KECERDASAN BUATAN UNTUK BLOKUS CLASSIC MENGGUNAKAN HEURISTIK, FLOODFILL, DAN ALGORITMA GREEDY

Hong Liang Cai[1)], Sebastian Aldi[2)], Winston Renatan[3*)]

[1]Teknik Informatika, Fakultas Ilmu Komputer, Universitas Pelita Harapan
Email: liangcai.stdnt@gmail.com[1)]
[2]Teknik Informatika, Fakultas Ilmu Komputer, Universitas Pelita Harapan
Email: sebastian.aldi17@gmail.com[2)]
[3]Teknik Informatika, Fakultas Ilmu Komputer, Universitas Pelita Harapan
Email: winstonrenatan@gmail.com[3)]

**Penulis Korespondensi*

Abstract – Blokus is an abstract strategy game which has complex variable to determine ones move. We propose an idea for Artificial Intelligence in Blokus using Heuristics, FloodFill, and Greedy algorithm, which will be called as LeakyAI. Both the game and its implementation are established in Java language. To test the result, we performed benchmarking towards Brute Force AI, itself, and the developers.

***Keywords**: Blokus, Artificial Intelligence, Game Theory, Heuristic Search, FloodFill Algorithm, Greedy*

Abstrak – Blokus merupakan sebuah permainan strategi abstrak yang memiliki variabel kompleks untuk menentukan pergerakan pemain. Kami mengusulkan ide Kecerdasan Buatan pada Blokus dengan Algoritma Heuristik, FloodFill, dan Greedy, yang akan disebut LeakyAI. Baik permainan maupun implementasinya dibuat menggunakan bahasa Java. Untuk menguji hasilnya, kami melakukan pembandingan terhadap Kecerdasan Buatan Brute Force, dirinya sendiri, dan dengan para pengembang.

***Kata Kunci**: Blokus, Kecerdasan Buatan, Teori Permainan, Pencarian Heuristik, Algoritma FloodFill, Greedy*

PENDAHULUAN

Blokus adalah permainan strategi abstrak yang dirancang oleh Bernard Tavitian dan diterbitkan pada tahun 2000. Permainan ini melibatkan empat pemain dengan warna berbeda, masing-masing dengan 21 potongan permainan berbentuk seperti tetris untuk diletakkan di papan persegi dengan ukuran 20x20 unit. Tujuan permainan ini adalah untuk menempatkan semua potongan di papan, permainan dimulai di keempat sudut papan dan setiap bagian harus menyentuh hanya bagian sudut dari potongan warna yang sama (tanpa menyentuh potongan milik pemain sendiri di bagian samping). Poin akan dihitung dari persegi unit pemain yang tersisa (1 unit persegi = -1 poin), +15 poin akan diberikan jika pemain dapat menempatkan semua 21

Diterima <23022021>, Revisi <11072021>, Diterima untuk publikasi <26072021>.

ISSN : 2528-5114

buah, dan tambahan +5 poin jika bagian terakhir ditempatkan adalah persegi satu unit [1].

Kami mendesain ulang permainan menggunakan bahasa Java karena cukup berguna ketika berhadapan dengan pemrograman berorientasi objek, dimana desain permainan akan diuraikan nanti. Membangun permainan dan Kecerdasan Buatan untuk permainan papan ini cukup menantang dalam prosesnya. Di sini Kecerdasan Buatan berfungsi untuk memberikan cara untuk mencapai tujuan pada lingkungan yang kompleks [2], banyak faktor berkontribusi pada satu keputusan seperti potongan yang dimiliki, potongan pemain lain, kerja tim dari pemain lain, dan lainnya. Beberapa pendekatan telah digunakan di Blokus Duo (jenis Blokus lain dengan hanya dua pemain dan papan persegi 14x14 unit) [3] seperti MiniMax dengan Alpha-Beta Pruning [4] dan Monte Carlo Tree Search [5]. Ide yang kami usulkan di sini adalah menggunakan Heuristik, FloodFill, dan Algorithma Greedy untuk mengatasi masalah ini.

