Michael (121110804)

Jansen (121110791)

Christian Yaputra (121111478)

Kelvin Hermawan (121110782)

Fandy (121111141)

Johny (121112422)

|  |
| --- |
|  |
| OTHELLO  TI A - SORE |
| |  |  |  | | --- | --- | --- | | Christian Yaputra |  |  | |

**Latar Belakang**

Di dalam dunia ini banyak sekali permainan-permainan klasik seperti Minesweeper, Pacman, TicTacToe, Chess, dll. Tetapi semua versi tersebut kebanyakan dimainkan dengan model GUI(Graphical User Interface) yakni sebuah model dengan model grafikal disertai design menawan atau bisa disebut Form Application. Dan untuk membuatnya sangat gampang sekali. Hanya membutuhkan cara drag and drop dikarenakan semua tools GUI sudah tersedia di dalam suatu Tools Programming. Sangat jarang ada yang membuat versi Console dari game tersebut dikarenakan sifatnya yang kaku alias sulit untuk pendesignan Grafik. Padahal jika dilihat, permainan versi Console itu akan menjadi suatu pembelajaran sekaligus suatu gambaran dasar/blueprint untuk membuat suatu permainan jikalau memang permainan tersebut bisa diimplementasikan ke dalam Console. Apalagi kalau di dalam permainan tersebut mengandung suatu algoritma yang bisa diimplementasikan juga untuk Project lainnya seperti algoritma Graph, algoritma AI(Artificial Intelligence). Oleh sebab itu, kelompok kami membuat sebuah permainan klasik yang sudah sangat dikenal dan tidak asing lagi di telinga yaitu Othello. Othello yang kami buat adalah versi Othello dengan Console, jadi diharapkan dengan project kami ini, maka project kami ini bisa dijadikan sebagai bahan referensi untuk pembelajaran permainan dengan teknik-teknik Graph seperti Project kami ini seperti contoh Labyrinth, Maze, Chess, Minesweeper, dll.

**Landasan Teori**

Bahasa pemrograman yang kami pakai adalah Bahasa C#, sebuah bahasa pemrograman dengan platform .NET yang mendukung OOP (Object Oriented Programming), sehingga memudahkan kami untuk menyelesaikan tugas project kami.

Pemrograman berorientasi objek (Object Oriented Programming (OOP)) merupakan paradigma pemrograman yang berorientasikan kepada objek. Semua data dan fungsi di dalam paradigma ini dibungkus dalam kelas-kelas atau objek-objek. Bandingkan dengan logika [pemrograman terstruktur](http://id.wikipedia.org/wiki/Pemrograman_terstruktur). Setiap objek dapat menerima [pesan](http://id.wikipedia.org/wiki/Pesan), memproses data, dan mengirim pesan ke objek lainnya,

Model data berorientasi objek dikatakan dapat memberi fleksibilitas yang lebih, kemudahan mengubah program, dan digunakan luas dalam [teknik piranti lunak](http://id.wikipedia.org/w/index.php?title=Teknik_piranti_lunak&action=edit&redlink=1) skala besar. Lebih jauh lagi, pendukung OOP mengklaim bahwa OOP lebih mudah dipelajari bagi pemula dibanding dengan pendekatan sebelumnya, dan pendekatan OOP lebih mudah dikembangkan dan dirawat.

Ada pun beberapa konsep OOP yang kami gunakan dalam Project kami diantaranya :

1. Class

Kumpulan atas definisi data dan fungsi-fungsi dalam suatu unit untuk suatu tujuan tertentu. Sebagai contoh 'class of dog' adalah suatu unit yang terdiri atas definisi-definisi data dan fungsi-fungsi yang menunjuk pada berbagai macam perilaku/turunan dari anjing. Sebuah class adalah dasar dari modularitas dan struktur dalam pemrograman berorientasi object. Sebuah class secara tipikal sebaiknya dapat dikenali oleh seorang non-programmer sekalipun terkait dengan domain permasalahan yang ada, dan kode yang terdapat dalam sebuah class sebaiknya (relatif) bersifat mandiri dan independen (sebagaimana kode tersebut digunakan jika tidak menggunakan OOP). Dengan modularitas, struktur dari sebuah program akan terkait dengan aspek-aspek dalam masalah yang akan diselesaikan melalui program tersebut. Cara seperti ini akan menyederhanakan pemetaan dari masalah ke sebuah program ataupun sebaliknya.

1. Objek

Object adalah sesuatu yang bisa dianalogikan dengan benda, orang, tempat, kejadian atau konsep-konsep yang ada di dunia nyata yang digunakan pada perangkat lunak atau sistem informasi. Contohnya kampus, gedung, mahasiswa, kuliah, registrasi, pembayaran dan yang lainnya.

