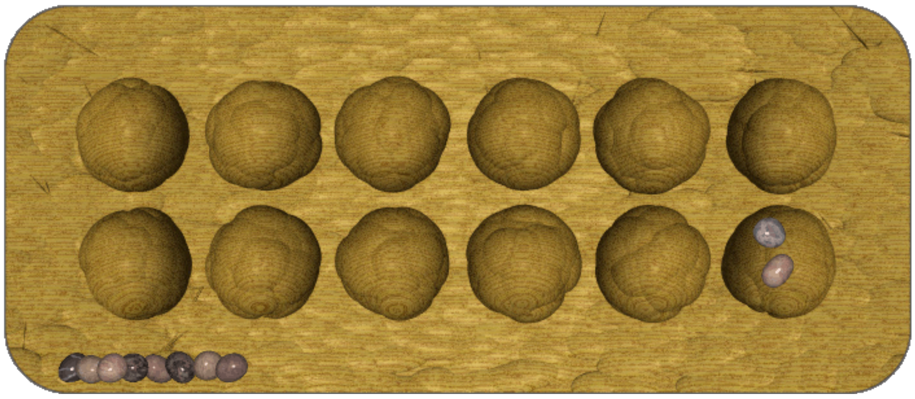




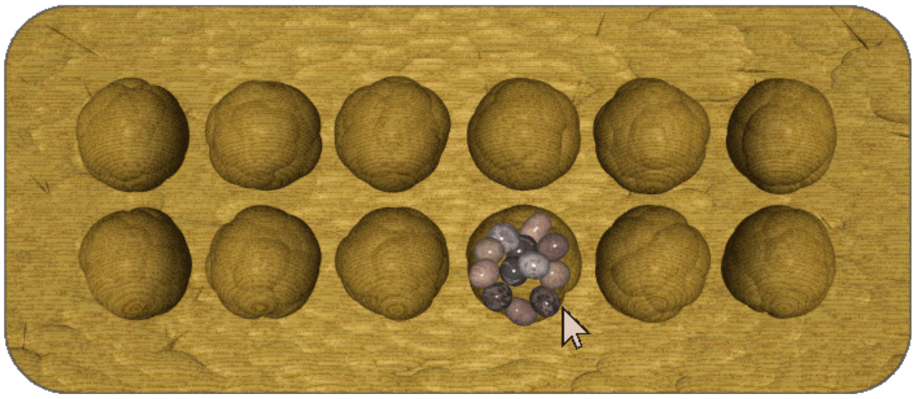
When the last moved stone arrives in a hole where there is already 1 or 2 stones, the current player captures all the stones in this hole (2 or 3 with the moved stone). Then, if the previous hole contains 2 or 3 stones, they are captured too and so on until we reach a hole with a number of stones different from 2 or 3 or go out of the current player's territory.

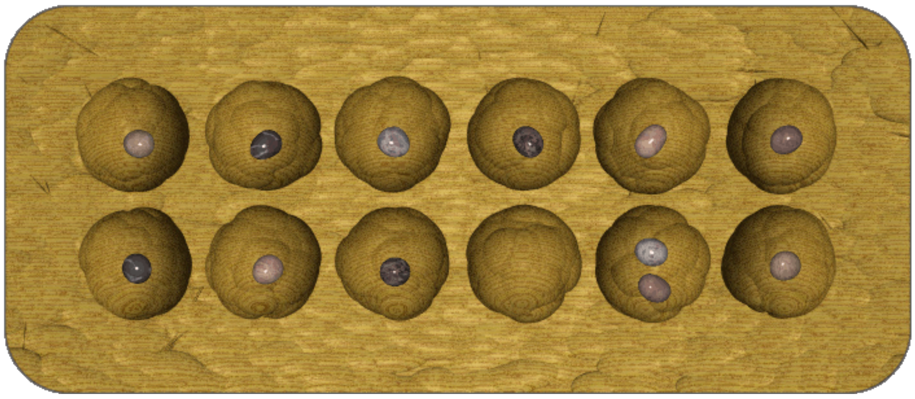






If the current player chooses a hole with more than 11 stones, the distribution of stones will loop back to the starting hole. In this case the starting hole is jumped over and the distribution continue with its neighbour.





The game ends when one player has captured at least half of the stones (25 stones or more). The winner is the player who has captured the more stones.

Special case 1. If the current player has no more stone in its territory, the opponent automatically captures all the stones in its own territory and the game ends.

Special case 2. If the current players captures at once all the stones in the opponent's territory, he immediately looses the game.

3 A.I.

The artificial intelligence for the game is a binary executable on the server which run with the exec PHP command. The algorithm of the A.I. is a Minimax on the tree of possible successive moves. The depth of the tree allows to control the strength of the A.I.: at 0 it chooses randomly a move, from 1 to 3 it search the best move in a tree of correpsonding depth. The evaluation function for the Minimax is simply (score of current player - score of opponent). The algorithm is implemented in C.

3.1 aiware.c

```
/* ============ aiware.c ========== */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <math.h>

// --------- Global constants and inline ----------------

#define NBPLAYER 2
#define NBHOLEPLAYER 6
#define NBHOLE (NBHOLEPLAYER * NBPLAYER)
#define NBINITSTONEPERHOLE 4
#define NBSTONE (NBHOLE * NBINITSTONEPERHOLE)
#define rnd() (double)(rand())/(float)(RAND_MAX)

// --------- Data structures ----------------

typedef struct Board Board;
struct Board {
  int _level;
  int _nbStone[NBHOLE];
  Board *_next[NBHOLEPLAYER];
  int _score[NBPLAYER];
  char _end;
};

// --------- Function declaration ----------------

void BoardInit(Board *board, char **arg);
int BoardGetRandMove(Board *board, int iPlayer);
int BoardGetBestMove(Board *board, int iPlayer);
void BoardFree(Board *board);
void BoardExpand(Board *board, int iPlayer, int depth);
void BoardPlayMove(Board *board, int iHole, int iPlayer);
int BoardSearchBestMove(Board *board, int refPlayer,
  int iPlayer, float *v);
float BoardValue(Board *board, int iPlayer);
void BoardDisplay(Board *board);

// --------- Function implementation ----------------

void BoardInit(Board *board, char **arg) {
  // Decode the arguments
  // Level
  board->_level = atoi(arg[1]);
  // Sanitize the level value
  if (board->_level < 0) board->_level = 0;
  if (board->_level > 3) board->_level = 3;
  // Score
  for (int iPlayer = 0; iPlayer < NBPLAYER; ++iPlayer) {
    board->_score[iPlayer] = atoi(arg[2 + iPlayer]);
  }
  // Nb stone in holes
  for (int iHole = 0; iHole < NBHOLE; ++iHole) {
    board->_nbStone[iHole] = atoi(arg[4 + iHole]);
  }
  // Initialize the pointers toward next boards
  for (int iHole = 0; iHole < NBHOLEPLAYER; ++iHole) {
    board->_next[iHole] = NULL;
  }
  // Initialize the flag for end of game
  board->_end = 0;
}

int BoardGetRandMove(Board *board, int iPlayer) {
  // Select randomly a hole and loop until we find one containing
  // at least one stone
  // To ensure no infinite loop put a limit to the number of loop
  int iHole = -1;
  int iLoop = 1000;
  while (iHole == -1 && iLoop > 0) {
    int h = (int)floor(rnd() * ((float)NBHOLEPLAYER - 0.0001));
    if (board->_nbStone[iPlayer * NBHOLEPLAYER + h] > 0) {
      iHole = h;
    }
    --iLoop;
  }
  return iHole;
}

void BoardPlayMove(Board *board, int iHole, int iPlayer) {
  int nbStone = board->_nbStone[iHole];
  // Remove stones from starting hole
  board->_nbStone[iHole] = 0;
  // Distribute stones
  int jHole = iHole;
  while (nbStone > 0) {
    ++jHole;
    if (jHole == NBHOLE) jHole = 0;
    // Jump over starting hole
    if (jHole == iHole) ++jHole;
    if (jHole == NBHOLE) jHole = 0;
    ++(board->_nbStone[jHole]);
    --nbStone;
  }
  // Check for captured stones
  int iOpp = 1 - iPlayer;
  char flagCaptured = 0;
  while (jHole >= iOpp * NBHOLEPLAYER &&
    jHole < (iOpp + 1) * NBHOLEPLAYER &&
    (board->_nbStone[jHole] == 2 ||
    board->_nbStone[jHole] == 3)) {
    board->_score[iPlayer] += board->_nbStone[jHole];
    board->_nbStone[jHole] = 0;
    flagCaptured = 1;
    --jHole;
  }
  // Check for end conditions
  // First, check that the opponent is not starving
  int nbStoneOpp = 0;
  for (int iHole = 0; iHole < NBHOLEPLAYER; ++iHole) {
    nbStoneOpp += board->_nbStone[iOpp * NBHOLEPLAYER + iHole];
  }
  // If the opponent is starving
  if (nbStoneOpp == 0) {
    if (flagCaptured == 1) {
      // If there has been captured stones, it means the current
      // player has starved the opponent. The current player looses.
      board->_end = 1;
      board->_score[iPlayer] = -100.0;
      board->_score[iOpp] = 100.0;
    } else {
      // If there was no captured stones, it means the opponent
      // starved itself. The current player catches all his own stones.
      board->_end = 1;
      for (int iHole = 0; iHole < NBHOLEPLAYER; ++iHole) {
        board->_score[iPlayer] +=
          board->_nbStone[iPlayer * NBHOLEPLAYER + iHole];
      }
    }
  } else if (board->_score[0] * 2 > NBSTONE ||
    board->_score[1] * 2 > NBSTONE) {
    // If one of the player has captured more than half the stones
    // the game ends.
    board->_end = 1;
  }
}

void BoardDisplay(Board *board) {
  // Display the board (for debug)
  for (int iHole = 0; iHole < NBHOLE; ++iHole) {
    printf("%d ", board->_nbStone[iHole]);
  }
  printf("\n");
}

void BoardExpand(Board *board, int iPlayer, int depth) {
  // Expand the story from this board up to a depth equal
  // to level given in argument or end o fthe game
  if (depth != 0 && board->_end == 0) {
    // For each hole containing stone
    for (int iHole = 0; iHole < NBHOLEPLAYER; ++iHole) {
      if (board->_nbStone[iPlayer * NBHOLEPLAYER + iHole] > 0) {
        // Create a copy of the current board
        Board *nBoard = (Board*)malloc(sizeof(Board));
        if (nBoard != NULL) {
          // Affect the copy of the board to the pointer corresponding
          // to this hole
          board->_next[iHole] = nBoard;
          memcpy(nBoard, board, sizeof(Board));
          for (int iHole = 0; iHole < NBHOLEPLAYER; ++iHole) {
            nBoard->_next[iHole] = NULL;
          }
          // Play the move
          BoardPlayMove(nBoard, iHole + iPlayer * NBHOLEPLAYER, iPlayer);
          // Get the next player
          int nPlayer = iPlayer + 1;
          if (nPlayer == NBPLAYER)
            nPlayer = 0;
          // Expand from the new board for next player
          BoardExpand(nBoard, nPlayer, depth - 1);
        }
      }
    }
  }
}

float BoardValue(Board *board, int iPlayer) {
  // Get the value of the board for player iPlayer
  int iOpp = 1 - iPlayer;
  // Simple evaluation based on score
  float ret = board->_score[iPlayer] - board->_score[iOpp];
  // Add noise to discriminate randomly between equal values move
  ret += rnd() * 0.001;
  return ret;
}

int BoardSearchBestMove(Board *board, int refPlayer,
  int iPlayer, float *v) {
  // Search for best move using minimax
  int bestMove = -1;
  float bestVal = 0.0;
  int iOpp = 1 - iPlayer;
  char flagLeaf = 1;
  // Search among childs
  for (int iHole = 0; iHole < NBHOLEPLAYER; ++iHole) {
    if (board->_next[iHole] != NULL) {
      flagLeaf = 0;
      float val;
      // Get best move for this child
      int m = BoardSearchBestMove(board->_next[iHole],
        refPlayer, iOpp, &val);
      // To avoid warning at compilation
      m = m;
      if (bestMove == -1) {
        bestMove = iHole;
        bestVal = val;
      } else {
        // Select best move if its turn of refPlayer, worst move else
        if ((refPlayer == iPlayer && val > bestVal) ||
          (refPlayer != iPlayer && val < bestVal)) {
          bestMove = iHole;
          bestVal = val;
        }
      }
    }
  }
  // If this board is a leaf
  if (flagLeaf == 1) {
    // The value of the board is its own value
    *v = BoardValue(board, refPlayer);
  } else {
    // The value of the board is the best value in its child
    *v = bestVal;
  }
  return bestMove;
}

int BoardGetBestMove(Board *board, int iPlayer) {
  int iHole = -1;
  // Expand the story
  BoardExpand(board, iPlayer, board->_level);
  // Search the best move
  float bestVal;
  iHole = BoardSearchBestMove(board, iPlayer, iPlayer, &bestVal);
  return iHole;
}

void BoardFree(Board *board) {
  // Free memory used by board
  for (int iHole = 0; iHole < NBHOLEPLAYER; ++iHole) {
    if (board->_next[iHole] != NULL) {
      BoardFree(board->_next[iHole]);
      free(board->_next[iHole]);
      board->_next[iHole] = NULL;
    }
  }
}

// ---------- Main ----------------

int main(int argc, char **argv) {
  // Intialise the random generator
  srandom(time(NULL));

  // Check the number of arguments
  if (argc != 16) {
    printf("-1");
  }

  // Load the arguments
  Board theBoard;
  BoardInit(&theBoard, argv);

  // Get the move
  int iHole = -1;
  if (theBoard._level == 0) {
    iHole = BoardGetRandMove(&theBoard, 0);
  } else {
    iHole = BoardGetBestMove(&theBoard, 0);
  }

  // Free the memory
  BoardFree(&theBoard);

  // Output the move
  printf("%d", iHole);
  fflush(stdout);

  return 0;
}
```