DESAIN PERMAINAN

Blokus memiliki tiga bagian utama dalam permainannya, yang secara umum terdiri dari papan, pemain, dan potongan-potongannya. Untuk mensimulasikan permainan papan nyata secara digital, kita perlu memahami dan mengerti setiap komponen permainan dengan cara yang lebih rinci termasuk bagaimana ia dapat bergerak dan keadaan apa yang membuat permainan berakhir.

Papan dan Pemain

Papan terdiri dari persegi berukuran 20x20 unit, di mana masing-masing persegi unit memiliki nomor dan warna mereka sendiri yang menunjukkan pemilik persegi tersebut. Jumlah dan warna persegi akan berubah setelah pemain meletakkan potongannya di persegi berwarna putih atau ruang kosong di papan. Persegi berwarna tidak dapat diubah dalam kondisi apa pun (misalnya menempatkan di atas bagian lain, di luar batas, dan lainnya). Informasi terperinci diberikan dalam tabel berikut.

Tabel 1 Sistem Angka dan Pewarnaan

Warna atau Pemain	Angka
Putih	-1
Merah	0
Hijau	1
Biru	2
Kuning	3

Warna putih berarti saat ini persegi kosong (tidak dimiliki oleh pemain mana pun), sementara warna lainnya menunjukkan persegi tersebut dimiliki oleh pemain dengan warna yang sesuai. Warna dan penomoran tersebut diperlukan untuk mengetahui langkah mana yang mungkin dapat dilakukan oleh pemain dalam persegi tertentu. Pemain kemudian akan diinisialisasi di kelas Blokus menggunakan kode di bawah ini. LeakyAI atau Player dapat diubah ke AI lain yang akan ditempatkan di papan.

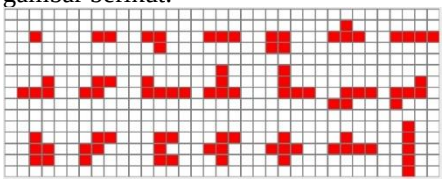
```
1   public static void main(String args[]) {
2     players[0] = new Player(0);
3     players[1] = new LeakyAI(1);
4     players[2] = new LeakyAI(2);
5     players[3] = new LeakyAI(3);
6     mainGui = new Gui(players, board);
7     mainGui.renderBoard();
8     setTurn(0);
9   }
```

Gambar 1 Kode Menginisialisasi Pemain di Papan

Potongan

Ada 21 buah potongan dalam permainan untuk setiap pemain yang didistribusikan dari 1 unit persegi hingga 5 unit persegi. Semua 21 potongan ini akan disimpan dalam bentuk berbagai matriks. Selain itu, potongan-potongan itu tidak statis, tetapi itu adalah bagian dinamis yang dapat diputar dan dicerminkan (*mirror*) oleh pemain untuk mencocokkan penggunaannya di papan permainan.

Semua potongan ditampilkan dalam gambar berikut.



Gambar 2 Semua Potongan di Blokus

Setiap bagian diwakili dengan cara menggunakan penomoran koordinat kartesius, mulai dari paling atas kiri. Sebuah potongan akan memiliki matriks[x][y] dan menyimpan nilai baik 1 atau 2. Masing-masing nilai memiliki interpretasinya sendiri, 1 berarti bahwa potongan hadir pada koordinat [x][y], sedangkan 2 berarti potongan hadir dan merupakan titik tengah potongan.

Tabel 2 Representasi Potongan

Potongan	Representasi
	matriks[0][0] = 1 matriks[0][1] = 2 matriks[0][2] = 1
	matriks[0][1] = 1 matriks[0][2] = 1 matriks[1][0] = 1 matriks[1][1] = 2
	matriks[0][1] = 1 matriks[0][2] = 1 matriks[1][0] = 1 matriks[1][1] = 2 matriks[2][1] = 1

