TUGAS AKHIR – K141502

DESAIN DAN ANALISIS ALGORITMA PENGOPTIMALAN SKOR COLOUR BRICK GAME DENGAN PENGGABUNGAN FITUR PAIRING DAN POSSIBLE DISSOLVE SERTA PENENTUAN WEIGHT BERBASIS ALGORITMA GENETIKA STEADY STATE: STUDI KASUS SPOJ 18073

DESTIANA NURLIASARI
NRP 05111440000148

Dosen Pembimbing
Rully Soelaiman, S.Kom., M.Kom.
Wijayanti Nurul Khotimah, S.Kom., M.Sc.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2018

[Halaman ini sengaja dikosongkan]

TUGAS AKHIR – K141502

DESAIN DAN ANALISIS ALGORITMA PENGOPTIMALAN SKOR COLOUR BRICK GAME DENGAN PENGGABUNGAN FITUR PAIRING DAN POSSIBLE DISSOLVE SERTA PENENTUAN WEIGHT BERBASIS ALGORITMA GENETIKA STEADY STATE: STUDI KASUS SPOJ 18073

DESTIANA NURLIASARI
NRP 05111440000148

Dosen Pembimbing I
Rully Soelaiman, S.Kom., M.Kom.

Dosen Pembimbing II
Wijayanti Nurul Khotimah, S.Kom., M.Sc.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2018

[Halaman ini sengaja dikosongkan]

FINAL PROJECT – K141502

DESIGN AND ANALYSIS ALGORITHM TO OPTIMIZE SCORE IN COLOUR BRICK GAME WITH COMBINATION PAIRING FEATURE AND POSSIBLE DISSOLVE AND WEIGHT DETERMINATION BASED ON STEADY STATE GENETIC ALGORITHM: CASE STUDY SPOJ 18073

DESTIANA NURLIASARI
NRP 05111440000148

Supervisor I
Rully Soelaiman, S.Kom., M.Kom.

Supervisor II
Wijayanti Nurul Khotimah, S.Kom., M.Sc.

DEPARTMENT OF INFORMATICS
Faculty of Information and Communication Technology
Sepuluh Nopember Institute of Technology
Surabaya 2018

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

DESAIN DAN ANALISIS ALGORITMA PENGOPTIMALAN SKOR COLOUR BRICK GAME DENGAN PENGGABUNGAN FITUR PAIRING DAN POSSIBLE DISSOLVE SERTA PENENTUAN WEIGHT BERBASIS ALGORITMA GENETIKA STEADY STATE: STUDI KASUS SPOJ 18073

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Algoritma Pemrograman
Program Studi S-1 Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh :
DESTIANA NURLIASARI
NRP : 05 1 1 14 40 00 0148

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Rully Soelaiman, S.Kom., M.Kom. .....
NIP. 19700213 199402 1 001 (pembimbing 1)

Wijayanti Nurul Khotimah, S.Kom., M.Sc. .....
NIP. 19860312 201212 2 004 (pembimbing 2)

SURABAYA
JULI 2018

[Halaman ini sengaja dikosongkan]

DESAIN DAN ANALISIS ALGORITMA PENGOPTIMALAN SKOR COLOUR BRICK GAME DENGAN PENGGABUNGAN FITUR PAIRING DAN POSSIBLE DISSOLVE SERTA PENENTUAN WEIGHT BERBASIS ALGORITMA GENETIKA STEADY STATE: STUDI KASUS SPOJ 18073

Nama Mahasiswa	:	Destiana Nurliasari
NRP	:	05 1 1 14 40 00 0148
Jurusan	:	Departemen Informatika-ITS
Dosen Pembimbing 1	:	Rully Soelaiman, S.Kom., M.Kom.
Dosen Pembimbing 2	:	Wijayanti Nurul Khotimah, S.Kom., M.Sc.