3.2 Makefile

```
OPTIONS_DEBUG= -Wall -ggdb -g3 -lSDL2 -lm
OPTIONS_RELEASE=-O3 -Wall -s -lSDL2 -lm
OPTIONS=$(OPTIONS_RELEASE)

all : AIware

AIware : aiware.o
        gcc aiware.o $(OPTIONS) -o AIware

aiware.o : aiware.c
        gcc -g $(OPTIONS) -c aiware.c

clean :
        rm -rf *.o AIware

valgrind :
        valgrind -v --track-origins=yes --leak-check=full
        --gen-suppressions=yes --show-leak-kinds=all
        AIware 0 0 0 1 6 5 5 5 4 4 4 4 0 5 5
```

```
test :
       ./AIware 1 0 0 1 1 1 1 3 1 1 1 0 0 0 0
```

4 User interface

The User interface is a web page. It animates the design with JavaScript, and communicates with the A.I. through Http Request. It also store information about the win/loss of the game into a MySQL database (also through Http Request) to display statistics (see next section).

4.1 index.php

```
<?php
  // ----------------- index.php --------------------->
  // Start the PHP session
  session_start();

  // Ensure no message will interfere with output
  ini_set('display_errors', 'Off');
  error_reporting(0);

  // Turn on display of errors and warning for debug
  /*ini_set('display_errors', 'On');
  error_reporting(E_ALL ^ E_WARNING);
  error_reporting(E_ALL | E_STRICT);*/

  // Include the PHP files
  include("./db.php");

  // Create the statistics database if requested
  if ($_GET["installDB"] == "1") {
    CreateDB();
  }
?>
<!DOCTYPE html>
<html>
  <head>

    <!-- Meta -->
    <meta content="text/html; charset=UTF-8;">
    <meta name="viewport"
      content="width=device-width, initial-scale=1, maximum-scale=1">
    <meta name="description" content="Mancala online game A.I.ware" />
    <meta name="keywords" content="mancala, awale, oware, game, artificial,
    intelligence, board, A.I.ware" />

    <!-- Icon -->
    <link rel="icon" type="image/x-icon"
      href="./Img/aiware.ico" />

    <!-- Include the CSS files -->
    <link href = "./animate.css"
      rel = "stylesheet" type = "text/css">
    <link href = "./aiware.css"
      rel = "stylesheet" type = "text/css">
    <!-- Include the JS files -->
    <script charset = "UTF-8" src = "./jquery.min.js"></script>
    <script charset = "UTF-8" src = "./aiware.js"></script>

    <title>A.I.ware</title>
  </head>
  <body onload = 'BodyOnLoad();'>
    <!-- Main div -->
    <div id = "divMain">

      <!-- Title div -->
      <div id = "divTitle">
        A.I.ware
      </div>

      <!-- Main div -->
      <div id = "divBoard">

      </div>

      <!-- Info div -->
      <div id = "divInfo">
        <div id = "divScore"></div>
        <div id = "divTurn"></div><br>
        <div id = "divInfoHole"></div>
      </div>

      <!-- Cmd div -->
      <div id = "divCmd">
        <select id = "selLevel" onChange = "theAIware.InitNewGame();">
          <option value = "0">Beginner level</option>
          <option value = "1">Easy level</option>
          <option value = "2">Intermediate level</option>
          <option value = "3">Strong level</option>
        </select>
        <input type = "button" value = "Start a new game"
          onclick = "theAIware.InitNewGame();">
        <input type = "button" value = "Show rules"
          onclick = "ShowRules();">
      </div>

      <!-- footer div -->
      <div id = "divFooter">
        Copyright <a href="mailto:Pascal@BayashiInJapan.net">
            P. Baillehache
        </a>, 2017,
        <a href="showStat.php">Statistics</a>,
        <a href="doc.pdf">Documentation</a>.
      </div>

    </div>

    <!-- Rules div -->
    <div id = "divRules">
      <div id = "divRulesContent">
        <img src = "./Img/close.gif" id = "imgRulesClose"
          onclick = "HideRules();">
        <div id = "divRulesTitle">- Rules -</div>
        <div class = "divOneRule">
          The game of Oware is a game among the Mancala family board games. It has
          many variants and many names as it has been played all around the world
          since hundreds of years. A.I.ware is an online version of this game where
          the human player can play against an artifical intelligence. The
          rules and how to play are explained below.
        </div>
        <div class = "divOneRule">
          The game is played between two players (the Human and the A.I.) with a
          board and 48 stones. The board has 12 holes divided into 2 lines.
          The 6 holes on the top line are the A.I. territory. The 6 holes on
          the bottom line are the Human territory.
        </div>
        <img src = "./Img/rules01-1.gif" class = "imgRule"
          style = "height: 100px;">
        <img src = "./Img/rules01-2.gif" class = "imgRule"
          style = "height: 150px;">
        <div class = "divOneRule">
          To start a new game, select the level of the A.I. (beginner, easy,
          intermediate, strong) and click "Start a new game". The 48 stones are
          distributed into the 12 holes, 4 stones per hole.
        </div>
        <img src = "./Img/rules02-1.gif" class = "imgRule">
        <div class = "divOneRule">
          Players play one at a time. The current player turn is displayed below
          the board. The current player choose one hole containing stones
          in its territory. Stones in the choosen hole are moved in the following
          holes, one stone per hole, in a counter clockwise order. To select a
          hole, click on it and it will display below the board the number of stones
          inside. To play this hole, click on it a second time.
        </div>
        <img src = "./Img/rules03-1.gif" class = "imgRule"><br>
        <img src = "./Img/rules03-2.gif" class = "imgRule">
        <div class = "divOneRule">
          When the last moved stone arrives in a hole where there is already 1
          or 2 stones, the current player captures all the stones in this
          hole (2 or 3 with the moved stone). Then, if the previous hole
          contains 2 or 3 stones, they are captured too and so on until we reach
          a hole with a number of stones different from 2 or 3 or go out of
          the current player's territory.
        </div>
        <img src = "./Img/rules04-1.gif" class = "imgRule"><br>
        <img src = "./Img/rules04-2.gif" class = "imgRule"><br>
        <img src = "./Img/rules04-3.gif" class = "imgRule">
        <div class = "divOneRule">
          If the current player chooses a hole with more than 11 stones, the
          distibution of stones will loop back to the starting hole. In this
          case the starting hole is jumped over and the distribution
          continue with its neighbour.
        </div>
        <img src = "./Img/rules05-1.gif" class = "imgRule"><br>
        <img src = "./Img/rules05-2.gif" class = "imgRule">
        <div class = "divOneRule">
          The game ends when one player has captured at least half of the
          stones (25 stones or more). The winner is the player who has captured
          the more stones.
        </div>
        <div class = "divOneRule">
          Special case 1. If the current player has no more stone in its
          territory, the opponent automatically captures all the stones in its
          own territory and the game ends.
        </div>
        <div class = "divOneRule">
          Special case 2. If the current players captures at once all the
          stones in the opponent's territory, he immediately looses the game.
```

```
        </div>
      </div>
    </div>

  </body>

</html>
```