1. Abstraksi

Kemampuan sebuah program untuk melewati aspek informasi yang diproses olehnya, yaitu kemampuan untuk memfokus pada inti. Setiap objek dalam sistem melayani sebagai model dari "pelaku" abstrak yang dapat melakukan kerja, laporan dan perubahan keadaannya, dan berkomunikasi dengan objek lainnya dalam sistem, tanpa mengungkapkan bagaimana kelebihan ini diterapkan. Proses, fungsi atau metode dapat juga dibuat abstrak, dan beberapa teknik digunakan untuk mengembangkan sebuah pengabstrakan.

1. Enkapsulasi

Memastikan pengguna sebuah objek tidak dapat mengganti keadaan dalam dari sebuah objek dengan cara yang tidak layak. Hanya metode dalam objek tersebut yang diberi izin untuk mengakses keadaannya. Setiap objek mengakses interface yang menyebutkan bagaimana objek lainnya dapat berinteraksi dengannya. Objek lainnya tidak akan mengetahui dan tergantung kepada representasi dalam objek tersebut.

1. Inheritance

Inheritance adalah konsep pemrograman berorientasi object yang memungkinkan membuat hubungan di mana ada class yang mewarisi semua attribute dan method dari class lain. Class baru yang lebih spesifik disebut subclass atau class turunan, sedangkan class yang diturunkan disebut superclass.

1. Polymorphism

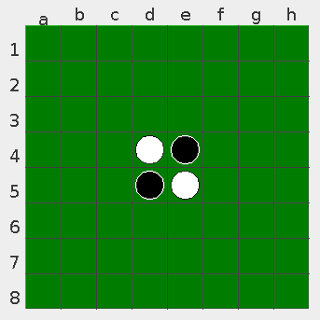
Secara singkat, Polymorphism adalah suatu obyek yang memiliki sifat yang sama dengan **parent**nya atau pada class nya sendiri. Sifat obyeknya ini bisa dalam bentuk yang berbeda-beda. Pada Polymorphism, bisa terjadi banyak perubahan sifat pada subclass nya.

Polymorphism mempunyai 2 bentuk yaitu:  
**a. Override**  
Override merupakan pendefinisian ulang suatu metode oleh subclass. Syarat Override yaitu nama metode, return type, dan parameter harus sama. Jika tidak sama maka bukan dianggap sebagai override tetapi metode yang baru pada subclass.  
  
**b. Overload**  
Overload merupakan pendefinisian ulang suatu metode dalam class yang sama. Syarat overload yaitu nama metode dan tipe parameter harus berbeda dalam class yang sama. Pada class anjing terdapat overload pada metode suara.

Dengan prinsip-prinsip OOP itulah, kemudian kami membuat sebuah tugas Project permainan klasik yaitu Othello. Othello atau disebut juga reversi adalah permainan yang menggunakan papan berisi kotak sebanyak 8x8, antara dua orang pemain dengan keping hitam dan putih seperti di bawah.

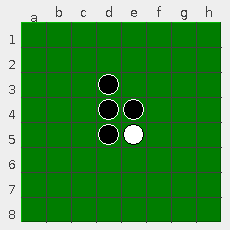
Sejarah Othello berawal tahun 1945, setelah bom atom dijatuhkan di Hiroshima dan Nagasaki. September 1945, Hasegawa Goro yang tengah duduk di kelas satu SMP menerima pelajaran sembari duduk di tanah di bawah langit biru. Mito juga menjadi sasaran pengeboman hingga kastil dan bangunan bersejarah lain ikut habis dalam kobaran api. Dalam suasana seperti itulah permainan ini dilahirkan. Aturan awalnya, bila batu milik pemain pertama diapit oleh batu milik pemain kedua, maka batu pemain pertama menjadi milik pemain kedua. Permainan ini cukup merepotkan karena harus memakai batu dengan 2 warna, hingga akhirnya dipakai kertas yang diwarnai hitam pada salah satu sisi dan putih pada sisi yang lain.

Gambar ilustrasi Othello

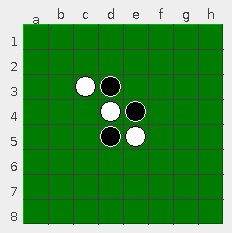


Aturan permainannya adalah sebagai berikut:

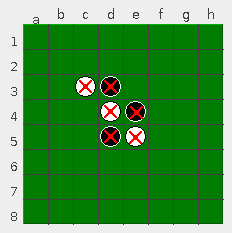
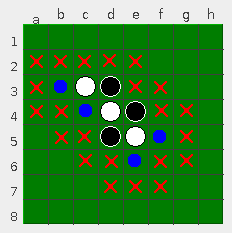
1. Permainan dimulai dari posisi papan dengan susunan keping hitam dan putih seperti pada gambar di atas, yaitu dua keping hitam dan dua keping putih tepat di tengah-tengah papan dengan posisi saling memotong secara diagonal, dengan keping hitam miring ke kanan-atas dan putih miring ke kiri-atas.
2. Pemain hitam melangkah pertama kali dengan meletakkan keping hitam di kotak kosong di mana dia bisa mengapit keping putih di antara dua keping hitam, yaitu di antara keping hitam yang sudah ada di papan dan keping hitam yang baru diletakkan, boleh mengapit secara horizontal, vertikal maupun diagonal, misalnya di kotak "d3".
3. Keping putih yang terjepit pada no. 2 di atas, dibalik semua menjadi keping hitam.