ABSTRAK

Colour brick game adalah permasalahan berbentuk permainan dimana pemain harus menyusunkan brick yang terdiri dari 3 kotak dengan nilai warna dan brick-brick tersebut harus disusun agar terdapat minimal 3 kotak dengan warna yang sama secara horizontal, vertikal atau diagonal, sehingga 3 kotak itu lenyap dan mendapatkan skor. Tujuan dari permasalahan ini adalah menyelesaikan permainan dengan skor semaksimal mungkin atau disebut optimasi skor.

Desain algoritma penyelesaian colour brick game ini mengacu pada penyelesaian permainan tetris dikarenakan kemiripan dari dua permainan tersebut. Tugas akhir ini menggunakan algoritma genetika untuk mendapatkan weight optimal yaitu weight yang menghasilkan skor optimal.

Proses pengujian mengaplikasikan weight hasil algoritma genetika dalam kode penyelesaian colour brick game yang diunggah pada SPOJ. Hasil pengujian dari tugas akhir ini memperoleh peringkat 1 pada ranking SPOJ.

Kata kunci: ***algoritma genetika , optimasi skor, tetris, weight.***

[Halaman ini sengaja dikosongkan]

DESIGN AND ANALYSIS ALGORITHM TO OPTIMIZE SCORE IN COLOUR BRICK GAME WITH COMBINATION PAIRING FEATURE AND POSSIBLE DISSOLVE AND WEIGHT DETERMINATION BASED ON STEADY STATE GENETIC ALGORITHM: CASE STUDY SPOJ 18073

Student Name : Destiana Nurliasari
Student ID : 05 1 1 14 40 00 0148
Major : Informatics Department -ITS
1st Supervisor : Rully Soelaiman, S.Kom., M.Kom.
2nd Supervisor : Wijayanti Nurul Khotimah, S.Kom., M.Sc.

ABSTRACT

Colour brick is a game-shaped problem where player arrange brick that formed from 3 squares with colour value in every square and the brick must be arranged to form minimum 3 squares with same colour horizontal, vertical, or diagonal, for 3 squares to dissolve and player get score. Player don't get information about next input of brick colour combination. This problem purpose is to finish the game with maximum score or score optimization.

Design of algorithm to solve colour brick game refer to tetris solution because the similarity between the two games. This final project uses genetic algorithm to produce optimum weight that result in optimum score.

The testing process use weight result from genetic algorithm in solution code for colour brick game that uploaded to SPOJ. Testing result of this final project is rank 1 on SPOJ ranking.

Keywords* : *genetic algorithm, score optimization, tetris, weight.

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Segala puji syukur bagi Allah SWT yang telah melimpahkan rahmat dan anugerah-Nya sehingga penulis dapat menyelesaikan tugas akhir yang berjudul "DESAIN DAN ANALISIS ALGORITMA PENGOPTIMALAN SKOR COLOUR BRICK GAME DENGAN PENGGABUNGAN FITUR PAIRING DAN POSSIBLE DISSOLVE SERTA PENENTUAN WEIGHT BERBASIS ALGORITMA GENETIKA STEADY STATE: STUDI KASUS SPOJ 18073".

Selesainya tugas akhir ini tidak lepas dari bantuan dan dukungan berbagai pihak. Pada kesempatan ini penulis mengucapkan terima kasih kepada:

1. Kedua orang tua, Ir. Zaenal Abidin, M.M., M.T. serta Ir. Mieke Juli Astuti, M.Si. atas doa dan dukungan material maupun moral selama penulis belajar di Teknik Informatika ITS.
2. Bapak Rully Soelaiman, S.Kom., M.Kom. dan ibu Wijayanti Nurul Khotimah, S.Kom., M.Sc. selaku dosen pembimbing penulis yang telah mengarahkan penulis dalam pengerjaan tugas akhir ini.
3. Teman penulis Afifah dan 'Ulya yang telah memberikan motivasi, ide dan dukungan moral pada penulis dalam pengerjaan tugas akhir ini.
4. Teman-teman seperti Tionia, Dini, Datin, Dave, Petrus dan teman-teman Teknik Informatika ITS angkatan 2014 yang telah memberi bantuan pada penulis dalam perkuliahan.

Penulis meminta maaf apabila terdapat kesalahan atau kekurangan pada penulisan Tugas Akhir ini. Penulis mengharapkan kritik dan saran untuk perbaikan kedepannya.