Ketika ada rotasi, pencerminan secara horizontal atau vertikal yang dilakukan pada bagian tersebut maka harus ada penyesuaian yang harus dilakukan. Di pencerminan horizontal, kolom pertama akan menjadi kolom kelima dan kolom kedua akan menjadi kolom keempat, sedangkan kolom ketiga tetap sama. Pencerminan vertikal identik dengan yang horizontal, tetapi dilakukan pada baris. Rotasi agak berbeda di mana potongan pertama akan diubah urutannya dan hasilnya akan dicerminkan horizontal, yang akan menjadi produk akhir.

```
1   public Integer[][] rotate() {
2     Integer newMatrix[][] = new Integer[5][5];
3     for (int i = 0; i < 5; i++) {
4       for (int j = 0; j < 5; j++) {
5         newMatrix[i][j] = matrix[j][i];
6       }
7     }
8     newMatrix = horizontalMirror(newMatrix);
9     matrix = newMatrix;
10    return newMatrix;
11  }
```

Gambar 3 Kode Fungsi Cermin Horizontal

```
1   public Integer[][] horizontalMirror(Integer[][] newMatrix) {
2     for (int i=0; i<5; i++) {
3       for (int j=0; j<3; j++) {
4         int temp = newMatrix[i][4-j];
5         newMatrix[i][4-j] = newMatrix[i][j];
6         newMatrix[i][j] = temp;
7       }
8     }
9     matrix = newMatrix;
10    return newMatrix;
11  }
```

Gambar 4 Kode Fungsi Cermin Vertikal

```
1   public Integer[][] verticalMirror(Integer[][] newMatrix) {
2     for (int i = 0; i < 5; i++) {
3       for (int j = 0; j < 3; j++) {
4         int temp = newMatrix[4-i][ j];
5         newMatrix[4-i][j] = newMatrix[i][j];
6         newMatrix[i][j] = temp;
7       }
8     }
9     matrix = newMatrix;
10    return newMatrix;
11  }
```

Gambar 5 Kode Fungsi untuk Memutar

Status dan Gerakan Permainan

Pemain akan mengambil giliran untuk bergerak persis dalam urutan merah, hijau, biru, dan kuning kemudian kembali lagi ke pemain merah. Untuk memeriksa apakah pemain memiliki opsi untuk bergerak dalam persegi tertentu, teknik *Brute Force* akan dilakukan melalui pengecekan dan verifikasi potongan-potongan yang tersisa pada pemain dan potongan tersebut kemudian akan dicoba untuk diputar sebanyak empat kali, kemudian dicerminkan dan diputar lagi beberapa kali. Jika salah satu proses benar (bagian cocok di papan), maka fungsi akan mengembalikan bahwa pemain memiliki kemungkinan bergerak dan permainan belum berakhir dan dapat dilanjutkan. Praktik ini akan dilakukan untuk semua potongan yang tersisa dari para pemain, jika tidak ada potongan

yang cocok di papan maka pemain kehilangan giliran mereka.

```
1   IF piece is fully used → RETURN false
2   FOR i in board side:
3     FOR j in board side:
4       FOR ii in players piece:
5         IF piece is used → pass check
6         FOR 4:
7           rotate piece
8           IF valid move → RETURN true
9         horizontal mirror piece
10        FOR 4:
11          rotate piece
12          IF valid move → RETURN true
13  RETURN false
```

Gambar 6. Fungsi Algoritma hasMove

Sementara itu, fungsi *hasMove()* juga memanggil dan menggunakan *isValidMove()* untuk memeriksa apakah penempatan sebuah potongan tertentu dimungkinkan untuk dilakukan atau tidak. Kualifikasinya seperti: di luar batas atau papan, menumpuk kemungkinan pada potongan-potongan lain, giliran pertama pemain (harus berada di sudut papan), sisi potongan bertabrakan, dan tidak menempatkan potongan dengan secara diagonal.