4.2 db.php

```
<?php
  /* ============ db.php =========== */

  // Function to manage exception
  function ManageException($msg) {
    try {
      // Send an email to the developper with the exception message
      $head = "From:you@yourwebsite.net\r\n";
      $head .= "Content-type: text/plain; charset=UTF-8\r\n";
      $email = "you@yourwebsite.net";
      $subject = "ManageException on AIware";
      mail($email, $subject, $msg, $head);
    } catch (Exception $e) {
      echo "ManageException : " . $e->getMessage() . "\n";
    }
  }

  // Function to connect to the database
  function ConnectDatabase() {
    try {
      // Connection information
      $servername = "servername";
      $username = "username";
      $password = "password";
      $dbname = "dbname";

      // Create connection
      $conn = new mysqli(
        $servername,
        $username,
        $password,
        $dbname);
      // Set the charset
      $conn->set_charset("utf8");
      // Return the connection object
      return $conn;
    } catch (Exception $e) {
      ManageException("ConnectDatabase : " . $e->getMessage() . "\n");
      return null;
    }
  }

  // Function to close the connection to the database
  function CloseDatabase($conn) {
    try {
      $conn->close();
    } catch (Exception $e) {
      ManageException("CloseDatabase : " . $e->getMessage() . "\n");
    }
  }
```

```
// Function to create the database tables
function CreateDB() {
  try {
    // Open a connection to the database;
    $conn = ConnectDatabase();
    if ($conn->connect_error) {
      throw new Exception("Error connecting to DB. " .
        $conn->connect_error);
      return;
    }
    // Create the table
    $sql = "CREATE TABLE IF NOT EXISTS AIwareStat (
      Ref INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
      DateGame DATETIME,
      Country CHAR(2),
      Level INT,
      Result INT
      ) COLLATE utf8_bin";
    if ($conn->query($sql) === true) {
      echo "Table AIwareStat created successfully<br>";
    } else {
      throw new Exception("Error creating table. " . $conn->error);
    }
    // Close the connection to the database
    CloseDatabase($conn);
  } catch (Exception $e) {
    ManageException("CreateDB : " . $e->getMessage() . "\n");
  }
}

// Function to get the country code ('FR') from the IP
function GetCountryFromIP() {
  try {
    $client  = @$_SERVER['HTTP_CLIENT_IP'];
    $forward = @$_SERVER['HTTP_X_FORWARDED_FOR'];
    $remote  = @$_SERVER['REMOTE_ADDR'];
    $country  = "??";
    if(filter_var($client, FILTER_VALIDATE_IP)){
      $ip = $client;
    }elseif(filter_var($forward, FILTER_VALIDATE_IP)){
      $ip = $forward;
    }else{
      $ip = $remote;
    }
    $ip_data = @json_decode(
      file_get_contents("http://www.geoplugin.net/json.gp?ip=".$ip));
    if($ip_data && $ip_data->geoplugin_countryName != null){
      $country = $ip_data->geoplugin_countryCode;
    }
    return $country;
  } catch (Exception $e) {
    ManageException("GetCountryFromIP : " . $e->getMessage() . "\n");
  }
}

// Function to save one game result
function SaveOneResult($dateGame, $country, $level, $result) {
  try {
    $conn = ConnectDatabase();
    if ($conn->connect_error) {
      throw new Exception("Error connecting to DB. " .
        $conn->connect_error);
    }
    $sql = "INSERT INTO AIwareStat ";
    $sql .= "(Ref, DateGame, Country, Level, Result) ";
    $sql .= "VALUES (NULL, ?, ?, ?, ?)";
    $stmt = $conn->stmt_init();
    if (!$stmt->prepare($sql)) {
      throw new Exception("Prepare statement failed. " . $stmt->error);
    }
    $stmt->bind_param("ssii", $dateGame, $country, $level, $result);
    if (!$stmt->execute()) {
      throw new Exception("Execute statement failed. " . $stmt->error);
    }
    $stmt->close();
    CloseDatabase($conn);
  } catch (Exception $e) {
    ManageException("SaveOneResult : " . $e->getMessage() . "\n");
    CloseDatabase($conn);
  }
}

// Function to get all the game results
function GetAllResult() {
  try {
    $conn = ConnectDatabase();
    if ($conn->connect_error) {
      throw new Exception("Error connecting to DB. " .
        $conn->connect_error);
    }
    $sql = "SELECT Ref as Ref, ";
    $sql .= "DateGame as DateGame, ";
    $sql .= "Country as Country, ";
    $sql .= "Level as Level, ";
    $sql .= "Result as Result ";
    $sql .= "FROM AIwareStat ";
    $stmt = $conn->stmt_init();
    if (!$stmt->prepare($sql)) {
      throw new Exception("Prepare statement failed. " . $stmt->error);
    }
    if (!$stmt->execute()) {
      throw new Exception("Execute statement failed. " . $stmt->error);
    }
    $row = array();
    $stmt->bind_result(
      $row["Ref"],
      $row["DateGame"],
      $row["Country"],
      $row["Level"],
      $row["Result"]);
    $ret = array();
    while ($stmt->fetch()) {
      // Clone the returned array
      $narr = array();
      foreach ($row as $k => $v) {
        $narr[$k] = $v ;
      }
      $ret[] = $narr;
    }
    $stmt->close();
    CloseDatabase($conn);
    return $ret;
  } catch (Exception $e) {
    ManageException("GetAllResult : " . $e->getMessage() . "\n");
    CloseDatabase($conn);
  }
}

// Function to get win rate for a level
function GetWinRate($level) {
  try {
    $conn = ConnectDatabase();
    if ($conn->connect_error) {
      throw new Exception("Error connecting to DB. " .
        $conn->connect_error);
    }
    // Get the total number of games
    $sqlAll = "SELECT COUNT(Ref) ";
    $sqlAll .= "FROM AIwareStat ";
    $sqlAll .= "WHERE Level = ? ";
    $stmt = $conn->stmt_init();
    if (!$stmt->prepare($sqlAll)) {
      throw new Exception("Prepare statement failed. " . $stmt->error);
    }
    $stmt->bind_param("i", $level);
    if (!$stmt->execute()) {
      throw new Exception("Execute statement failed. " . $stmt->error);
    }
    $nbGame = 0;
    $stmt->bind_result($nbGame);
    $stmt->fetch();
    // Get the number of tie
    $sqlTie = $sqlAll . "AND Result = 3";
    if (!$stmt->prepare($sqlTie)) {
      throw new Exception("Prepare statement failed. " . $stmt->error);
    }
    $stmt->bind_param("i", $level);
    if (!$stmt->execute()) {
      throw new Exception("Execute statement failed. " . $stmt->error);
    }
    $nbTie = 0;
    $stmt->bind_result($nbTie);
    $stmt->fetch();
    // Get the number of Human win
    $sqlHuman = $sqlAll . "AND Result = 1";
    if (!$stmt->prepare($sqlHuman)) {
      throw new Exception("Prepare statement failed. " . $stmt->error);
    }
    $stmt->bind_param("i", $level);
    if (!$stmt->execute()) {
      throw new Exception("Execute statement failed. " . $stmt->error);
    }
    $nbHuman = 0;
    $stmt->bind_result($nbHuman);
    $stmt->fetch();
    // Get the number of A.I. win
    $sqlAI = $sqlAll . "AND (Result = 0 OR Result = 2)";
    if (!$stmt->prepare($sqlAI)) {
      throw new Exception("Prepare statement failed. " . $stmt->error);
    }
    $stmt->bind_param("i", $level);
    if (!$stmt->execute()) {
      throw new Exception("Execute statement failed. " . $stmt->error);
    }
    $nbAI = 0;
    $stmt->bind_result($nbAI);
    $stmt->fetch();
    $stmt->close();
    CloseDatabase($conn);
    $ret = array();
    $ret[0] = $nbGame;
    $ret[1] = $nbTie;
    $ret[2] = $nbHuman;
    $ret[3] = $nbAI;
    return $ret;
  } catch (Exception $e) {
    ManageException("GetWinRate : " . $e->getMessage() . "\n");
    CloseDatabase($conn);
  }
}

// Function to get access rate by day
function GetAccessRateByDay() {
  try {
    $conn = ConnectDatabase();
    if ($conn->connect_error) {
      throw new Exception("Error connecting to DB. " .
        $conn->connect_error);
    }
    // Get the total number of games
    $sql = "SELECT YEAR(DateGame) as Year, ";
    $sql .= "MONTH(DateGame) as Month, ";
    $sql .= "DAY(DateGame) as Day, COUNT(Ref) as Nb ";
    $sql .= "FROM AIwareStat ";
    $sql .= "GROUP BY YEAR(DateGame), MONTH(DateGame), ";
    $sql .= "DAY(DateGame) ";
    $sql .= "ORDER BY YEAR(DateGame) DESC, MONTH(DateGame) DESC, ";
    $sql .= "DAY(DateGame) DESC ";
    $sql .= "LIMIT 30 ";
    $stmt = $conn->stmt_init();
    if (!$stmt->prepare($sql)) {
      throw new Exception("Prepare statement failed. " . $stmt->error);
    }
    if (!$stmt->execute()) {
      throw new Exception("Execute statement failed. " . $stmt->error);
    }
    $row = array();
    $stmt->bind_result(
      $row["Year"],
      $row["Month"],
      $row["Day"],
      $row["Nb"]);
    $ret = array();
    while ($stmt->fetch()) {
      // Clone the returned array
      $narr = array();
      foreach ($row as $k => $v) {
        $narr[$k] = $v ;
      }
      $ret[] = $narr;
    }
    $stmt->close();
    CloseDatabase($conn);
    return $ret;
  } catch (Exception $e) {
    ManageException("GetAccessRateByDay : " . $e->getMessage() . "\n");
    CloseDatabase($conn);
  }
```

```
}

// Function to get access rate by day
function GetAccessRateByMonth() {
  try {
    $conn = ConnectDatabase();
    if ($conn->connect_error) {
      throw new Exception("Error connecting to DB. " .
        $conn->connect_error);
    }
    // Get the total number of games
    $sql = "SELECT YEAR(DateGame) as Year, ";
    $sql .= "MONTH(DateGame) as Month, COUNT(Ref) as Nb ";
    $sql .= "FROM AIwareStat ";
    $sql .= "GROUP BY YEAR(DateGame), MONTH(DateGame) ";
    $sql .= "ORDER BY YEAR(DateGame) DESC, MONTH(DateGame) DESC ";
    $sql .= "LIMIT 12 ";
    $stmt = $conn->stmt_init();
    if (!$stmt->prepare($sql)) {
      throw new Exception("Prepare statement failed. " . $stmt->error);
    }
    if (!$stmt->execute()) {
      throw new Exception("Execute statement failed. " . $stmt->error);
    }
    $row = array();
    $stmt->bind_result(
      $row["Year"],
      $row["Month"],
      $row["Nb"]);
    $ret = array();
    while ($stmt->fetch()) {
      // Clone the returned array
      $narr = array();
      foreach ($row as $k => $v) {
        $narr[$k] = $v ;
      }
      $ret[] = $narr;
    }
    $stmt->close();
    CloseDatabase($conn);
    return $ret;
  } catch (Exception $e) {
    ManageException("GetAccessRateByMonth : " . $e->getMessage() . "\n");
    CloseDatabase($conn);
  }
}

// Function to get access rate by country
function GetAccessRateByCountry() {
  try {
    $conn = ConnectDatabase();
    if ($conn->connect_error) {
      throw new Exception("Error connecting to DB. " .
        $conn->connect_error);
    }
    // Get the total number of games
    $sql = "SELECT Country as Country, ";
    $sql .= "COUNT(Ref) as Nb ";
    $sql .= "FROM AIwareStat ";
    $sql .= "GROUP BY Country ";
    $sql .= "ORDER BY Nb DESC ";
    $stmt = $conn->stmt_init();
```

```
      if (!$stmt->prepare($sql)) {
        throw new Exception("Prepare statement failed. " . $stmt->error);
      }
      if (!$stmt->execute()) {
        throw new Exception("Execute statement failed. " . $stmt->error);
      }
      $row = array();
      $stmt->bind_result(
        $row["Country"],
        $row["Nb"]);
      $ret = array();
      while ($stmt->fetch()) {
        // Clone the returned array
        $narr = array();
        foreach ($row as $k => $v) {
          $narr[$k] = $v ;
        }
        $ret[] = $narr;
      }
      $stmt->close();
      CloseDatabase($conn);
      return $ret;
    } catch (Exception $e) {
      ManageException("GetAccessRateByCountry : " . $e->getMessage() . "\n");
      CloseDatabase($conn);
    }
  }


?>
```