Surabaya, Juli 2018

Destiana Nurliasari

[Halaman ini sengaja dikosongkan]

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

DAFTAR KODE SUMBER

DAFTAR PERSAMAAN

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

Pada bab ini dibahas mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika laporan tugas akhir. Dari penjelasan dalam bab ini gambaran tugas akhir secara umum dapat dipahami.

1.1. Latar Belakang

Colour Brick Game adalah permasalahan yang ada pada situs SPOJ dan digunakan pada kompetisi *Bubble Cup* 7 babak 2, permasalahan ini berbentuk permainan dimana pemain harus menyusunkan *brick* tersusun dari 3 kotak yang memiliki nilai warna pada tiap kotaknya dan *brick*-*brick* tersebut harus disusun agar terdapat minimal 3 kotak dengan warna yang sama secara horizontal, vertikal atau diagonal, sehingga 3 kotak itu lenyap dan pemain mendapatkan skor. Pemain tidak diberikan informasi akan kombinasi dari nilai warna yang dapat muncul dari *brick* yang akan disusun selanjutnya pada permainan. Tujuan dari permasalahan ini adalah menciptakan program yang dapat menyelesaikan permainan dengan skor semaksimal mungkin. Dimana pada peletakan *brick* tersebut perlu dicari posisi yang berpengaruh maksimal pada peletakan brick selanjutnya.

Dalam pembuatan penyelesain permasalahan ini melibatkan algoritma untuk permasalahan mencari keadaan-keadaan yang dapat terjadi dari susunan *brick* dan algoritma untuk permasalahan pengambilan keputusan, dimana algoritma untuk menyelesaikan permasalahan tersebut telah banyak ditemukan. Dari berbagai jenis algoritma yang telah ditemukan tersebut, dicari kombinasi algoritma yang paling tepat untuk menyelesaikan permasalahan pada studi kasus SPOJ Challenge *Colour Brick Game*.

1.2. Rumusan Permasalahan

Rumusan masalah yang diangkat dalam tugas akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana mencari fitur yang digunakan untuk pengambilan keputusan penyelesaian masalah *Colour Brick Game* pada situs SPOJ?
2. Bagaimana mencari weight atau beban terhadap fitur yang digunakan untuk pengambilan keputusan penyelesaian masalah *Colour Brick Game* pada situs SPOJ?
3. Bagaimana kinerja algortima yang telah dibuat untuk permasalahan *Colour Brick Game*?

1.3. Batasan Masalah

Permasalahan yang dibahas dalam tugas akhir ini memiliki beberapa batasan, yaitu sebagai berikut:

1. Studi kasus yang digunakan adalah permainan *Colour Brick Game* pada situs SPOJ dengan nomer soal 18073 dan kode soal BRICKGM.
2. Waktu maksimum untuk program berjalan pada satu *testcase* adalah 10 detik.
3. Dalam setiap test case, ukuran dari board adalah 10 x 20.
4. Dalam setiap test case, brick memilik panjang 3 kotak yang direpresentasikan menjadi 3 buah angka.
5. Dalam setiap test case, jumlah brick maksimal adalah total *brick* (N) x total warna(N) $\leq$ 100.000.
6. Dalam setiap test case, jumlah warna yang terdapat brick adalah minimal 3 dan maksimal 9.

1.4. Tujuan

Tujuan dari pembuatan tugas akhir ini adalah merancang program yang dapat menyelesaikan permasalahan *Colour Brick Game* pada situs SPOJ nomer soal 18073 dengan mengoptimalkan skor akhir permasalahan.

1.5. Manfaat

Manfaat dari hasil pembuatan tugas akhir ini adalah tugas akhir ini dapat membantu dalam penyelesaian masalah yang berhubungan dengan pengambilan keputusan dalam tahapnya untuk mendapatkan hasil maksimum dari rangkaian keputusan yang telah diambil sebelumnya.

1.6. Metodologi

Tahapan-tahapan yang dilakukan dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Penyusunan proposal Tugas Akhir.