N.B. "warning" pada Gambar 7, adalah panel yang menunjukkan bahwa langkah tertentu tidak dapat dilakukan dan bahwa pemain harus membuat langkah lain.

```
1   IF out of bounds → warning
2   IF stack other piece on placement → warning
3   IF player is on first turn:
4     IF place not in corner of board → warning
5   ELSE:
6     check [x-1][y-1], [x][y-1], [x+1][y-1], ..., [x+1][y+1]
7     IF there exist block in [x-1][y-1], [x+1][y-1], [x-1][y+1],
      [x+1][y+1] → warning touch diagonal
8     IF there exist block in [x][y-1], [x-1][y], [x+1][y], [x][y+1]
      → warning touch side
9   IF no warning → move is valid
```

Gambar 7. Fungsi Algoritma isValidMove

Akhir Permainan

Permainan berakhir baik saat tidak ada lagi gerakan yang tersisa untuk setiap pemain atau tidak ada lagi potongan yang tersisa untuk ditempatkan di papan permainan. Beberapa skenario akhir dari permainan seperti: semua pemain menempatkan semua potongan mereka, beberapa pemain meletakkan semua potongan mereka dan beberapa lainnya tidak memiliki langkah lagi, dan tidak ada lagi gerakan untuk setiap pemain. Sistem akan memeriksa bagian yang tersisa yang dilakukan setiap pemain dan langkah apa yang diambil pemain terakhir. Akan ada sebuah tampilan yang menampilkan peringkat setiap pemain dengan poin mereka di akhir permainan.

Tabel 3 Sistem Penilaian

Kondisi	Poin
Untuk setiap potongan yang tersisa per satu unit persegi	-1
Semua potongan ditempatkan di atas papan	+15
Potongan terakhir yang ditempatkan adalah persegi satu unit	+5

STRATEGI PERMAINAN

Seperti yang disebutkan sebelumnya, kami menerapkan Heuristik, FloodFill, dan Algoritma Greedy dalam merancang Kecerdasan Buatan untuk permainan Blokus yang akan lebih diuraikan lebih lanjut pada bagian ini.

Fungsi Potongan

Ketika seorang pemain memulai langkah mereka, hanya akan ada sepotong yang akan digunakan pada giliran itu, tentu saja kita ingin memiliki potongan terbaik untuk ditempatkan di papan untuk menang. Umumnya, potongan terbaik untuk beberapa putaran pertama adalah menggunakan potongan-potongan yang terbesar. Ada beberapa fungsi yang dibuat untuk mengetahui mana potongan yang tersisa dari pemain adalah yang terbaik untuk ditempatkan. Kami akan memanggil fungsi eksternal dari kelas lain yang berfungsi untuk memeriksa apakah langkah tertentu valid atau tidak, bersama dengan melakukan iterasi dari bagian terbesar ke terkecil. Yang pertama adalah iterasi dan memeriksa potongan mana saja yang masih tersisa adalah dengan getLargestAvailablePiece().

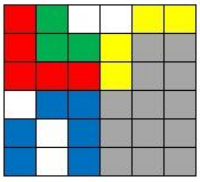
Sementara itu, kami ingin juga mendapatkan nilainya dalam satuan unit persegi yang akan menggunakan fungsi *getPieceSize()*, algoritma akan iterasi

melalui blok potongan 5x5 dan memeriksa blok mana yang diisi dan mencatatnya.

Berikutnya adalah largestMoveAt() yang akan berfungsi untuk mengetahui dan menghitung potongan terbesar untuk ditempatkan dalam koordinat (misalnya dalam koordinat x, y bagian yang dapat ditempatkan adalah 5 unit persegi) fungsi ini akan mengembalikan bilangan bulat. Kemudian ada juga largestPieceAt() yang dapat dikatakan sangat mirip dengan largestMoveAt() tetapi ini akan mengembalikan potongan yang mana bukan nilai bilangan bulat. Kemudian, akhirnya kita tentu saja perlu mengetahui indeks potongan mana yang harus digunakan menggunakan fungsi largestPieceIndexAt(). Terdapat juga suatu kebutuhan untuk menciptakan kembali atau mengkloning papan karena kami ingin melakukan perhitungan FloodFill di papan kloning untuk setiap pemain alih-alih di papan utama di mana ia dapat diakses oleh semua pemain lain.