[](http://4.bp.blogspot.com/-TcIz-qWL7t8/TfNtvLbUmTI/AAAAAAAABdw/_1adA1ds6Uc/s1600/first.png)

1. Berikutnya giliran pemain putih melangkah dengan meletakkan keping putih di kotak kosong di mana dia bisa mengapit keping hitam di antara dua keping putih, yaitu di antara keping putih yang sudah ada di papan dan keping putih yang baru diletakkan, boleh mengapit secara horizontal, vertikal maupun diagonal, misalnya di kotak "c3".
2. Keping hitam yang terjepit pada no. 4 di atas, dibalik semua menjadi keping putih.

[](http://1.bp.blogspot.com/-YldvYpo_W2A/TfNv2OGDCCI/AAAAAAAABd4/49IVNgB92kA/s1600/c3.png)

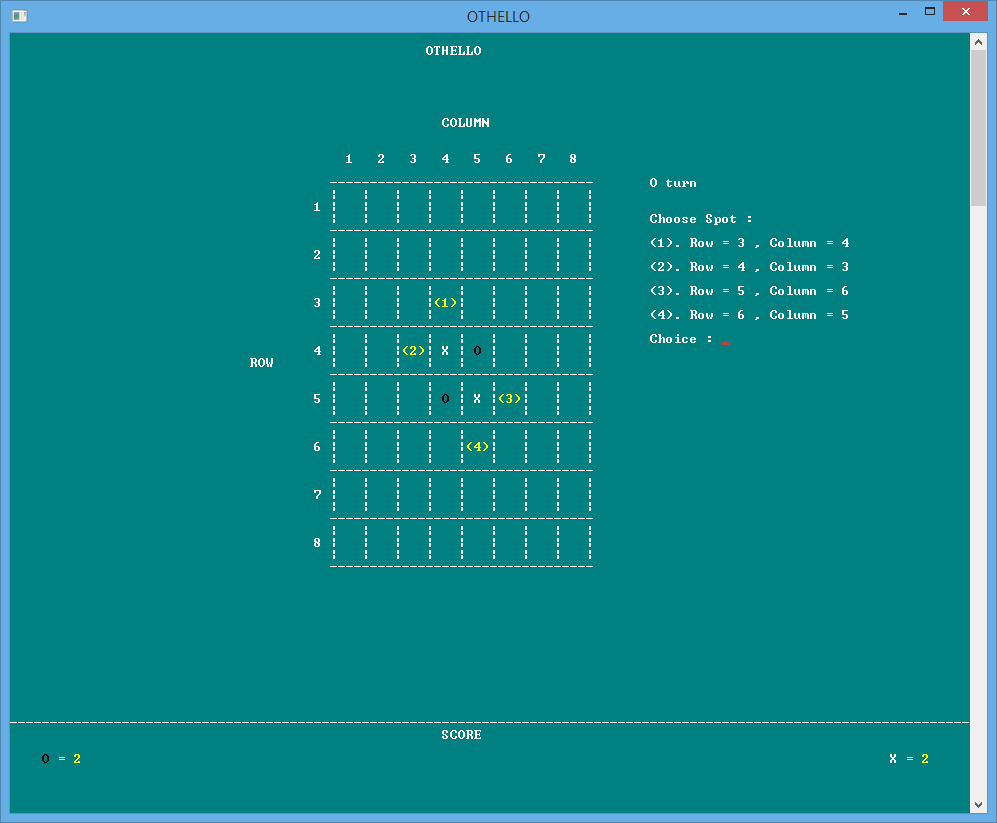
1. Demikian kedua pemain bergantian saling meletakkan kepingnya di kotak kosong di mana dia bisa mengapit keping lawannya, dan membalik keping lawannya menjadi kepingnya.
2. Kedua pemain tidak boleh meletakkan kepingnya di kotak yang sudah terisi, atau di kotak kosong di mana dia tidak mengapit keping lawannya.

