Assignment 2
Artificial Intelligence
EA C461

Submitted By: Gaurav Ahuja, 2008A3PS697

Table of Contents

Introduction

The Game is written is C++ and was successfully compiled and built using Microsoft Visual Studio 2008. The program depends on the following external libraries for Graphics handling:

1. SDL version 1.2.14 (stable)
2. SDL_image 1.2

Since the game is written in C++ and the external libraries are cross-platform the program should compile on platforms like linux and Mac OS X without much hassles.

The Game consists of mainly three Classes:

1. Stone
2. Board
3. AIPlayer

Stone Class

Each stone maintains the following list of properties:

1. X: The X coordinate of the Stone
2. Y: The Y coordinate of the Stone
3. Colour: The Colour of the Stone
4. History: History of Movements of the Stone
5. LastPostion: Coordinates of the last Stone Position
6. LastCapturedStones: Number of Stones captured in the last move
7. CapturedStones: Total Stones captured

The functions that assist the Stone Class are:

1. bool LegalMove(_Stone Stone, _Board Board, int NewX, int NewY)
   //Returns true if the move to NewX, NewY by the Stone is Legal else returns false

2. int CanMakeCapturingMove(_Stone Stone, _Board Board, int &X, int &Y, int &Action);
   //Returns the maximum number of stones that can be captured by any move from current location
   //If No Capturing move can be made 0 is returned

3. int CaptureStones(_Stone &Stone, _Board &Board, int NewX, int NewY, int Action, bool real);
   //Captures Stones on the Board by making a move to NewX, NewY by the action specified by Action which can be APPROACH or WITHDRAW
   //The Move should be Legal
   //The Stone and Board is only affected if the real variable is set to true
   //Return Value: Number of Stones Captured

4. list<_Point2D> Approach(_Stone Stone, _Board Board, int NewX, int NewY);
   //Returns the list of Stones that can be captured by Approaching the point NewX,NewY

5. list<_Point2D> Withdraw(_Stone Stone, _Board Board, int NewX, int NewY);
   //Returns the list of Stones that can be captured by Withdrawing to the point NewX,NewY

6. bool CanMove(_Stone Stone, _Board Board, int *x = NULL, int *y = NULL);
   //Returns true if the stone can Make Legal moves from its current postion

7. int MakeMove(_Stone &Stone, _Board &Board, int NewX, int NewY, bool real, int &NB);
   //Makes a move to the point NewX, NewY
   //Move has to be legal
   //The Stone and Board are only affected if the real variable is set to true
   //Returns the Action Taken
   //If the Return Value is MUSTCHOOSE(which will be less than 0) the player must decide between APPROACH or WITHDRAW

8. void Select(_Stone Stone, _Board &Board);
   //Selects the Stone on the Board and Unselects others Selected stones if any

9. bool IsSelected(_Stone Stone, _Board Board);
   //Returns true if the stone is selected on the board

10. `void UnSelect(_Stone Stone, _Board &Board);`
    `//Unselects the Stone on the Board`

11. `bool CanBeCaptured(_Stone Stone, _Board Board);`
    `//Returns true if the Stone can be captured by the opponent given the current board status`

12. `bool IsOnStrongPosition(_Stone Stone);`
    `//Returns true if the Stone is on a Strong Postion else false`

Board Class

The Board Class maintains the following properties:

1. Rows: Number of Rows on the board
2. Columns: Number of Columns on the board
3. Matrix[Rows][Columns]: Stores the position of stones on the board

The functions that assist the Board class are:

1. `void InitializeBoard(_Board &Board);`
   `//Initilizes the Board`

2. `void Populate(_Board &Board);`
   `//Stores the initial postions of the BLACK and WHITE stones in the matrix`

3. `void GetMatrix(_Board Board, int NewMatrix[5][9]);`
   `//Gets the 2D matrix`

4. `void PrintMatrix(_Board Board);`
   `//Prints the Matrix`

5. `void Show(_Board Board);`
   `//Shows the Board on the Screen`

6. `int GetStoneColour(_Board Board, int x, int y);`
   `//Returns the Colour of Stone at (x, y)`

7. `bool StoneExists(_Board Board, int x, int y);`
   `//Returns the true if Stone exists at (x, y)`

8. `void PopulateGUI(_Board Board, SDL_Surface *Screen);`
   `//Populates the SDL Surface with images of Stones`

9. `_Point2D GetSelectedPostion(_Board Board, _Point2D MousePos);`
   `//Returns the row and column for given mouse poisiton`