Algoritma FloodFill

Seperti apa yang telah disebutkan sebelum pada pendahuluan, hal pertama yang perlu kita miliki untuk pengambilan keputusan adalah algoritma FloodFill, yang kemudian akan banyak membantu. FloodFill akan membantu Kecerdasan Buatan yang diterapkan untuk menentukan apakah layak atau tidak untuk menempatkan sepotong ke ruang kosong dalam mendapatkan poin dengan jumlah tertentu. Representasi angka tentang cara kerja FloodFill dapat dilihat di bawah ini, pada Gambar 8.



Gambar 8 Sistem Keputusan FloodFill

Algoritma FloodFill akan diimplementasikan dengan algoritma lain yang dijelaskan kemudian. Dari gambar tersebut, misalnya Kecerdasan Buatan bermain sebagai Potongan Merah, FloodFill kemudian akan menyelidiki kotak abu-abu (ruang kosong) dan melihat nilainya apakah layak untuk meletakkan potongan di sana atau di tempat lain. Jumlah ruang kosong yang tersedia bersama dengan nilai heuristik internal lain yang dibahas nantinya akan menentukan nilai sebuah langkah. Algoritma FloodFill dapat dilihat pada Gambar 9.

```
1   FloodFill (board, x, y, points) {
2     IF unit square value is -1:
3       SET unit square value = -999
4       ADD points
5       check four sides → CALL FloodFill
6       RETURN points
7     ELSE:
8       RETURN points
9   }
```

Gambar 9 Algoritma FloodFill

Heuristik

Selain itu, kami juga datang dengan nilai heuristik untuk melengkapi algoritma yang ada. Pada awalnya kami memiliki pemikiran dengan dua jenis heuristik di papan, angka pada setiap kotak unit mewakili nilainya. Rencana awal yang kami adalah dengan tetap fokus untuk mendapatkan semua potongan ke tengah papan dan kemudian didistribusikan secara merata di antara papan permainan. Dengan membidik pusat (bagian tengah) papan sesegera mungkin, harapannya adalah akan ada lebih banyak kemungkinan bergerak dibandingkan dengan penempatan yang tetap berada di sudut awal. Tapi, model ini tidak bekerja dengan cara yang cukup memuaskan. Alasan di baliknya adalah karena ada kesempatan bagi seseorang untuk memenuhi papan ujung pada bagian mereka memulai permainan, alih-alih maju dan menempati area papan pemain lain.

1	1	1	1	1	1	1	1	1	1	10	10	10	10	10	10	10	10	10	10
1	2	2	2	2	2	2	2	2	2	10	10	10	10	10	10	10	10	10	10
1	2	3	3	3	3	3	3	3	3	10	10	10	10	10	10	10	10	10	10
1	2	3	4	4	4	4	4	4	4	10	10	10	10	10	10	10	10	10	10
1	2	3	4	5	5	5	5	5	5	10	10	10	10	10	10	10	10	10	10
1	2	3	4	5	6	6	6	6	6	10	10	10	10	10	10	10	10	10	10
1	2	3	4	5	6	7	7	7	7	10	10	10	10	10	10	10	10	10	10
1	2	3	4	5	6	7	8	8	8	10	10	10	10	10	10	10	10	10	10
1	2	3	4	5	6	7	8	9	9	10	10	10	10	10	10	10	10	10	10
1	2	3	4	5	6	7	8	9	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10

Gambar 10 Rencana Heuristik Awal di Papan (Pemain 1)

Oleh karena itu, setelah beberapa perbaikan kami datang dengan model baru (Model Agresif) yang akan cenderung menaklukkan papan pemain lain setelah membuat jalan ke tengah papan. Seperti yang dapat dilihat pada Gambar 11 diasumsikan bahwa pemain adalah Player 1 (Potongan Merah), dapat ditunjukkan bahwa nilai sudut adalah 1 dan dengan cara naik ke tengah naik ke 10. Jadi, dalam sudut pandang ini pemain akan bergerak di luar areanya sendiri, meskipun mungkin ada potongan yang lebih besar nilainya yang dapat ditempatkan di areanya sendiri.

