NATIONAL UNIVERSITY OF SINGAPORE


# HEXAREVERSI

## Project Description

**Group t77**

**Lu Si Hong** **A0094511L**
**Ling Xiao Xiong Shaun** **A0112036Y**

This document serves to bring attention to the reader about the approaches used in the implementation of Hexareversi and the way that the User Interface has been implemented, as well as introduces the playing algorithm in pseudo-code

# Table of Contents



# Introduction

In this project, our goal is to improve the existing implementations of a hexareversi game. Our task includes improving the user interface, provide a good playing algorithm, and provide a tournament option.

# User Interface Redesign

For the aesthetics of the Hexareversi page, editions were made to the Hexareversi%20Frame.htm.

Header title was enlarged (capitalised and bold) to make it more welcoming

```
<h2><b>HEXAREVERSI</b></h2>
```

Options-frame size was enlarged and recoloured (width, height, line thickness, and colour were added) to add a cooler feel to the game than a thin red line

* *

Delay of each move for the next round input option was added to allow the change of pace while the game is running. Before the game starts, it is taken as round-0. Thus, while a game is running, next round delay can be changed and the delay will be updated starting the next round.

*document.getElementById("delay").value = document.getElementById("ndelay").value;*

Number of rounds input option was added to allow the implementation a high number of rounds, especially during the testing phase where consistency was desired

* Number of rounds: <input id="rounds" value="1" type="text"> *

# Tournament Implementation

Tournament option, number of wins acquired by each player, number of pieces acquired by each player, and number of draws were added as part of the project requirements.

Utilizing the current function start(#) where it accepts a value that is determined to be the number of rounds, tournament calls the function with a value 10.

*<button onclick="start10(10)" id = "tenround">Tournament</button>*

Because of the need to swap player strategy being used by X and Y while not disturbing a normal calling of a 10-round game, start10(10) calls a separate game function.

*function Game10(container, playerX, playerY, poked)*

To swap the starting player, the algorithm pX is given the value of playerY after 5 games, and likewise, pY is given the value of playerX.

```
if (poke > 5)
{
	var pX = window[document.getElementById("playerX").value];
	var pY = window[document.getElementById("playerY").value];
}
if (poke <= 5)
{
	var pX = window[document.getElementById("playerY").value];
	var pY = window[document.getElementById("playerX").value];
}
```

Similarly, to ensure the correct algorithm is being rewarded their respective wins and pieces acquired, the values to be changed are swapped as well.

```
if (poked > 5)
{
	if (x > y)
	{
		document.getElementById("xscore").value =
Number(document.getElementById("xscore").value) + 1;
	}
	if (y > x)
	{
		document.getElementById("yscore").value =
Number(document.getElementById("yscore").value) + 1;
	}
	if (x == y)
	{
		document.getElementById("draw").value =
Number(document.getElementById("draw").value) + 1;
	}
				document.getElementById("xpiece").value =
Number(document.getElementById("xpiece").value) + x;
				document.getElementById("ypiece").value =
Number(document.getElementById("ypiece").value) + y;
}

if (poked <= 5)
{
	if (x > y){
		document.getElementById("yscore").value =
Number(document.getElementById("yscore").value) + 1;
	}
	if (y > x){
		document.getElementById("xscore").value =
Number(document.getElementById("xscore").value) + 1;
	}
	if (x == y){
		document.getElementById("draw").value =
Number(document.getElementById("draw").value) + 1;
	}
```

```
            document.getElementById("xpiece").value =
Number(document.getElementById("xpiece").value) + y;
            document.getElementById("ypiece").value =
Number(document.getElementById("ypiece").value) + x;
}
```