4.3 requestMove.php

```
<?php
 /* ============ requestMove.php =========== */
 // Ensure no message will interfere with output
 ini_set('display_errors', 'Off');
 error_reporting(0);

 // Turn on display of errors and warning for debug
 /*ini_set('display_errors', 'On');
 error_reporting(E_ALL ^ E_WARNING);
 error_reporting(E_ALL | E_STRICT);*/

 // Start the PHP session
 session_start();

 // Include the PHP files
 include("./db.php");

 try {
   // Sanitize args
   if (isset($_GET["arg"])) {
     $match = preg_match("/^[0-9 ]+$/", $_GET["arg"]);
     if ($match == 0) {
       $_GET["arg"] = "";
     }
   } else {
     $_GET["arg"] = "";
   }
   if ($_GET["arg"] != "") {
      // Create the command
      $cmd = "./AIware ". $_GET["arg"];
      // Execute the command
      unset($output);
      unset($returnVal);
      exec($cmd, $output, $returnVal);
      // Prepare the returned data
      $data["return"] = $returnVal;
      if ($returnVal == 0) {
        $data["error"] = "";
        $data["move"] = $output;
      } else {
        $data["error"] = "binary failure " . $returnVal;
        $data["move"] = -1;
      }
    } else {
      $data = array();
      $data["error"] = "no arguments";
      $data["move"] = -1;
      $data["return"] = 0;
    }
    // Convert the object to JSON format
    $ret = json_encode($data);
    // Return the JSON formatted result
    echo $ret;
  } catch (Exception $e) {
     ManageException("requestMove.php " . $e);
  }
?>
```

4.4 updateStat.php

```
<?php
 /* ============ updateStat.php =========== */
 // Start the PHP session
 session_start();

 // Ensure no message will interfere with output
 ini_set('display_errors', 'Off');
 error_reporting(0);

 // Turn on display of errors and warning for debug
 /*ini_set('display_errors', 'On');
 error_reporting(E_ALL ^ E_WARNING);
 error_reporting(E_ALL | E_STRICT);*/

 // Include the PHP files
 include("./db.php");

 try {
   // Check arguments validity
   if (isset($_GET["l"]) && isset($_GET["c"]) &&
     is_numeric($_GET["l"]) && is_numeric($_GET["c"]) &&
     $_GET["l"] >= 0 && $_GET["l"] <= 3 &&
     $_GET["c"] >= 0 && $_GET["c"] <= 3) {
     // Get the date
     $dateGame = date("Y-m-d H:i:s");
     // Get the country of the user
     $country = GetCountryFromIP();
     // Save the result
     SaveOneResult($dateGame, $country, $_GET["l"], $_GET["c"]);
```

```
    }
  } catch (Exception $e) {
     ManageException("updateStat.php " . $e);
  }
?>
```