10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10

Gambar 11 Papan Model Heuristik Agresif

Keputusan Gerakan

Ketika kami mendekati keputusan yang akan dibuat dari Kecerdasan Buatan yang memanfaatkan algoritma yang disebutkan sebelumnya, kami ingin menyebutnya sebagai LeakyAI. Ini dinamai seperti itu ketika pendekatan datang untuk menemukan kebocoran atau celah antara potongan pemain lain dan apakah itu cukup berharga untuk menempatkannya di sana. Seperti yang kami lakukan di sini adalah menemukan nilai optimal bergerak tanpa melihat penempatan berikutnya yang mungkin menguntungkan seorang pemain baik dengan memprediksikan langkah tiga pemain lainnya itulah sebabnya algoritma Greedy sedang diterapkan di sini.

Hal pertama yang ingin kami ketahui sebelum membuat keputusan adalah posisi pemain lain, untuk menemukan kebocoran ini di papan, posisi ini adalah kanan dan bawah, kiri dan atas, bawah kiri, kanan atas adalah bagian lainnya. Kemudian kita mungkin ingin melanjutkan dengan terlebih dahulu mengkloning papan saat ini menggunakan fungsi yang disebutkan sebelumnya.

Kita kemudian dapat melanjutkan dengan menghitung ruang kosong kebocoran menggunakan algoritma FloodFill, apakah itu lebih besar dari kemungkinan potongan yang tersedia atau lebih kecil. Dalam kasus di mana ruang kebocoran lebih besar, kami kemudian memperbarui nilai heuristik sesuai dengan nilai yang kami dapatkan dari algoritma FloodFIll dan langkah terbesar yang mungkin. Sementara itu, jika tidak ada kebocoran yang ditangkap hanya menambahkan kemungkinan terbesar pindah ke nilai heuristik. Kami kemudian melakukan iterasi dan menemukan nilai heuristik terbesar yang mungkin. Setelah itu, kami ingin merekam potongan yang digunakan dan mengatur koordinat penempatan. Jika setelah semua iterasi dan

perhitungan, nilai heuristik ≤ 0 maka dapat disimpulkan bahwa tidak ada lagi gerakan untuk pemain. Algoritma untuk LeakyAI ditunjukkan di bawah ini.

N.B. nilai awal untuk CurrentHeuristic adalah 0, fungsi FloodFill mengembalikan jumlah persegi yang telah diisi, fungsi LargestMoveAt mengembalikan potongan terbesar yang dapat dimainkan di lokasi tersebut, dan array HeuristicBoard sesuai dengan Rencana Heuristik yang disebutkan di atas.

```
1   TRAVERSE Row 1 Until 20:
2     TRAVERSE Column 1 Until 20:
3       SET Leaky = false
4       IF Right AND Bottom is NOT vacant and ARE enemies:
5         SET Leaky = true
6       IF Left AND Top is NOT vacant and ARE enemies:
7         SET Leaky = true
8       IF Left AND Bottom is NOT vacant and ARE enemies:
9         SET Leaky = true
10      IF Top AND Right is NOT vacant and ARE enemies:
11        SET Leaky = true
12
13      IF Leaky is true:
14        FloodFillAmount += FloodFill(Row, Column, 0)
15        IF FloodFillAmount > Largest Piece Available:
16          CurrentHeuristic += FloodFillAmount
17      CurrentHeuristic += LargestMoveAt(Row, Column)
18      IF HeuristicBoard[Row][Col] is 10:
19        ACTIVATE Aggressive Mode
20  Place Highest Heuristic Piece at the Location
```