10. `void UnselectALL(_Board &Board);`
    `//Unselects all stones on the board`

11. `_Point2D GetSelectedStone(_Board Board, int &Colour);`
    `//Gets the Postion selected stone`

12. `int CheckWinner(_Board Board);`
    `//return BLACK, White of Empty for winner`

13. `bool IsOn(_Board Board, int xx, int yy);`
    `//returns if the row or col is on the board or not`

14. `void Copy(_Board OldBoard, _Board &NewBoard);`
    `//Copies OldBoard to New Board`

AIPlayer Class

The functions that assist this class are:

1. void AIPlay(_Board &Board, SDL_Surface* Screen, int AIColour, int Mode);
   //Plays a new Move the mode specifies the algorithm to be used
   //Currently only Greedy Algorithm is implemented

2. int GreedyPlay(_Board &Board, SDL_Surface* Screen, int AIColour);
   //Initiates the Greedy Algorithm

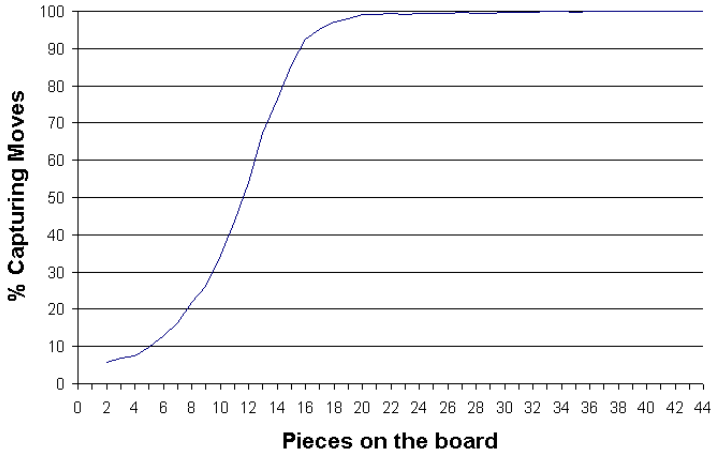
3. list<_Stone> GetFreeStones(int Colour, _Board Board);
   //Returns the list of stones that can be moved

4. _Stone SelectStone(_Board Board, int Colour, int &MoveX, int &MoveY, int &Action, int &Nb);
   //Selects a stone for the current move

Game Strategy

With reference to '*BEST PLAY IN FANORONA LEADS TO DRAW, MAARTEN P.D. SCHADD, MARK H.M. WINANDS, JOS W.H.M. UITERWIJK, H. JAAP VAN DEN HERIK and MAURICE H.J. BERGSMA, MICC-IKAT Games and AI Group, Faculty of Humanities and Sciences Universite it Maastricht, P.O. Box 616, 6200 MD Maastricht, The Netherlands*'.

The average game length of Fanorona is 43.81 plies and the average branching factor is 11.19 moves. This gives us a game-tree complexity of approximately 10^{46}. A typical game of Fanorona can be divided into two parts. In the first part of the game mostly capturing moves are played until most pieces are captured. In the second part, the endgame, mainly paika moves are played.