[](http://2.bp.blogspot.com/-oY2Jahia45U/TfN1c88-6FI/AAAAAAAABd8/4pjL7swDCoM/s1600/occupied.png)[](http://1.bp.blogspot.com/-WcqX2wG2jF8/TfN1fzEvbSI/AAAAAAAABeA/WMvn4uiHtg8/s1600/invalid.png)

Gambar kiri menunjukkan kotak yang sudah terisi sehingga tidak bisa diletakkan keping baru. Gambar kanan, tanda lingkaran biru menunjukkan kotak kosong yang mengapit keping putih oleh hitam, sehingga hitam boleh melangkah ke sana. Tanda silang merah dan semua kotak kosong diluarnya, menunjukkan kotak kosong yang tidak mengapit keping putih oleh hitam, sehingga hitam tidak boleh melangkah ke sana.

1. Apabila pemain tidak mempunyai kotak di mana dia bisa melangkah, maka dia harus 'pass' yaitu memberikan gilirannya melangkah kepada lawannya.
2. Apabila kedua pemain sama-sama tidak mempunyai kotak di mana dia bisa melangkah, biasanya ketika papan sudah penuh, maka permainan selesai.
3. Pemain dengan jumlah keping terbanyak adalah pemenang.

Ilustrasi diatas adalah ilustrasi dengan menggunakan model GUI. Akan tetapi di dalam Project kami, kami menggunakan model Console seperti pada gambar dibawah ini



Kami merepresentasikan O sebagai piringan hitamnya dan X sebagai piringan putih. Dan kami memberikan list-list query untuk memilih spot bagi Player untuk meletakkan piringan miliknya. Selanjutnya untuk cara bermain tetap sama seperti aturan permainan Othello yang sudah dijelaskan sebelumnya.

Pada project kami ini juga bukan hanya terdapat mode Player vs Player(Multiplayer) saja tetapi terdapat mode AI(Artificial Intelligence) atau bisa disebut Player vs Computer. Kecerdasan Buatan atau Intelegensi Artifisial(Artificial Intelligence atau hanya disingkat AI) didefinisikan sebagai kecerdasan entitas ilmiah. Sistem seperti ini umumnya dianggap komputer. Kecerdasan diciptakan dan dimasukkan ke dalam suatu mesin (komputer) agar dapat melakukan pekerjaan seperti yang dapat dilakukan manusia. Beberapa macam bidang yang menggunakan kecerdasan buatan antara lain sistem pakar, permainan komputer (games), logika fuzzy, jaringan syaraf tiruan dan robotika.

**Pembahasan**

Di dalam pembahasan ini kami akan membahas mengenai 3 hal yakni Algoritma Pencarian Piringan dari Othello itu sendiri, Algoritma Reverse Piringan dan Pemikiran dari AI(Artificial Intelligence).

Pertama yang akan kami bahas adalah Algoritma Pencarian Piringan dari Othello. Kami menemukan banyak sekali cara dalam menggunakan Algoritma pencarian piringan tersebut. Diantaranya Algoritma BFS(Breadth-First Search), DFS(Depth-First Search), Flood Fill, Sequential Search,dll. Diantara banyak Algoritma yang dipakai, Algoritma yang kami gunakan untuk Project kami adalah Algoritma Sequential Search dikarenakan gampang untuk diimplementasikan dan merupakan dasar dari pembuatan permainan ini. Jadi disini kami menggunakan method findleft(kiri), findright(kanan), findup(atas), finddown(bawah), finddleftup(diagonal kiri atas), finddleftdown(diagonal kiri bawah), finddrightup(diagonal kanan atas), finddrightdown(diagonal kanan bawah) untuk mencari piringan tersebut. Adapun juga codingnya adalah sebagai berikut :

public bool findleft(int i,int j,int flag)

{

data dummy;

int tmpj = j;

if (tmpj - 1 < 0)

tmpj = 0;

else

tmpj -= 1;

if (flag == 1 || flag==3)

{

if (row[i][tmpj] is ButtonX)

{

for (int c = j - 2; c >= 0; c--)

{

if (flag == 1)

{

if (row[i][c] is ButtonX) continue;

else if (!(row[i][c] is ButtonO))

break;

else

{

dummy.r = i;

dummy.c = j;

query.Add(dummy);

return true;

}

}

else if (flag == 3)

{

if (row[i][c] is ButtonX) continue;

else if (!(row[i][c] is ButtonO))

break;

else

return true;

}

}

}

}

else if(flag==2 || flag==4)

{

if (row[i][tmpj] is ButtonO)

{

for (int c = j - 2; c >= 0; c--)

{

if (flag == 2)

{

if (row[i][c] is ButtonO) continue;

else if (!(row[i][c] is ButtonX))

break;

else

{

dummy.r = i;

dummy.c = j;

query.Add(dummy);

return true;

}

}

else if (flag == 4)

{

if (row[i][c] is ButtonO) continue;

else if (!(row[i][c] is ButtonX))

break;

else

return true;

}

}

}

}

return false;