4.5 aiware.js

```
/* ============= aiware.js =========== */

// The holes' indexes start at 0 from top-right and increment CCW

// ------------ Global variables
var theAIware = {};
var idAI = 0;
var idHuman = 1;
var nbPlayer = 2;
var nbHolePlayer = 6;
var nbHole = nbPlayer * nbHolePlayer;
var nbStoneHoleInit = 4;
var nbStone = nbHole * nbStoneHoleInit;
var gameRunning = 0;
var gameOver = 1;
var gameWaiting = 2;
var gameChecking = 3;
var zIndexSky = nbStone + 1;
var handTickInterval = 100;
var stoneMaxSpeed = 30.0;
var incSpeed = 0.5;
var preloadImg = new Array();

// ------------ AIware: main class

function AIware() {
  try {
    // Create arrays
    this._score = new Array();
    this._level = 0;
    this._holes = new Array();
    this._stones = new Array();
    // Init variables about the board graphics
    this._holeSize = 35.0;
    this._sizeStone = 30.0;
    // Init a new game
    this._nbTurn = 0;
    this.InitNewGame();
  } catch (err) {
    console.log("AIware " + err.stack);
  }
}

// ------------ Init a fresh new game

AIware.prototype.InitNewGame = function() {
  try {
    // Init the level
    this._level = $("#selLevel").val();
    console.log("start a new game at level " + this._level);
    // Init the number of turn
    if (this._nbTurn > 2) {
      console.log("Human resigns");
      this.HumanResign();
    }
    this._nbTurn = 0;
    // Init the score
    this._score[idHuman] = 0;
    this._score[idAI] = 0;
    // Init the game status
    this._status = gameRunning;
    // Init the current player
    this._curPlayer = Math.round(Math.random());
    // Init the holes
    for (var iHole = 0; iHole < nbHole; iHole++) {
      this._holes[iHole] = {};
      this._holes[iHole]._nbStone = 0;
      this._holes[iHole]._stones = new Array();
    }
    this._selectedHole = -1;
    // Init the stones
    var iHole = 0;
    var jStone = 0;
    for (var iStone = 0; iStone < nbStone; ++iStone) {
      this._stones[iStone] = {};
      this._stones[iStone]._index = iStone;
      this._stones[iStone]._pos = this.GetFreePosHole(iHole);
      this._stones[iStone]._moveTo = {};
      this._stones[iStone]._moveTo._x =
        this._stones[iStone]._pos._x;
      this._stones[iStone]._moveTo._y =
        this._stones[iStone]._pos._y;
      this._stones[iStone]._moveTo._z =
        this._stones[iStone]._pos._z;
      this._stones[iStone]._scale = 1.0;
      this._stones[iStone]._curHole = -1;
      this._stones[iStone]._moveToHole = -1;
      this._stones[iStone]._speed = 0.0;
      this.UpdateStonePos(iStone);
      this.AddStoneToHole(iStone, iHole);
      // Inc the nb of stones in this hole,
      // if it's full, go to next hole
      jStone++;
      if (jStone == nbStoneHoleInit) {
        jStone = 0;
        iHole++;
      }
    }
    // Init the hand
    this._hand = {};
    this._hand._stones = new Array();
    this._hand._nbStone = 0;
    this._hand._firstMovingStone = 0;
    this._hand._delayMove = 0.0;
    this._hand._lastMovingStone = 0;
    // Update the info
    this.UpdateInfo();
    // Fade in the board
    $("#divBoard").addClass("animated fadeIn");
    setTimeout(function(){
      $("#divBoard").removeClass("animated fadeIn");
    }, 1000);
    // If the AI starts call it now
    if (this._curPlayer == idAI) {
      this.RequestMoveFromAI();
```

```
    }
  } catch (err) {
    console.log("InitNewGame " + err.stack);
  }
}

// ----------- Update stat when human resigns

AIware.prototype.HumanResign = function() {
  try {
    // Prepare the arguments
    // level, resign cmd (0)
    var arg = "";
    arg += "l=" + this._level;
    arg += "&c=0";
    // Send the HTTP request
    this.UpdateStat(arg);
  } catch (err) {
    console.log("HumanResign " + err.stack);
  }
}

// ----------- Update stat when human wins

AIware.prototype.HumanWin = function() {
  try {
    // Prepare the arguments
    // level, human win cmd (1)
    var arg = "";
    arg += "l=" + this._level;
    arg += "&c=1";
    // Send the HTTP request
    this.UpdateStat(arg);
  } catch (err) {
    console.log("HumanWin " + err.stack);
  }
}

// ----------- Update stat when A.I. wins

AIware.prototype.AIWin = function() {
  try {
    // Prepare the arguments
    // level, A.I. win cmd (2)
    var arg = "";
    arg += "l=" + this._level;
    arg += "&c=2";
    // Send the HTTP request
    this.UpdateStat(arg);
  } catch (err) {
    console.log("AIWin " + err.stack);
  }
}

// ----------- Update stat when there was a tie

AIware.prototype.Tie = function() {
  try {
    // Prepare the arguments
    // level, tie cmd (3)
    var arg = "";
    arg += "l=" + this._level;
    arg += "&c=3";
    // Send the HTTP request
    this.UpdateStat(arg);
  } catch (err) {
    console.log("AIWin " + err.stack);
  }
}

// ----------- Update stat

AIware.prototype.UpdateStat = function(arg) {
  try {
    // Prepare the url for the PHP interfacing with the database
    url = "./updateStat.php?" + arg;
    // Create the HTTP request entity
    if (window.XMLHttpRequest) {
      xmlhttp = new XMLHttpRequest();
    } else {
      xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
    }
    xmlhttp.onreadystatechange = function() {
      if (xmlhttp.readyState == 4) {
        if (xmlhttp.status == 200) {
          // The request was successful, return the JSON data
          data = xmlhttp.responseText;
        } else {
          // The request failed, return error as JSON
          data ="{\"error\":\"HTTPRequest failed : " +
            xmlhttp.status +
            "\"}";
        }
      }
    };
    // Send the HTTP request
    xmlhttp.open("GET", url);
    xmlhttp.send();
  } catch (err) {
    console.log("UpdateStat " + err.stack);
  }
}

// ----------- Update the position of a stone

AIware.prototype.UpdateStonePos = function(iStone) {
  try {
    var stone = this._stones[iStone];
    // Get the vector to destination
    var v = new Array();
    v[0] = stone._moveTo._x - stone._pos._x;
    v[1] = stone._moveTo._y - stone._pos._y;
    // Get the distance ot destination
    var l = Math.sqrt(v[0] * v[0] + v[1] * v[1]);
    // If we haven't reached destination yet
    if (l > 1.0) {
      // Increase speed up to half the dist to destination per tick
      // else decrease
      if (2.0 * stone._speed < l) {
        stone._speed += incSpeed;
      } else {
        stone._speed -= incSpeed;
      }
      // Ensure the speed is never more than max speed
      if (stone._speed > stoneMaxSpeed) {
        stone._speed = stoneMaxSpeed;
      }
      // Ensure the speed is never more than half the dist
      // to destination per tick
      if (stone._speed * 2.0 > l) {
        stone._speed = l * 0.5;
      }
      // Normalise vector to destination
      v[0] = v[0] / l;
      v[1] = v[1] / l;
      v[2] = v[2] / l;
      // Move the stone toward destination at current speed
      stone._pos._x += v[0] * stone._speed;
      stone._pos._y += v[1] * stone._speed;
    } else {
      // If the stone has reached destination
      // Stop it and set its position to exactly the destination
      stone._pos._x = stone._moveTo._x;
      stone._pos._y = stone._moveTo._y;
      stone._speed = 0.0;
    }
    // If the stone is moving, set is zIndex to sky to be sure it will
    // be displayed above other stones. Not perfect but enough.
    if (stone._speed > 1.0) {
      stone._pos._z = zIndexSky;
    } else if (l < this._holeSize) {
      // If the stone enter its hole destination, set its z-index
      // to its destination z-index
      stone._pos._z = stone._moveTo._z;
    }
    // Update the div's css properties to current position
    var divStone =
      document.getElementById("divStone" + iStone);
    var x = this._stones[iStone]._pos._x - 0.5 * this._sizeStone;
    var y = this._stones[iStone]._pos._y - 0.5 * this._sizeStone;
    divStone.style.left = x + "px";
    divStone.style.top = y + "px";
    divStone.style.zIndex = Math.floor(this._stones[iStone]._pos._z);
    // Special effect, scale the stone make it go up and down
    // along a nice curve
    var s = 35.0 * (1.0 + stone._speed / stoneMaxSpeed);
    divStone.style.width = s + "px";
    divStone.style.height = s + "px";
  } catch (err) {
    console.log("UpdateStonePos " + err.stack);
  }
}

// ----------- Add a stone to a hole

AIware.prototype.AddStoneToHole = function(iStone, iHole) {
  try {
    this._holes[iHole]._stones[this._holes[iHole]._nbStone] =
      iStone;
    (this._holes[iHole]._nbStone)++;
    this._stones[iStone]._curHole = iHole;
  } catch (err) {
    console.log("AddStoneToHole " + err.stack);
  }
}
```

```
// ----------- Remove the top stone from a hole

AIware.prototype.RemoveTopStoneFromHole = function(iHole) {
  var iStone = -1;
  try {
    if (this._holes[iHole]._nbStone > 0) {
      (this._holes[iHole]._nbStone)--;
      iStone = this._holes[iHole]._stones[this._holes[iHole]._nbStone];
      this._stones[iStone]._curHole = -1;
    }
  } catch (err) {
    console.log("RemoveTopStoneFromHole " + err.stack);
  }
  return iStone;
}

// ----------- Get a free position (x, y, zIndex) in a hole

AIware.prototype.GetFreePosHole = function(iHole) {
  var pos = {};
  pos._x = 0;
  pos._y = 0;
  pos._z = 0;
  try {
    pos._z = this._holes[iHole]._nbStone;
    var c = this.GetCenterPosHole(iHole);
    var r = 0.0;
    if (pos._z > 0) {
      if (pos._z < 9) {
        r = this._sizeStone * 0.5;
      } else if (pos._z < 18) {
        r = this._sizeStone;
      } else {
        r = this._sizeStone * Math.random();
      }
    }
    var theta = 6.2831 * Math.random();
    pos._x = c._x + r * Math.cos(theta);
    pos._y = c._y + r * Math.sin(theta);
  } catch (err) {
    console.log("GetFreePosHole " + err.stack);
  }
  return pos;
}

// ----------- Get the center position (x, y) of a hole

AIware.prototype.GetCenterPosHole = function(iHole) {
  var pos = {};
  pos._x = 0;
  pos._y = 0;
  try {
    var iPlayer = Math.floor(iHole / nbHolePlayer);
    var jHole = iHole % nbHolePlayer;
    pos._x = Math.floor(105 + jHole * 100);
    if (iPlayer == 0) {
      pos._x = Math.floor(105 + (nbHolePlayer - jHole - 1) * 100);
    } else {
      pos._x = Math.floor(105 + jHole * 100);
    }
    pos._y = Math.floor(105 + iPlayer * 100);
  } catch (err) {
    console.log("GetCenterPosHole " + err.stack);
  }
  return pos;
}

// ----------- Update the content of the info div

AIware.prototype.UpdateInfo = function() {
  try {
    // Update the score
    var score = "";
    score += "Score: A.I.(" + this._score[idAI] + ") - ";
    score += "Human(" + this._score[idHuman] + ")";
    $("#divScore").html(score);
    // Update the turn
    var turn = "";
    if (this._status == gameOver) {
      turn += "Game Over ! ";
      if (this._score[0] == this._score[1]) {
        turn += "Tie";
      } else if (this._score[0] > this._score[1]) {
        turn += "A.I.ware wins";
      } else {
        turn += "Human wins";
      }
      $("#divTurn").addClass("animated tada");
      setTimeout(function(){
        $("#divTurn").removeClass("animated tada");
      }, 1000);
    } else if (this._curPlayer == idHuman) {
      if (this._status == gameWaiting ||
        this._status == gameChecking) {
        turn += "please wait";
      } else {
        turn += "it's Human turn";
      }
    } else {
      if (this._status == gameWaiting ||
        this._status == gameChecking) {
        turn += "please wait";
      } else {
        turn += "it's A.I.ware turn";
      }
    }
    $("#divTurn").html(turn);
    // Refresh the info about the selected hole
    var html = this.GetInfoSelectedHole();
    $("#divInfoHole").html(html);
  } catch (err) {
    console.log("UpdateInfo " + err.stack);
  }
}

// ----------- Get the info to display about the selected hole

AIware.prototype.GetInfoSelectedHole = function() {
  var html = "";
  try {
    if (this._selectedHole != -1) {
      html = "Hole #" + this._selectedHole + " contains " +
        this._holes[this._selectedHole]._nbStone + " stone(s).";
    }
  } catch (err) {
    console.log("GetInfoSelectedHole " + err.stack);
  }
  return html;
}

// ------------ Click on a hole

AIware.prototype.ClickHole = function(iHole) {
  try {
    // If the A.I. is playing
    if (this._curPlayer != idHuman) {
      // Alarm the human he has to wait
      $("#divTurn").addClass("animated swing");
      setTimeout(function(){
        $("#divTurn").removeClass("animated swing");
      }, 1000);
    } else {
      // If the game is ready to receive the human command
      if (this._status == gameRunning) {
        // If the human command is valid
        if (iHole >= nbHolePlayer && iHole < 2 * nbHolePlayer) {
          this.PlayMove(iHole);
        }
      } else {
        // If the hand is busy moving stones or checking board
        // Alarm the human he has to wait
        $("#divTurn").addClass("animated swing");
        setTimeout(function(){
          $("#divTurn").removeClass("animated swing");
        }, 1000);
      }
    }
  } catch (err) {
    console.log("ClickHole " + err.stack);
  }
}

// ------------ Play a move

AIware.prototype.PlayMove = function(iHole) {
  try {
    (this._nbTurn)++;
    console.log("#" + this._nbTurn + " Player " + this._curPlayer +
      " plays hole " + iHole);
    this._status = gameWaiting;
    // Loop on stone in played hole
    var iStone = this.RemoveTopStoneFromHole(iHole);
    var shiftKrou = 0;
    while (iStone != -1) {
      // Put the stone in hand
      this._hand._stones[this._hand._nbStone] = iStone;
      (this._hand._nbStone)++;
      // Set the destination of the stone
      var destHole = iHole + this._hand._nbStone + shiftKrou;
      while (destHole >= nbHole) {
        destHole -= nbHole;
      }
      // Jump over starting hole (so called Krou)
      if (destHole == iHole) {
        shiftKrou++;
        destHole++;
        while (destHole >= nbHole) {
          destHole -= nbHole;
        }
      }
      this._stones[iStone]._moveTo = this.GetFreePosHole(destHole);
      this._stones[iStone]._moveToHole = destHole;
      // Take another stone
      iStone = this.RemoveTopStoneFromHole(iHole);
    }
    // Start moving the first stone
    this._hand._firstMovingStone = 0;
    this._hand._lastMovingStone = 0;
    this.UpdateInfo();

  } catch (err) {
    console.log("ClickHole " + err.stack);
  }
}

// ------------ Capture stones in a hole

AIware.prototype.CaptureStone = function(iHole, iPlayer) {
  try {
    if (this._holes[iHole]._nbStone > 0) {