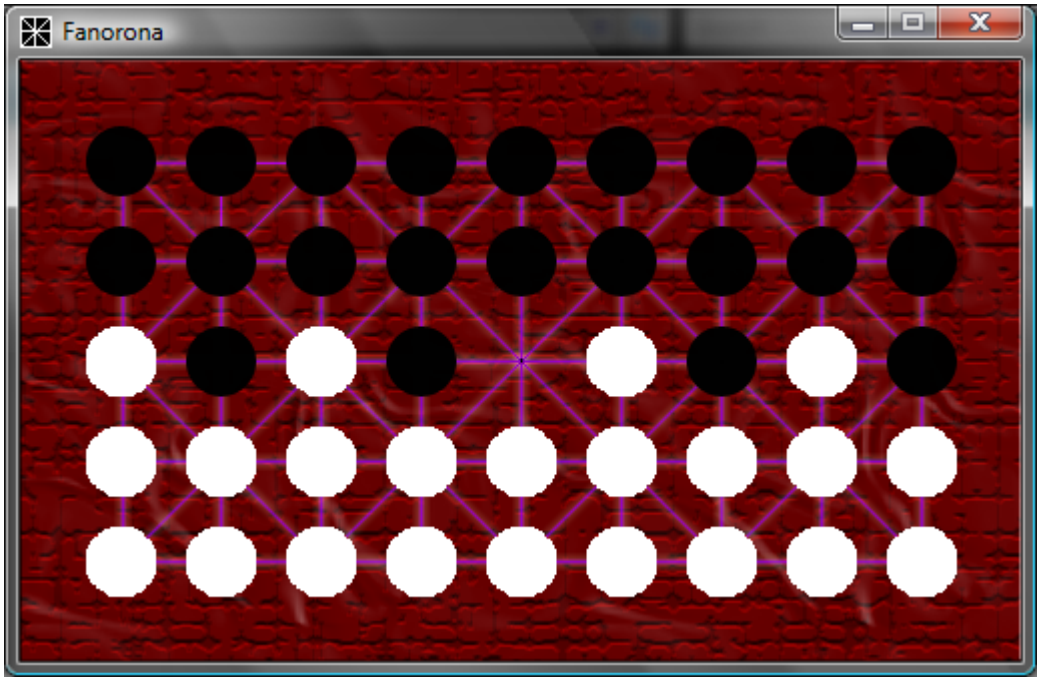


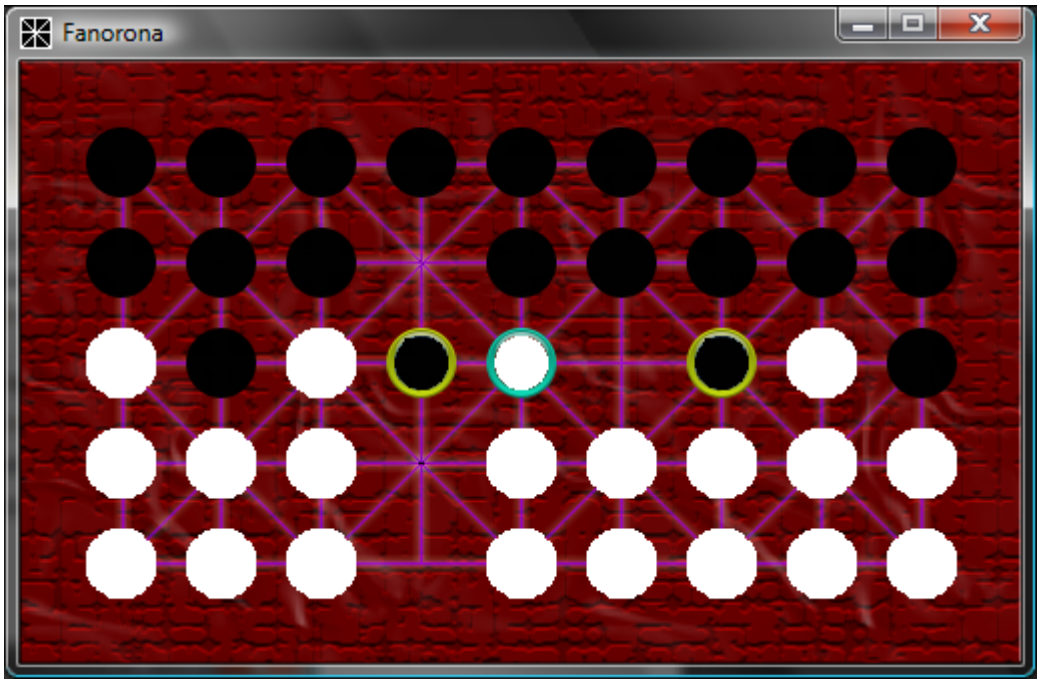
Since the first part of the game consist mostly of capturing moves it is better to play a move that can capture many stones. Thus a Greedy algorithm was chosen for the Game.