}

public bool findright(int i, int j, int flag)

{

data dummy;

int tmpj = j;

if (tmpj + 1 > 7)

tmpj = 7;

else

tmpj += 1;

if (flag == 1 || flag == 3)

{

if (row[i][tmpj] is ButtonX)

{

for (int c = j + 2; c < 8; c++)

{

if (flag == 1)

{

if (row[i][c] is ButtonX) continue;

else if (!(row[i][c] is ButtonO))

break;

else

{

dummy.r = i;

dummy.c = j;

query.Add(dummy);

return true;

}

}

else if (flag == 3)

{

if (row[i][c] is ButtonX) continue;

else if (!(row[i][c] is ButtonO))

break;

else

return true;

}

}

}

}

else if(flag==2 || flag==4)

{

if (row[i][tmpj] is ButtonO)

{

for (int c = j + 2; c < 8; c++)

{

if (flag == 2)

{

if (row[i][c] is ButtonO) continue;

else if (!(row[i][c] is ButtonX))

break;

else

{

dummy.r = i;

dummy.c = j;

query.Add(dummy);

return true;

}

}

else if (flag == 4)

{

if (row[i][c] is ButtonO) continue;

else if (!(row[i][c] is ButtonX))

break;

else

return true;

}

}

}

}

return false;

}

public bool findup(int i, int j, int flag)

{

data dummy;

int tmpi = i;

if (tmpi -1 < 0)

tmpi = 0;

else

tmpi -= 1;

if (flag == 1 || flag == 3)

{

if (row[tmpi][j] is ButtonX)

{

for (int r = i - 2; r >= 0; r--)

{

if (flag == 1)

{

if (row[r][j] is ButtonX) continue;

else if (!(row[r][j] is ButtonO))

break;

else

{

dummy.r = i;

dummy.c = j;

query.Add(dummy);

return true;

}

}

else if (flag == 3)

{

if (row[r][j] is ButtonX) continue;

else if (!(row[r][j] is ButtonO))

break;

else

return true;

}

}

}

}

else

{

if (row[tmpi][j] is ButtonO)

{

for (int r = i - 2; r >= 0; r--)

{

if (flag == 2)

{

if (row[r][j] is ButtonO) continue;

else if (!(row[r][j] is ButtonX))

break;

else

{

dummy.r = i;

dummy.c = j;

query.Add(dummy);

return true;

}

}

else if (flag == 4)

{

if (row[r][j] is ButtonO) continue;

else if (!(row[r][j] is ButtonX))

break;

else

return true;

}

}

}

}

return false;

}

public bool finddown(int i, int j, int flag)

{

data dummy;

int tmpi = i;

if (tmpi + 1 > 7)

tmpi = 7;

else

tmpi += 1;

if (flag == 1 || flag == 3)

{

if (row[tmpi][j] is ButtonX)

{

for (int r = i + 2; r < 8; r++)

{

if (flag == 1)

{

if (row[r][j] is ButtonX) continue;

else if (!(row[r][j] is ButtonO))

break;

else

{

dummy.r = i;

dummy.c = j;

query.Add(dummy);

return true;

}

}

else if (flag == 3)

{

if (row[r][j] is ButtonX) continue;

else if (!(row[r][j] is ButtonO))

break;

else

return true;

}

}

}

}

else

{

if (row[tmpi][j] is ButtonO)

{

for (int r = i + 2; r < 8; r++)

{

if (flag == 2)

{

if (row[r][j] is ButtonO) continue;

else if (!(row[r][j] is ButtonX))

break;

else

{

dummy.r = i;

dummy.c = j;

query.Add(dummy);

return true;

}

}

else if (flag == 4)

{

if (row[r][j] is ButtonO) continue;

else if (!(row[r][j] is ButtonX))

break;

else

return true;

}

}

}

}

return false;

}

public bool findDleftup(int i, int j, int flag)

{

data dummy;

int tmpi = i, tmpj=j;

if (tmpj - 1 < 0 || tmpi - 1 < 0)

{

tmpj = 0;

tmpi = i;

}

else

{

tmpj -= 1;

tmpi -= 1;

}

int k = tmpj;

if (flag == 1 || flag == 3)

{

if (row[tmpi][tmpj] is ButtonX)

{

for (int r = tmpi - 1; r >= 0; r--)

{

k--;

if (k < 0)

break;

if (flag == 1)

{

if (row[r][k] is ButtonX) continue;

else if (!(row[r][k] is ButtonO))

break;

else

{

dummy.r = i;

dummy.c = j;

query.Add(dummy);

return true;

}

}

else if (flag == 3)

{

if (row[r][k] is ButtonX) continue;

else if (!(row[r][k] is ButtonO))

break;

else

return true;

}

}

}

}

else

{

if (row[tmpi][tmpj] is ButtonO)