      console.log("Player " + iPlayer +
        " captures stones of hole #" + iHole);
      // Loop on stones in hole
      var iStone = this.RemoveTopStoneFromHole(iHole);
      while (iStone != -1) {
        // Put the stone in hand
        this._hand._stones[this._hand._nbStone] = iStone;
        (this._hand._nbStone)++;
        // Set the destination of the stone
        this._stones[iStone]._moveTo = this.GetPosPlayer(iPlayer);
        this._stones[iStone]._moveToHole = -1;
        // Increase the score
        (this._score[iPlayer])++;
        // Take another stone
        iStone = this.RemoveTopStoneFromHole(iHole);
      }
      // Start moving the first stone
      this._hand._firstMovingStone = 0;
      this._hand._delayMove = 0.0;
      this._hand._lastMovingStone = 0;
      this.UpdateInfo();
    }
  } catch (err) {
    console.log("CaptureStone " + err.stack);
  }
}

// ------------ Get a position on the border where to store
// captured stones

AIware.prototype.GetPosPlayer = function(iPlayer) {
  var pos = {}
  pos._x = 0;
  pos._y = 0;
  pos._z = 0;
  try {
    if (iPlayer == 0) {
      pos._x = 650.0 - this._score[iPlayer] * this._sizeStone * 0.5;
      pos._y = 30.0;
    } else {
      pos._x = 50.0 + this._score[iPlayer] * this._sizeStone * 0.5;
      pos._y = 280.0;
    }
    pos._z = this._score[iPlayer];
  } catch (err) {
    console.log("GetPosPlayer " + err.stack);
  }
  return pos;
}

// ----------- Get the hole index from x,y pos in document

AIware.prototype.GetHoleAtPos = function(x, y) {
  var ret = -1;
  try {
    for (var iPlayer = 0; iPlayer < nbPlayer; iPlayer++) {
      for (var iHole = 0; iHole < nbHolePlayer; iHole++) {
        var cx = 105 + iHole * 100;
        var cy = 105 + iPlayer * 100;
        var d = Math.sqrt((cx - x) * (cx - x) +
          (cy - y) * (cy - y));
        if (d < this._holeSize) {
          ret = iHole + iPlayer * nbHolePlayer;
          if (ret < nbHolePlayer) {
            ret = nbHolePlayer - ret - 1;
          }
        }
      }
    }
  } catch (err) {
    console.log("GetHoleAtPos " + err.stack);
  }
  return ret;
}

// ----------- Move the stones currently in hand

AIware.prototype.HandMoveStone = function() {
  try {
    // Check for stone at destination
    var iStone = this._hand._stones[this._hand._lastMovingStone];
    var stone = this._stones[iStone];
    if (Math.abs(stone._pos._x - stone._moveTo._x) < 1.0 &&
      Math.abs(stone._pos._y - stone._moveTo._y) < 1.0) {
      // This stone has arrived at destination, stop it.
      stone._speed = 0.0;
      if (stone._moveToHole != -1) {
        // If it arrived at a hole, add it to this hole
        this.AddStoneToHole(iStone, stone._moveToHole);
      }
      // Update the index of moving stone in hand
      (this._hand._lastMovingStone)++;
    }
    // Update the moving stones' position
    for (var iStone = this._hand._lastMovingStone;
      iStone < this._hand._firstMovingStone; iStone++) {
      this.UpdateStonePos(this._hand._stones[iStone]);
    }
    // Start moving one more stone if there are yet unmoving one
    if (this._hand._firstMovingStone < this._hand._nbStone) {
      // Artificially delay the start for aesthetic purpose
      this._hand._delayMove += 0.2;
      if (this._hand._delayMove > this._hand._firstMovingStone) {
        (this._hand._firstMovingStone)++;
      }
    }
  } catch (err) {
    console.log("HandMoveStone " + err.stack);
  }
}

// ----------- Check the board status

AIware.prototype.CheckBoard = function() {
  try {
    // Check for captured stones
    var iOpp = 1 - this._curPlayer;
    var flagCapture = 0;
    if (this._hand._lastHole >= iOpp * nbHolePlayer &&
      this._hand._lastHole < (iOpp + 1) * nbHolePlayer) {
      for (var iHole = this._hand._lastHole;
        iHole >= iOpp * nbHolePlayer; iHole--) {
        if (this._holes[iHole]._nbStone == 2 ||
          this._holes[iHole]._nbStone == 3) {
          this.CaptureStone(iHole, this._curPlayer);
          this._status = gameWaiting;
          flagCapture= 1;
        } else {
          iHole = 0;
        }
      }
    }
    // Get the number of stones in opp holes
    var nbOppStone = 0;
    for (var iHole = iOpp * nbHolePlayer;
      iHole < (iOpp + 1) * nbHolePlayer; iHole++) {
      nbOppStone += this._holes[iHole]._nbStone;
    }
    // If the opponent is starving
    if (nbOppStone == 0) {
      if (flagCapture == 1) {
        // If there has been captured stones, it means the current
        // player has starved the opponent. The current player looses.
        console.log("Player " + this._curPlayer +
          " eliminated because he starved the opponent");
        this._score[iOpp] = 48;
        this._score[this._curPlayer] = 0;
        this.GameOver();
        this.UpdateInfo();
      } else {
        // If there was no captured stones, it means the opponent
        // starved itself. The current player catches all his own stones.
        for (var iHole = this._curPlayer * nbHolePlayer;
          iHole < (this._curPlayer + 1) * nbHolePlayer; iHole++) {
          this.CaptureStone(iHole, this._curPlayer);
        }
        console.log("Game ends because player " + iOpp +
          " has no more stones");
        this.GameOver();
        this.UpdateInfo();
      }
    } else if (this._score[0] > nbStone * 0.5 ||
        this._score[1] > nbStone * 0.5) {
      // If one of the player has captured more than half the stones
      // the game ends.
      console.log("Game ends by score");
      this.GameOver();
      this.UpdateInfo();
    } else if (flagCapture == 0) {
      // If there was no capture, resume the game with the next player
      this._status = gameRunning;
      this.NextPlayer();
      if (this._curPlayer == idAI) {
        // If the next player is the A.I., request its move
        this.RequestMoveFromAI();
      }
    }
  } catch (err) {
    console.log("CheckBoard " + err.stack);
  }
}

// ----------- Function called when the game ends

AIware.prototype.GameOver = function() {
  try {
    // If the game is not already over
    if (this._status != gameOver) {
      if (this._score[idAI] > this._score[idHuman]) {
        this.AIWin();
      } else if (this._score[idAI] < this._score[idHuman]) {
        this.HumanWin();
      } else {
        this.Tie();
      }
      // Set the status of the game to over
      this._status = gameOver;
      this._nbTurn = 0;
    }
  } catch (err) {
    console.log("GameOver " + err.stack);
  }
}

// ----------- Hand tick function

AIware.prototype.HandTick = function() {
  try {
    if (this._status == gameWaiting ||
      this._status == gameOver) {
      // if we have moving stones in hand
      if (this._hand._nbStone > 0) {
        if (this._hand._lastMovingStone != this._hand._nbStone) {
          // There is still moving stones
          this.HandMoveStone();
        } else {
          // No more moving stones
          // Empty the hand
          var iStone = this._hand._stones[this._hand._nbStone - 1];
          var stone = this._stones[iStone];
          this._hand._lastHole = stone._moveToHole;
          this._hand._nbStone = 0;
          this._hand._firstMovingStone = 0;
          this._hand._delayMove = 0.0;
          this._hand._lastMovingStone = 0;
        }
      } else {
        // No more moving stones in hand
        if (this._status == gameWaiting) {
          this._status = gameChecking;
        }
      }
    } else if (this._status == gameChecking) {
      // Check the board
      this.CheckBoard();
    }
  } catch (err) {
    console.log("HandTick " + err.stack);
  }
}

// ----------- Move turn to next player

AIware.prototype.NextPlayer = function() {
  try {
    if (this._curPlayer == idHuman) {
      this._curPlayer = idAI;
    } else {
      this._curPlayer = idHuman;
    }
    this.UpdateInfo();
    $("#divTurn").addClass("animated zoomIn");
    setTimeout(function(){
      $("#divTurn").removeClass("animated zoomIn");
    }, 1000);
  } catch (err) {
    console.log("NextPlayer " + err.stack);
  }
}

// ----------- Process data returned from AI

AIware.prototype.ProcessAIMove = function(data) {
  try {
    // Interpret data at JSON format
    PHPExecData = JSON.parse(data);
    // Get the played hole
    var iHole = parseInt(PHPExecData.move[0]);
    if (iHole != -1) {
      // Playe the A.I. move
      this.PlayMove(iHole);
    } else {
      $("#divTurn").html("A.I. failure: " + PHPExecData.error +
        " Sorry, start again.");
    }
  } catch (err) {
    console.log("ProcessAIMove " + err.stack);
  }
}
// ----------- Request move from A.I.

AIware.prototype.RequestMoveFromAI = function() {
  try {
    // Prepare the arguments
    // level score[0] score[1] nbStoneHole[0..11]
    var usrInput = "";
    usrInput += this._level;
    usrInput += " " + this._score[0];
    usrInput += " " + this._score[1];
    for (var iHole = 0; iHole < nbHole; iHole++) {
      usrInput += " " + this._holes[iHole]._nbStone;
    }
    // Prepare the url for the PHP interfacing with the binary executable
    url = "./requestMove.php?arg=" + usrInput;
    // Create the HTTP request entity
    if (window.XMLHttpRequest) {
      xmlhttp = new XMLHttpRequest();
    } else {
      xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
    }
    xmlhttp.onreadystatechange = function() {
      if (xmlhttp.readyState == 4) {
        if (xmlhttp.status == 200) {
          // The request was successful, return the JSON data
          data = xmlhttp.responseText;
        } else {
          // The request failed, return error as JSON
          data ="{\"move\":[\"-1\"],\"error\":\"HTTPRequest failed : " +
            xmlhttp.status + "\"}";
        }
        // Process the returned data from the binary executable
        theAIware.ProcessAIMove(data);
      }
    };
    // Send the HTTP request
    xmlhttp.open("GET", url);
    xmlhttp.send();
  } catch (err) {
    console.log("RequestMoveFromAI " + err.stack);
  }
}

// ----------- OnLoad function

function BodyOnLoad() {
  try {
    // Create the stones div
    for (var iStone = 0; iStone < nbStone; iStone++) {
      var div = document.createElement("div");
      div.setAttribute("class", "divStone");
      var id = "divStone" + iStone;
      div.setAttribute("id", id);
      var imgStone = "url('./Img/stone";
      if (iStone < 10) imgStone += "0";
      imgStone += iStone + ".gif')";
      div.style.backgroundImage = imgStone;
      //div.innerHTML = iStone;
      $("#divBoard").append(div);
    }
    // Create the AIware entity
    theAIware = new AIware();
    // Bind events
    document.onclick = documentOnClick;
    window.onbeforeunload = windowUnload;
    // Set tick function for animation
    setInterval(HandTick, handTickInterval);
    // Preload images for fast rendering
    for (var iImg = 0; iImg < nbHole; iImg++) {
      preloadImg[iImg] = new Image();
      preloadImg[iImg].src = "./Img/board" + iImg + ".jpg"
    }
    preloadImg[nbHole] = new Image();
    preloadImg[nbHole].src = "./Img/board.jpg"
  } catch (err) {
    console.log("BodyOnLoad " + err.stack);
  }
}

// ----------- function called when the user qui or refresh the page

function windowUnload() {
  try {
    if (theAIware._nbTurn > 2) {
      console.log("Human resigns");
      theAIware.HumanResign();
    }
  } catch (err) {
    console.log("windowUnload " + err.stack);
  }
}

// ----------- hook for theAIware.HandTick

function HandTick() {
  try {
    theAIware.HandTick();
  } catch (err) {
    console.log("HandTick " + err.stack);
  }
}

// ----------- event listener for click on document

function documentOnClick(event) {
  try {
    // Get the index of the clicked hole
    var divBoard = document.getElementById("divBoard");
    var x = event.clientX - divBoard.offsetLeft;
    var y = event.clientY - divBoard.offsetTop;
    var iHole = theAIware.GetHoleAtPos(x, y);
    var divBoard = document.getElementById('divBoard');
    if (iHole != -1) {
      // If the user has clicked on a hole,
      // If the user has clicked on the currently selected hole
      if (theAIware._selectedHole == iHole) {
        // Execute the second click
        documentOnSndClick(event);
        // Unselect hole
        theAIware._selectedHole = -1;
        // Refresh the board image
        divBoard.style.backgroundImage =
          "url('./Img/board.jpg')";
      } else {
        theAIware._selectedHole = iHole;
        divBoard.style.backgroundImage =
          "url('./Img/board" + iHole + ".jpg')";
      }
    } else {
      // If the user has click out of the holes,
      // cancel selected hole and refresh board image
      theAIware._selectedHole = -1;
      divBoard.style.backgroundImage =
        "url('./Img/board.jpg')";
    }
    // Update info about current selected hole
    var html = theAIware.GetInfoSelectedHole();
    $("#divInfoHole").html(html);
    CancelTxtSelection();
  } catch (err) {
    console.log("documentOnClick() " + err.stack);
  }
}

// ----------- event listener for second click on document

function documentOnSndClick(event) {
  try {
    // Get the index of the clicked hole
    var divBoard = document.getElementById("divBoard");
    var x = event.clientX - divBoard.offsetLeft;
    var y = event.clientY - divBoard.offsetTop;
    var iHole = theAIware.GetHoleAtPos(x, y);
    // If the user has clicked on the selected hole
    if (iHole != -1 && iHole == theAIware._selectedHole) {
      // Execute the click on the selected hole
      theAIware.ClickHole(iHole);
    }
    CancelTxtSelection();
  } catch (err) {
    console.log("divBoardOnSndClick() " + err.stack);
  }
}

// ----------- function to cancel text selection due to click on board

function CancelTxtSelection() {
  try {
    if (window.getSelection) {
      if (window.getSelection().empty) {
        window.getSelection().empty();
      } else if (window.getSelection().removeAllRanges) {
        window.getSelection().removeAllRanges();
      }
    } else if (document.selection) {
      document.selection.empty();
    }
  } catch (err) {
    console.log("CancelTxtSelection() " + err.stack);
  }
}

// ----------- function to display the rules

function ShowRules() {
  try {
    $("#divRules").css("visibility", "visible");
  } catch (err) {
    console.log("ShowRules() " + err.stack);
  }
}

// ----------- function to hide the rules
```

```
function HideRules() {
  try {
    $("#divRules").css("visibility", "hidden");
  } catch (err) {
    console.log("HideRules() " + err.stack);
  }
}
```