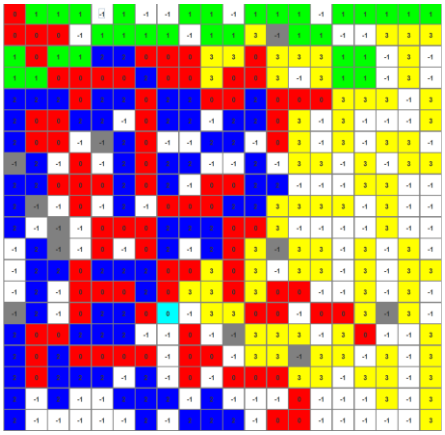
Gambar 12 Algoritma LeakyAI

IMPLEMENTASI DAN HASIL

Pada implementasinya kami telah mencoba menempatkan LeakyAI untuk bersaing dengan 3 Kecerdasan Buatan Brute Force lainnya. Brute Force AI hanya menempatkan bagian terbesar yang tersedia di dek dan memutarkannya untuk menemukan apakah cocok atau tidak, maka ia pergi dengan cara menurun menuju bagian terkecil. Hasilnya cukup konstan dengan LeakyAI memenangkan 100% permainan dengan total 20 poin di setiap pertandingan. LeakyAI ditempatkan sebagai Pemain Merah. Dengan papan skor seperti di bawah ini.

Tabel 4 LeakyAI vs Brute Force AI

Pemain	Poin
Merah	20
Hijau	-58
Biru	-13
Kuning	-10

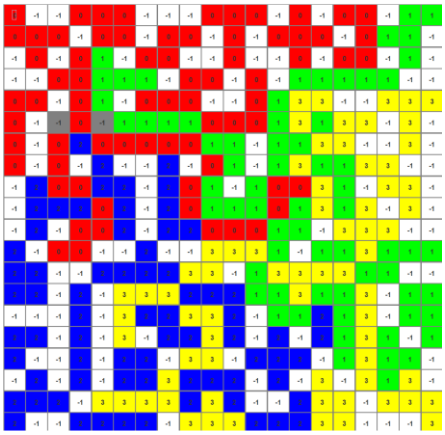


Gambar 13 LeakyAI vs Brute Force AI

Kemudian, kami mencoba untuk membandingkannya terhadap dirinya sendiri. Kami menempatkan 4 LeakyAI pada papan permainan dan tindakan yang dilakukan menunjukkan bahwa tidak ada LeakyAI yang berhasil menempatkan semua potongannya pada papan. Hasilnya ditunjukkan di bawah ini, dengan poin yang diraih tertinggi adalah -17.

Tabel 5 LeakyAI vs LeakyAI

Pemain	Poin
Merah	-17
Hijau	-22
Biru	-17
Kuning	-20



Gambar 14 LeakyAI vs LeakyAI

Dari permainan antara LeakyAI dengan dirinya sendiri dan LeakyAI dan Brute Force AI, tidak ada elemen keacakan dalam Kecerdasan Buatan yang dibangun. Kami menyimpulkan bahwa tidak peduli berapa banyak ronde/permainan yang dilakukan, hasilnya akan selalu menunjukkan apa yang ditampilkan pada Gambar 13 dan Gambar 14. Tapi, beberapa langkah pertama yang dilakukan oleh Kecerdasan Buatan ditempatkan di papan yang berbeda yang mengakibatkan potongan yang berbeda ditempatkan dengan cara yang berbeda.

Akhirnya, kami sendiri juga mencoba bersaing dengan LeakyAI sendiri setiap pengembang memainkan 3 putaran permainan (dengan total 9 putaran) bersaing dengan 3 LeakyAI yang tidak bekerja sama. Hasilnya diperlihatkan dalam gambar berikut.

N.B. "Dev." berarti pengembang dan "R" berarti putaran.