{

for (int r = tmpi-1; r >=0; r--)

{

k--;

if (k < 0)

break;

if (flag == 2)

{

if (row[r][k] is ButtonO) continue;

else if (!(row[r][k] is ButtonX))

break;

else

{

dummy.r = i;

dummy.c = j;

query.Add(dummy);

return true;

}

}

else if (flag == 4)

{

if (row[r][k] is ButtonO) continue;

else if (!(row[r][k] is ButtonX))

break;

else

return true;

}

}

}

}

return false;

}

public bool findDleftdown(int i, int j, int flag)

{

data dummy;

int tmpi = i,tmpj=j;

if (tmpj - 1 < 0 || tmpi + 1 > 7)

{

tmpj = 0;

tmpi = i;

}

else

{

tmpj -= 1;

tmpi += 1;

}

int k = tmpj;

if (flag == 1 || flag == 3)

{

if (row[tmpi][tmpj] is ButtonX)

{

for (int r = tmpi + 1; r < 8; r++)

{

k--;

if (k < 0) break;

if (flag == 1)

{

if (row[r][k] is ButtonX) continue;

else if (!(row[r][k] is ButtonO))

break;

else

{

dummy.r = i;

dummy.c = j;

query.Add(dummy);

return true;

}

}

else if (flag == 3)

{

if (row[r][k] is ButtonX) continue;

else if (!(row[r][k] is ButtonO))

break;

else

return true;

}

}

}

}

else

{

if (row[tmpi][tmpj] is ButtonO)

{

for (int r = tmpi + 1; r < 8; r++)

{

k--;

if (k < 0) break;

if (flag == 2)

{

if (row[r][k] is ButtonO) continue;

else if (!(row[r][k] is ButtonX))

break;

else

{

dummy.r = i;

dummy.c = j;

query.Add(dummy);

return true;

}

}

else if (flag == 4)

{

if (row[r][k] is ButtonO) continue;

else if (!(row[r][k] is ButtonX))

break;

else

return true;

}

}

}

}

return false;

}

public bool findDrightup(int i, int j, int flag)

{

data dummy;

int tmpi = i, tmpj = j;

if (tmpi - 1 < 0 || tmpj + 1 > 7)

{

tmpi = 0;

tmpj = j;

}

else

{

tmpi -= 1;

tmpj += 1;

}

int k = tmpi;

if (flag == 1 || flag == 3)

{

if (row[tmpi][tmpj] is ButtonX)

{

for (int c = tmpi + 1; c < 8; c++)

{

k--;

if (k < 0) break;

if (flag == 1)

{

if (row[k][c] is ButtonX) continue;

else if (!(row[k][c] is ButtonO))

break;

else

{

dummy.r = i;

dummy.c = j;

query.Add(dummy);

return true;

}

}

else if (flag == 3)

{

if (row[k][c] is ButtonX) continue;

else if (!(row[k][c] is ButtonO))

break;

else

return true;

}

}

}

}

else

{

if (row[tmpi][tmpj] is ButtonO)

{

for (int c = tmpi + 1; c < 8; c++)

{

k--;

if (k < 0) break;

if (flag == 2)

{

if (row[k][c] is ButtonO) continue;

else if (!(row[k][c] is ButtonX))

break;

else

{

dummy.r = i;

dummy.c = j;

query.Add(dummy);

return true;

}

}

else if (flag == 4)

{

if (row[k][c] is ButtonO) continue;

else if (!(row[k][c] is ButtonX))

break;

else

return true;

}

}

}

}

return false;

}

public bool findDrightdown(int i, int j, int flag)

{

data dummy;

int tmpi = i, tmpj = j;

if (tmpj + 1 > 7 || tmpi + 1 > 7)

{

tmpj = j;

tmpi = i;

}

else

{

tmpj += 1;

tmpi += 1;

}

int k = tmpj;

if (flag == 1 || flag == 3)

{

if (row[tmpi][tmpj] is ButtonX)

{

for (int r = tmpi + 1; r < 8; r++)

{

k++;

if (k > 7)

break;

if (flag == 1)

{

if (row[r][k] is ButtonX) continue;

else if (!(row[r][k] is ButtonO))

break;

else

{

dummy.r = i;

dummy.c = j;

query.Add(dummy);

return true;

}

}

else if (flag == 3)

{

if (row[r][k] is ButtonX) continue;

else if (!(row[r][k] is ButtonO))

break;

else

return true;

}

}

}

}

else

{

if (row[tmpi][tmpj] is ButtonO)

{

for (int r = tmpi + 1; r < 8; r++)

{

k++;

if (k > 7)

break;

if (flag == 2)

{

if (row[r][k] is ButtonO) continue;

else if (!(row[r][k] is ButtonX))

break;

else

{

dummy.r = i;

dummy.c = j;

query.Add(dummy);

return true;