4.6 aiware.css

```
/* ============= aiware.css =========== */

body {
  background-color: #aaaaaa;
  color: #433126;
}

input[type="button"] {
  background-color: #fecb5e;
  box-shadow: 2px 2px 10px #888888;
  margin: 2px 5px;
  padding: 2px 4px;
  font: 13px sans-serif;
  text-decoration: none;
  border: 1px solid #fee9aa;
  border-radius: 5px;
  color: #624838;
  height: 24px;
}

select {
  background-color: #fecb5e;
  box-shadow: 2px 2px 10px #888888;
  margin: 2px 5px;
  padding: 2px 4px;
  font: 13px sans-serif;
  text-decoration: none;
  border: 1px solid #fee9aa;
  border-radius: 5px;
  color: #624838;
  font: 13px sans-serif;
}

#divMain {
  text-align: center;
}

#divTitle {
  text-align: center;
  font-size: 25px;
  margin: 10px;
}

#divFooter {
  text-align: center;
  font-size: 15px;
  margin: auto;
  margin-top: 20px;
}
```

```
#divBoard {
  position: relative;
  top: 0px;
  left: 0px;
  width: 710px;
  height: 310px;
  background-image: url("./Img/board.jpg");
  background-size: 100%;
  background-repeat: no-repeat;
  margin: auto;
  z-index: -1;
}

#divInfo {
  margin: auto;
  margin-top: 10px;
}

#divCmd {
  margin: auto;
  margin-top: 10px;
}

.divTool {
  width: 500px;
  height: 300px;
  border: 1px solid #888888;
  background-color: #cccccc;
  display: inline-block;
  vertical-align: middle;
}

.divCharts {
  margin-top: 20px;
}

.divChart {
  margin: 5px 10px;
  width: 220px;
  height: 120px;
  border: 1px solid #888888;
  background-color: #ffffff;
  display: inline-block;
  vertical-align: middle;
  overflow: auto;
}

.divStone {
  position: absolute;
  top: 0px;
  left: 0px;
  z-index: 0;
  width: 30px;
  height: 30px;
  background-size: 100%;
}

#divScore {
  display: inline-block;
  text-align: center;
  width: 185px;
}
```

```
#divTurn {
  display: inline-block;
  text-align: center;
  width: 185px;
}

#divInfoHole {
  display: inline-block;
  width: 200px;
}

#divRules {
  width: 100%;
  margin: auto;
  position: absolute;
  top: 0px;
  left: 0px;
  visibility: hidden;
}

#divRulesContent {
  margin: auto;
  margin-top: 40px;
  width: 600px;
  height: 400px;
  border: 1px solid #888888;
  background-color: #dddddd;
  box-shadow: 0px 0px 15px #555555;
  text-align: center;
  overflow: auto;
}

#imgRulesClose {
  width: 50px;
  height: 50px;
  float: right;
}

#divRulesTitle {
  font-size: 25px;
  margin-top: 20px;
  margin-bottom: 20px;
}

.divOneRule {
  margin: 10px;
  padding-left: 20px;
  text-align: left;
}

.imgRule {
  height: 200px;
}
```