The display of number of wins, pieces acquired, draws, and winner are subsequently implemented in the Hexareversi%20Frame.htm, made to be read-only to prevent intrusion.

```
            <div>player <b><font color="#FF0000">X</font></b>: <input id="xscore" value="0"
type="text" readonly></div>

            <div><b><font color="#FF0000">X</font></b> pieces: <input id="xpiece" value="0"
type="text" readonly></div>

            <div>player <b><font color="#0000FF">Y</font></b>: <input id="yscore" value="0"
type="text" readonly></div>

            <div><b><font color="#0000FF">Y</font></b> pieces: <input id="ypiece" value="0"
type="text" readonly></div>

            <div>Winner is player: <input id="winner" type="text" readonly></div>

            <div>Draw: <input id="draw" value="0" type="text" readonly></div>
```

Colour coding is giving for X and Y for easier player identification.

```
<div><b><font color="#FF0000">X</font></b> for red

<div><b><font color="#0000FF">Y</font></b> for blue
```

## Algorithm for Player

For the algorithm, the algorithm works by applying 2 Heuristics and taking the best score for that.

These 2 heuristics are namely mobility and stability.

For mobility, the algorithm considers an analysis of **depth = 1.**

**Pseudo-Code:**

For all possible moves in the board,

1. Create a temporary board and place the move being considered in that board.(Call function updateBoard)
2. Count the number of moves that your opponent can have after you placed that piece. (call function seePossibleOpponentMove)
3. Set the optimal move to that of being the move that minimizes the number of your opponent's move.

Return optimal move if exists, else return 0(pass)

**Reasoning behind the mobility heuristic:**

If I am to win Reversi, I would want to minimize the possible moves available to my opponent. This will lead to the following 2 situations:

1. My opponent upon having his moves minimized, have to pass. By passing he lose all chances of getting more pieces

2. My opponent have no choice but to start choosing less optimal position, as his amount of choices has been reduced.

However, applying only the mobility heuristic is insufficient. This is because, the positions of the pieces were not taken into account, and corners are really good positions to have in Hexareversi.

So another heuristic was applied, namely the stability heuristic, where each location on the board is given a value, much like how t97 value is initialized.

However, we made some minor adjustments to the array.

```
var t77value = new Array(
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 9, 2, 6, 2, 9, 0, 0, 0, 0,
            0, 2, 1, 2, 2, 1, 2, 0, 0, 0,
            0, 6, 2, 4, 3, 4, 2, 6, 0, 0,
            0, 2, 2, 3, 3, 3, 3, 2, 2, 0,
            0, 9, 1, 4, 3, 3, 3, 4, 1, 9,
            0, 0, 2, 2, 3, 3, 3, 3, 2, 2,
            0, 0, 0, 6, 2, 4, 3, 4, 2, 6,
            0, 0, 0, 0, 2, 1, 2, 2, 1, 2,
            0, 0, 0, 0, 0, 9, 2, 6, 2, 9,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
```

**Corners**

- Corners are of greatest importance.
- No opposite-facing sides, capturable only by placing a piece on it.
- No risk.

**Stepping stone to achieve corner.**

- Sides against the edge only have one opposite face.
- Low risk.
- Value of 6

**Stepping stone to achieve corner.**

- Non-edge position have more than one opposite face.
- Easier to lose control if opponent dominants a stretch along the axis.
- Medium risk.
- Value of 4

**Positions between the corners and the middle of the edges are risky.**

- Counters the opponent's acquiring of a middle of an edge
- No value in achieving the corner it is near to
- Accelerates the opponents capturing of other vital positions if corner is taken
- Value of 2

The combination of these 2 heuristics was used through the use of a fraction. Since my goal is to both maximise the value of my next position, and minimize my opponent's move, the pseudo code is modified into

**Pseudo Code (modified):**

For all possible moves in the board:

1. Create a temporary board and place the move being considered in that board.(Call function updateBoard)
2. Count the number of moves that your opponent can have after you placed that piece.(Call function seePossibleOpponentMove)
3. Set the optimal move to that of being the move that minimizes the fraction of (the number of your opponent move) / (value of position considered).

Return optimal move if exists, else return 0(pass)