}

}

else if (flag == 4)

{

if (row[r][k] is ButtonO) continue;

else if (!(row[r][k] is ButtonX))

break;

else

return true;

}

}

}

}

return false;  
 }

Jadi disini kami mengibaratkan flag sebagai set pilihan dari piringan. Jikalau flag nya 1 atau 3 maka itu diperuntukkan untuk pencarian piringan X(piringan putih) sampai ketemu piringan O(hitam), jikalau flag nya 2 atau 4 maka diperuntukkan untuk pencarian piringan O(piringan hitam) sampai ketemu piringan X(putih). Untuk masing-masing pemilihan juga terdapat pembagian tugas. Jikalau flag adalah 1 dan 3 maka diperuntukkan untuk pengisian query dimana query tersebut adalah query list dari pilihan untuk User itu sendiri, Query itu kami buat dalam model List. Query itu akan terisi apabila piringan yang diletakkan di koordinat tersebut menemukan deretan piringan yang berlawanan sampai adanya piringan yang sama. Jikalau tidak ada, Query tidak akan terisi. Lalu jika flag adalah 2 dan 4 maka diperuntukkan hanya untuk mengecek suatu peletakan pada koordinat tertentu, apakah terdapat piringan yang berlawanan sampai ada piringan yang sama atau tidak. Jika iya maka kembalikan nilai true, jika tidak false.

Kedua yang akan kami bahas adalah Algoritma dari Mereversekan atau membalikkan piringan yang berlawanan. Jadi disini kami hanya menjelaskan pembalikan piringan putih menjadi hitam(X menjadi O). Adapun codingnya adalah sebagai berikut :

public void ReverseXtoO(int key)

{

//left

if(findleft(query[key-1].r,query[key-1].c,3))

{

row[query[key - 1].r][query[key - 1].c] = new ButtonO(query[key - 1].r, query[key - 1].c);

for(int c=query[key-1].c-1;c>=0;c--)

{

if (row[query[key - 1].r][c] is ButtonX)

row[query[key - 1].r][c] = new ButtonO(query[key - 1].r, c);

else if (row[query[key-1].r][c] is ButtonO)

break;

}

}

//right

if (findright(query[key - 1].r, query[key - 1].c, 3))

{

row[query[key - 1].r][query[key - 1].c] = new ButtonO(query[key - 1].r, query[key - 1].c);

for (int c = query[key - 1].c + 1; c < 8; c++)

{

if (row[query[key - 1].r][c] is ButtonX)

row[query[key - 1].r][c] = new ButtonO(query[key - 1].r, c);

else if (row[query[key-1].r][c] is ButtonO)

break;

}

}

//up

if (findup(query[key - 1].r, query[key - 1].c, 3))

{

row[query[key - 1].r][query[key - 1].c] = new ButtonO(query[key - 1].r, query[key - 1].c);

for (int r = query[key - 1].r - 1; r >= 0; r--)

{

if (row[r][query[key-1].c] is ButtonX)

row[r][query[key-1].c] = new ButtonO(r, query[key-1].c);

else if (row[r][query[key-1].c] is ButtonO)

break;

}

}

//down

if (finddown(query[key - 1].r, query[key - 1].c, 3))

{

row[query[key - 1].r][query[key - 1].c] = new ButtonO(query[key - 1].r, query[key - 1].c);

for (int r = query[key - 1].r + 1; r < 8; r++)

{

if (row[r][query[key - 1].c] is ButtonX)

row[r][query[key - 1].c] = new ButtonO(r, query[key - 1].c);

else if (row[r][query[key - 1].c] is ButtonO)

break;

}

}

//dleftup

if (findDleftup(query[key - 1].r, query[key - 1].c, 3))

{

int j = query[key - 1].c;

row[query[key - 1].r][query[key - 1].c] = new ButtonO(query[key - 1].r, query[key - 1].c);

for (int r = query[key - 1].r - 1; r >= 0; r--)

{

j--;

if(j<0) break;

if (row[r][j] is ButtonX)

row[r][j] = new ButtonO(r, j);

else if (row[r][j] is ButtonO)

break;

}

}

//dleftdown

if (findDleftdown(query[key - 1].r, query[key - 1].c, 3))

{

int j = query[key - 1].c;

row[query[key - 1].r][query[key - 1].c] = new ButtonO(query[key - 1].r, query[key - 1].c);

for (int r = query[key - 1].r + 1; r < 8; r++)

{

j--;

if (j < 0) break;

if (row[r][j] is ButtonX)

row[r][j] = new ButtonO(r, j);

else if (row[r][j] is ButtonO)

break;

}

}

//drightup

if (findDrightup(query[key - 1].r, query[key - 1].c, 3))