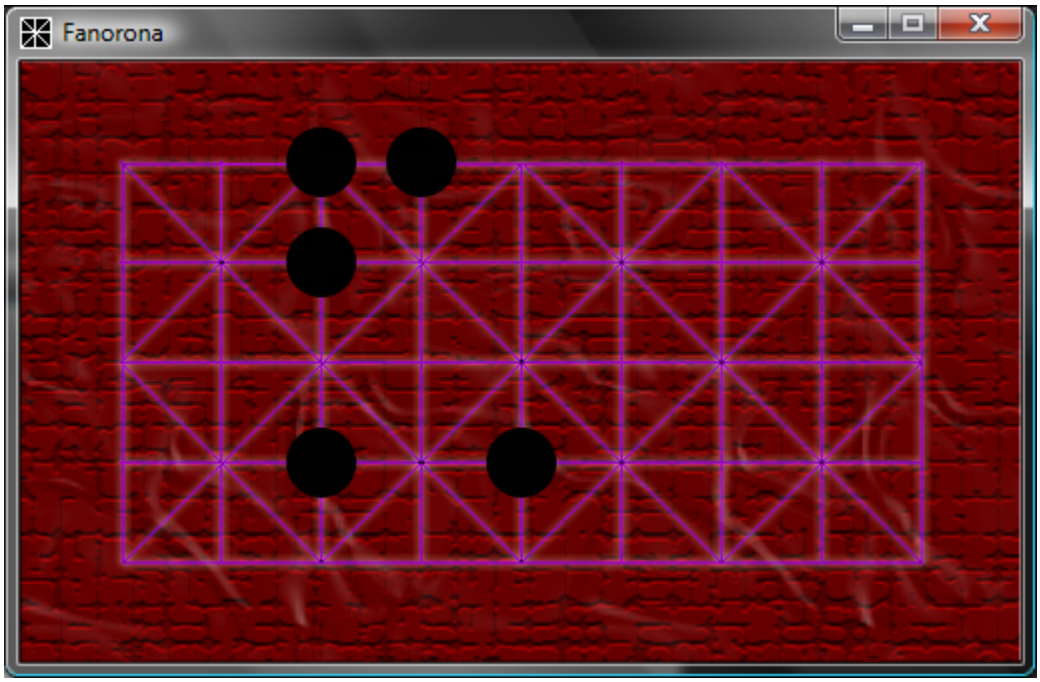
The Greedy Algorithm implemented in the game works as follows:

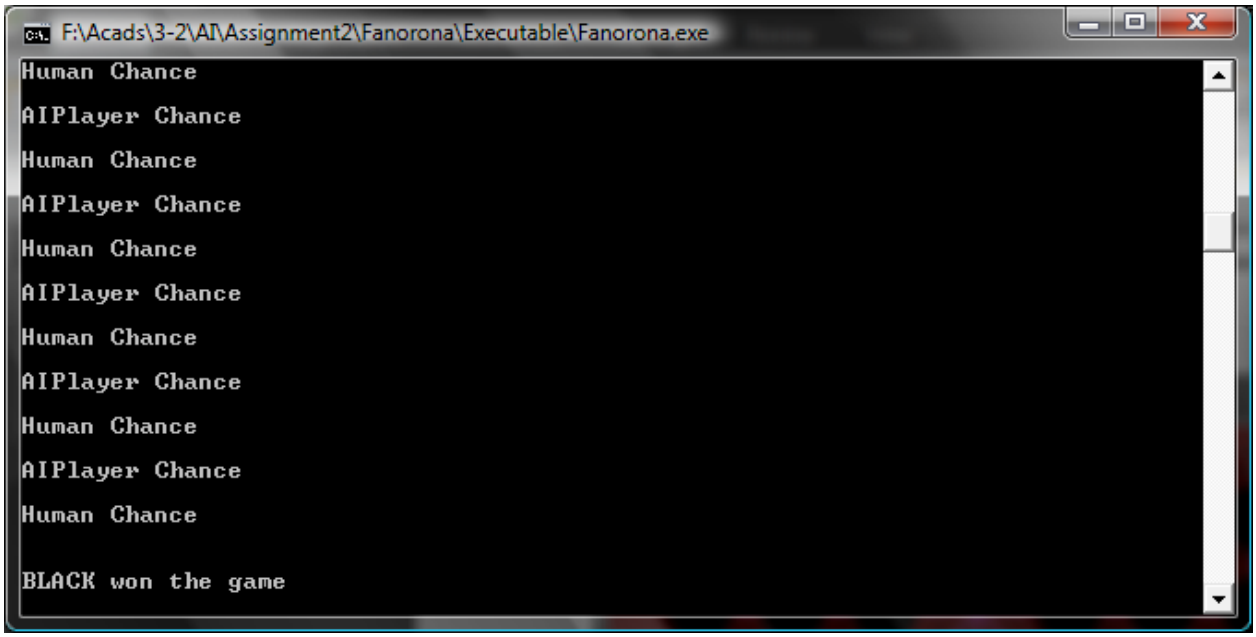
From the list of Stones that can be moved chose the stone that can capture maximum number of opponent stones. Once this stone is chosen moves are made until no capturing moves can be made, this is when the player loses his chance. If in case no stone can make any capturing move the stone that can be captured by the opponents stone is chosen to move.

Screenshots






Fanorona


F:\Acads\3-2\AI\Assignment2\Fanorona\Executable\Fanorona.exe
Human Chance
AIPlayer Chance
Human Chance
AIPlayer Chance
Human Chance
AIPlayer Chance
Human Chance
AIPlayer Chance
Human Chance
AIPlayer Chance
Human Chance
BLACK won the game