	Merah (Dev.)	Hijau	Biru	Kuning
Poin R1	-17	-27	-27	-34
Poin R2	-27	-15	-21	-9
Poin R3	-8	-23	-6	-21
Poin R4	-5	-27	-34	-20
Poin R5	20	-26	-8	-26
Poin R6	-20	-20	-8	-16
Poin R7	20	-32	-24	-39
Poin R8	-8	-32	-16	-17
Poin R9	-4	-8	-29	-18
Rata-Rata	-5,44	-23,33	-19,22	-22,22
MAX	20	-8	-6	-9
MIN	-27	-32	-34	-39

Gambar 15 Hasil Pengembang vs LeakAI

Berbeda dengan apa yang terjadi ketika papan dipenuhi dengan pemain Kecerdasan Buatan, bermain dengan manusia nyata (pengembang dalam hal ini) membawa gerakan acak dari LeakyAI yang menyesuaikan langkahnya ke langkah pemain lain. Dengan poin yang ditunjukkan dari sisi pengembang, masih mungkin bagi pengembang untuk mengalahkan LeakyAI dengan rata-rata tingkat kemenangan 66,67% (peringkat satu dari empat) dan dengan hanya rata-rata 22,22% mendapatkan skor penuh (tempatkan semua bagian pemain). Dengan rata-rata -5,44 poin dibandingkan dengan rata-rata LeakyAI terbaik dengan -19,22 poin. Pada keadaan optimalnya pengembang bisa mendapatkan skor penuh dengan 20 poin, sedangkan LeakyAI mendapat maksimal -6 poin. Di sisi lain, pada putaran permainan terburuk, pengembang berhasil mencapai -27 poin dibandingkan dengan LeakyAI terburuk adalah -39 poin. Tidak seperti apa yang dipraktikkan sebelumnya, ketika LeakyAI ditempatkan di papan dengan pemain nyata itu akan secara konsisten memberikan gerakan yang berbeda tergantung pada pemainnya. Kami menganalisis bahwa alur permainan LeakyAI adalah seperti yang telah disebutkan, bahwasannya cenderung agresif dan bergerak keluar dari sudutnya. Kami juga merasa sulit untuk menempatkan beberapa potongan tertentu karena LeakyAI selalu terlihat untuk mengisi celah atau kebocoran, di antara potongan-potongan.

KESIMPULAN

Makalah ini telah mengusulkan model permainan *Blokus Classic* bersama dengan implementasi Kecerdasan Buatan menggunakan bahasa pemrograman Java. Di sini kami telah menerapkan Kecerdasan Buatan untuk *pemain Blokus Classic* menggunakan algoritma Heuristik, FloodFill, dan Greedy secara bersamaan. Rencana Heuristik cukup banyak mempengaruhi seberapa LeakyAI akan memutuskan langkahnya dan mempengaruhi kinerjanya secara keseluruhan. Berdasarkan uji coba 9 permainan dengan para pengembang, masih banyak perbaikan yang bisa dilakukan kepada LeakyAI agar bisa mendapatkan peringkat terbaik. Selain

menggunakan algoritma berdasarkan teori, disarankan juga untuk menerapkan strategi dan trik dalam permainan Blokus. Merupakan suatu ide yang baik untuk memprediksi langkah pemain lain ke langkah-langkah tertentu untuk sistem keputusan yang lebih baik. Kemampuan ini dapat disediakan saat menerapkan *Monte Carlo Tree* Search atau MiniMax untuk pengembangan berikutnya.

DAFTAR PUSTAKA

[1] "Blokus | Board Game | BoardGameGeek." [Online]. Available: https://boardgamegeek.com/boardgame/2453/blokus. [Diakses: 01-Apr-2020].

[2] B. Goertzel, "The Hidden Pattern: A Patternist Philosophy of Mind," p. 470, 2006.

[3] "Blokus Duo | Board Game | BoardGameGeek." [Online]. Available: https://boardgamegeek.com/boardgame/16395/blokus-duo. [Diakses: 02-Apr-2020].

[4] N. Sugimoto, T. Miyajima, T. Kuhara, Y. Katuta, T. Mitsuichi, and H. Amano, "Artificial intelligence of Blokus Duo on FPGA using Cyber Work Bench," *FPT 2013 - Proc. 2013 Int. Conf. F. Program. Technol.*, pp. 498–501, 2013.

[5] A. Jahanshahi, M. K. Taram, and N. Eskandari, "Blokus Duo game on FPGA," *Proc. - 17th CSI Int. Symp. Comput. Archit. Digit. Syst. CADS 2013*, pp. 149–152, 2013.