{

int j = query[key - 1].r;

row[query[key - 1].r][query[key - 1].c] = new ButtonX(query[key - 1].r, query[key - 1].c);

for (int c = query[key - 1].c + 1; c < 8; c++)

{

j--;

if (j - 1 < 0) break;

if (row[j][c] is ButtonX)

row[j][c] = new ButtonO(j, c);

else if (row[j][c] is ButtonO)

break;

}

}

//drightdown

if (findDrightdown(query[key - 1].r, query[key - 1].c, 3))

{

int j = query[key - 1].c;

row[query[key - 1].r][query[key - 1].c] = new ButtonO(query[key - 1].r, query[key - 1].c);

for (int r = query[key - 1].r + 1; r < 8; r++)

{

j++;

if (j + 1 > 7) break;

if (row[r][j] is ButtonX)

row[r][j] = new ButtonO(r, j);

else if (row[r][j] is ButtonO)

break;

}

}

}

Jadi disini, algoritma Search pada penjelasan sebelumnya sudah kami set agar menghasilkan sebuah output bertipe boolean(True or False). Sehingga disini ada variabel key yang merupakan variabel yang berisi nilai pilihan List Query yang User input/masukkan dan List tersebut akan ditampilkan dilayar. Setiap List Query itu memiliki koordinat r(baris) dan c(kolom). Jadi diperiksa apakah koordinat dari query yang sudah dipilih apakah ditemukan piringan yang berlawanan dari 8 sisi tersebut(kiri,kanan,atas,bawah,diagonal kiri atas, diagonal kiri bawah, diagonal kanan atas dan diagonal kanan bawah) dengan flag = 2(dikarenakan ini untuk kondisi pengecekan adanya piringan putih sampai ketemu piringan hitam). Selanjutnya jika ketemu, maka lakukan pembalikan piringan tersebut dengan algoritma looping sederhana dengan peletakan di query koordinat itu adalah O dan sisanya looping pembalikan piringan X menjadi O sampai ketemu piringan O maka looping berhenti. Jadi sisanya untuk Algoritma Reverse piringan Hitam jadi Putih hanya mengubah flag menjadi 4 dan membalikkan Pemeriksaan atau Pengecekan objek piringan, yang mengecek objek X menjadi O dan yang mengecek objek O menjadi X.

Ketiga untuk Model AI nya tersendiri, kami juga menemukan banyak sekali algoritma AI yang diimplementasikan untuk berbagai permainan seperti contoh : Algoritma MinMax. Tetapi kami menemukan suatu cara yang lebih sederhana untuk membuat suatu AI dalam permainan yakni dengan menggunakan objek Random yang sudah terdapat di C#. Adapun codingnya adalah sebagai berikut :

public void Xturn()

{

**Random a = new Random();**

int key;

while (true)

{

Console.Clear();

Console.SetCursorPosition(Console.WindowWidth / 2 + 20, Console.WindowHeight / 2 - 20);

Console.WriteLine("X turn");

Console.SetCursorPosition(Console.WindowWidth / 2 + 20, Console.WindowHeight / 2 - 17);

fillQueryX();

PrintBoard();

if (hasQueryX)

{

Console.SetCursorPosition(Console.WindowWidth / 2 + 20, tmpy + 2);

**key = a.Next(1, idx);**

Thread.Sleep(1500);

Console.Write("Computer Choose List {0}",key);

Console.ReadKey();

ReverseOtoX(key);

query.Clear();

break;

}

else

break;

query.Clear();

}

}

Jadi disini terlihat jelas pada 2 baris coding yang sudah kami tandai bahwa kami menggunakan konsep random untuk pemilihan List Query pilihan Spot yang sebenarnya kami modifikasikan sedikit dari model Player vs Player. Variabel key akan menyimpan angka random yang diproses dari 1 sampai idx. Variabel idx disini didapat dengan cara membuatnya menjadi variabel public dan nilainya berubah pada saat pencarian Algoritma search yang sudah kami singgung di awal pembahasan. Jikalau terdapat Query maka nilainya bertambah 1 dan seterusnya. Atau bisa saja didapat dengan cara instan yaitu dengan menghitung ada berapa Query List yang sudah tersimpan di dalam List dengan menggunakan method Count yang sudah ada dalam Objek C# Visual .NET.

**Kesimpulan**

Kesimpulan yang dapat kami tarik dari Project kami ini adalah :

1. Algoritma sederhana seperti Sequential Search bisa diimplementasikan dalam permainan Othello.
2. Objek Random juga bisa dipakai untuk pengimplementasian model AI(Artificial Intelligence)
3. Dalam membuat project Console, kesulitan hanya terdapat pada penentuan di koordinat mana suatu teks mau ditulis. Selebihnya untuk pengimplementasian Algoritma, baik Console dan Form sama-sama bisa menggunakan Algoritma yang sama.