5 Statistics

The user can check the winning rate of the A.I., the number of game played per day and month, and the country of Human player through a dedicated page.

5.1 showStat.php

```
<?php
/* ============ showStat.php =========== */
// Start the PHP session
session_start();

  // Ensure no message will interfere with output
  ini_set('display_errors', 'Off');
  error_reporting(0);

  // Turn on display of errors and warning for debug
  /*ini_set('display_errors', 'On');
  error_reporting(E_ALL ^ E_WARNING);
  error_reporting(E_ALL | E_STRICT);*/

  // Include the PHP files
  include("./db.php");

  try {
    // Get the win rates
    $winRate = array();
    for ($iLevel = 0; $iLevel < 4; $iLevel++) {
      $winRate[$iLevel] = GetWinRate($iLevel);
    }

    // Get the access rate
    $accessRateByDay = GetAccessRateByDay();
    $accessRateByMonth = GetAccessRateByMonth();
    $accessRateByCountry = GetAccessRateByCountry();
  } catch (Exception $e) {
    ManageException("updateStat.php " . $e);
  }

  // Function to create one gauge
  function DivGauge($label, $width, $perc) {
    try {
      $block = "";
      $block .= "<div style='width:" . $width . "px;";
      $block .= "text-align:center;border: 1px solid rgb(255, 0, 0);";
      $block .= "position:relative;top:0px;left:0px;'>";
      $block .= "<div style='position:absolute;top:0px;left:0px;";
      $block .= "width:" . $perc . "%;height:100%;";
      $block .= "background-color: rgba(255, 0, 0, 0.3);'></div>";
      $block .= $label;
      $block .= "</div>";
      return $block;
    } catch (Exception $e) {
      return "DivWinRate : " . $e->getMessage() . "<br>";
    }
  }

  // Function to create the div for one win rate
  function DivWinRate($rate) {
    try {
      $block = "";
      $block .= "<div style='display:inline-block;margin-top:10px;'>";
      if ($rate[0] == 0) $rate[0] = 1;
      $perc = $rate[2] / $rate[0] * 100.0;
      $block .= DivGauge("Human : " . number_format($perc, 0) . "%", 200, $perc);
      $perc = $rate[3] / $rate[0] * 100.0;
      $block .= DivGauge("A.I.ware : " . number_format($perc, 0) . "%", 200, $perc);
      $perc = $rate[1] / $rate[0] * 100.0;
      $block .= DivGauge("Tie : " . number_format($perc, 0) . "%", 200, $perc);
      $block .= "in " . $rate[0] . " games";
      $block .= "</div>";
      return $block;
    } catch (Exception $e) {
      return "DivWinRate : " . $e->getMessage() . "<br>";
    }
  }

  // Function to create the div for access by day and month
  function DivAccessRate($rate) {
    try {
      $block = "";
      $block .= "<div style='display:inline-block;margin-top:10px;'>";
      // Search the maximum access rate
      $max = 0;
      foreach ($rate as $row) {
        if ($max < $row["Nb"]) {
          $max = $row["Nb"];
        }
      }
      // Display the dates
      foreach ($rate as $row) {
        $label = $row["Year"] . "-" . $row["Month"];
        if (isset($row["Day"]))
          $label .= "-" . $row["Day"];
        $label .= ": " . $row["Nb"] . "<br>";
        $perc = $row["Nb"] / $max * 100.0;
        $block .= DivGauge($label, 200, $perc);
      }
      $block .= "</div>";
      return $block;
    } catch (Exception $e) {
      return "DivAccessRate : " . $e->getMessage() . "<br>";
    }
  }

  // Function to create the div for access by country
  function DivAccessRateByCountry($rate) {
    try {
      $block = "";
      $block .= "<div style='display:inline-block;margin-top:10px;'>";
      // Search the maximum access rate
      $max = 0;
      foreach ($rate as $row) {
        if ($max < $row["Nb"]) {
          $max = $row["Nb"];
        }
      }
      // Display the dates
      foreach ($rate as $row) {
        $label = $row["Country"];
        $label .= ": " . $row["Nb"] . "<br>";
        $perc = $row["Nb"] / $max * 100.0;
        $block .= DivGauge($label, 200, $perc);
      }
      $block .= "</div>";
      return $block;
    } catch (Exception $e) {
      return "DivAccessRate : " . $e->getMessage() . "<br>";
    }
  }
?>
<!DOCTYPE html>
<html>
  <head>

    <!-- Meta -->
    <meta content="text/html; charset=UTF-8;">
    <meta name="viewport"
      content="width=device-width, initial-scale=1, maximum-scale=1">

    <!-- Icon -->
    <link rel="icon" type="image/x-icon"
      href="./Img/aiware.ico" />

    <!-- Include the CSS files -->
    <link href = "./animate.css"
      rel = "stylesheet" type = "text/css">
    <link href = "./aiware.css"
      rel = "stylesheet" type = "text/css">

    <!-- Include the JS files -->
    <script charset = "UTF-8" src = "./jquery.min.js"></script>

    <title>A.I.ware</title>
  </head>
  <!--<body onload = 'BodyOnLoad();'>-->
    <!-- Main div -->
    <div id = "divMain">

      <!-- Title div -->
      <div id = "divTitle">
        A.I.ware
      </div>

      <div id = "divWinRate" class = "divTool">
        Win Rate
        <div id = "divCharts">
          <div class = "divChart">
            Beginner
<?php
  echo DivWinRate($winRate[0]);
?>
          </div>
          <div class = "divChart">
            Easy
<?php
  echo DivWinRate($winRate[1]);
?>
          </div>
          <div class = "divChart">
            Intermediate
<?php
  echo DivWinRate($winRate[2]);
?>
          </div>
          <div class = "divChart">
            Strong
<?php
  echo DivWinRate($winRate[3]);
?>
          </div>
        </div>
      </div>

      <div id = "divAccessRate" class = "divTool">
        Games played
        <div class = "divCharts">
          <div class = "divChart" style = "height: 240px;">
            By day
<?php
  echo DivAccessRate($accessRateByDay);
?>
          </div>
          <div class = "divChart" style = "height: 240px;">
            By month
<?php
  echo DivAccessRate($accessRateByMonth);
?>
          </div>
        </div>
      </div>

      <div id = "divAccessRateCountry" class = "divTool">
        Player's country
        <div class = "divCharts">
          <div class = "divChart" style = "height: 240px;">
<?php
  echo DivAccessRateByCountry($accessRateByCountry);
?>
          </div>
        </div>
      </div>

      <!-- footer div -->
      <div id = "divFooter">
        Copyright <a href="mailto:Pascal@BayashiInJapan.net">
            P. Baillehache
        </a>, 2017,
        <a href="index.php">A.I.ware</a>,
        <a href="doc.pdf">Documentation</a>.
      </div>

    </div>
  </body>

</html>
```

6 Install and use

All the sources in this document are provided for free (a little email to say thank you would be appreciated though). You can use, modify and distribute them as you want. I hope it will be useful as an example of how to make an online game for beginners, and as a simple framework toward more complex games for more advanced developpers.

All the files described in this document are provided ”as is”, without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall I be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with these files or the use or other dealings in these files.

A.I.ware has two external dependancies: JQuery which can be found here:
https://jquery.com/download/
and animate.css, wich can be found here:
https://daneden.github.io/animate.css/

All the files and images are available here:
http://www.BayashiInJapan.net/A.I.ware/AIware.tar.gz

Upload the files in the ’www’ folder to your server as follow:

```
/A.I.ware/:
AIware      aiware.js    db.php     jquery.min.js     showStat.php
aiware.css  animate.css  index.php  requestMove.php   updateStat.php

/A.I.ware/Img:
aiware.ico   board.jpg        stone00.gif  stone13.gif  stone26.gif  stone39.gif
board0.jpg   close.gif        stone01.gif  stone14.gif  stone27.gif  stone40.gif
board10.jpg  highlight1.gif   stone02.gif  stone15.gif  stone28.gif  stone41.gif
board11.jpg  rules01-1.gif    stone03.gif  stone16.gif  stone29.gif  stone42.gif
board1.jpg   rules01-2.gif    stone04.gif  stone17.gif  stone30.gif  stone43.gif
board2.jpg   rules02-1.gif    stone05.gif  stone18.gif  stone31.gif  stone44.gif
board3.jpg   rules03-1.gif    stone06.gif  stone19.gif  stone32.gif  stone45.gif
board4.jpg   rules03-2.gif    stone07.gif  stone20.gif  stone33.gif  stone46.gif
board5.jpg   rules04-1.gif    stone08.gif  stone21.gif  stone34.gif  stone47.gif
board6.jpg   rules04-2.gif    stone09.gif  stone22.gif  stone35.gif
board7.jpg   rules04-3.gif    stone10.gif  stone23.gif  stone36.gif
board8.jpg   rules05-1.gif    stone11.gif  stone24.gif  stone37.gif
board9.jpg   rules05-2.gif    stone12.gif  stone25.gif  stone38.gif
```

If your server is not running on Linux you will have to generate the binary executable AIware for your server: execute ’make’ in the ’C/’ directory and replace the AIware with the newly generated one. MAke sure that the AIware binary file has permission to execute.

After copying the files, set your database connection information in db.php and access the following link to create the table used for statistics:
http://yourwebsite.net/A.I.ware/?installDB=1

You are no ready to use A.I.ware:
http://yourwebsite.net/A.I.ware/

7 To do

Possible improvement:

- The border of the board is too neat, there should some more carves to cut the angle.
- Add a stronger level which would use a more elaborate evaluation function.
- Add sounds when stones fall into holes.