   Tahap awal yang dilakukan dalam pengerjaan Tugas Akhir ini adalah penyusunan proposal Tugas Akhir. Proposal tugas akhir berisi tentang pendahuluan dari tugas akhir yang akan dibuat. Pendahuluan ini terdiri atas beberapa bagian yaitu latar belakang diajukannya usulan tugas akhir, rumusan masalah yang diangkat, batasan masalah untuk tugas akhir, tujuan dari pembuatan tugas akhir, dan manfaat dari hasil pembuatan tugas akhir. Pada proposal tugas akhir ini juga terdapat tinjauan pustaka yang digunakan sebagai referensi pendukung pembuatan tugas akhir, metodologi berisi penjelasan mengenai tahapan penyusunan tugas akhir dan jadwal kegiatan yang menjelaskan jadwal pengerjaan tugas akhir.

2. Studi literatur

   Pada tahap ini, akan dilakukan studi mengenai permasalahan pengambilan keputusan dengan nilai bobot atau heuristiknya serta algoritma yang akan digunakan. Studi bersumber dari buku literatur, paper, situs pembelajaran online dan materi perkuliahan terkait.

3. Perancangan

   Tahap ini meliputi perancangan algorithma dan penentuan beban berdasarkan studi literatur dan pembelajaran konsep yang ada. Tahap ini mendefinisikan kerangka algorithma berjalannya program, menentukan algorithma yang digunakan untuk melatih program, dan menetukan beban yang digunakan sebagai pertimbangaan dalam pengambilan keputusan.

4. Implementasi

   Pada tahap ini, algoritma yang telah dipelajari akan dibangun menjadi sebuah kecerdasan buatan yang dapat digunakan. Bahasa pemrograman yang digunakan dalam pembangunan adalah C++ dengan IDE Dev-C++.

5. Pengujian dan evaluasi

   Pada tahap ini, sumber kode dari kecerdasan buatan yang telah dibuat diunggah ke situs SPOJ. Pengujian dan evaluasi pada sumber kode yang telah dibuat untuk mengetahui kemampuan algoritma yang dipakai, mengamati kinerja program, serta mengidentifikasi kendala yang mungkin timbul pada program yang dibuat.

6. Penyusunan buku Tugas Akhir.

   Pada tahapan ini disusun buku yang memuat dokumentasi mengenai pembuatan serta hasil dari implementasi perangkat lunak yang telah dibuat.

1.7. Sistematika Penulisan Laporan Tugas Akhir

Buku tugas akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan tugas akhir ini. Selain itu, dapat berguna untuk pembaca yang tertarik untuk melakukan pengembangan lebih lanjut. Secara garis besar, buku tugas akhir terdiri atas beberapa bagian seperti berikut ini.

1. Bab I. Pendahuluan
   Bab ini berisi penjelasan mengenai latar belakang masalah, tujuan, dan manfaat dari pembuatan Tugas Akhir. Selain itu rumusan permasalahan, batasan masalah, dan sistematika penulisan juga merupakan bagian dari bab ini.

2. Bab II Dasar Teori
   Bab ini berisi penjelasan tentang deskripsi permasalahan, dan metode yang digunakan.

3. Bab III Analisis dan Perancangan
   Bab ini berisi penjelasan mengenai desain, perancangan, bahan, dan pemodelan algorithma yang digunakan dalam Tugas Akhir ini yang direpresentasikan dengan *pseudocode*.

4. Bab IV. Implementasi
   Bab ini merupakan pembangunan program dengan bahasa C++ sesuai permasalahan dan batasan yang telah dijabarkan pada Bab I.

5. Bab V. Uji Coba dan Hasil
   Bab ini berisi penjelasan mengenai hasil percobaan dan pembahasan mengenai hasil percobaan yang telah dilakukan.

6. Bab VI. Kesimpulan dan Saran
   Pada bab ini diberikan saran-saran yang berisi hal-hal yang masih dapat dikerjakan dengan lebih baik dan dapat dikembangkan lebih lanjut, atau berisi masalah-masalah yang dialami pada proses pengerjaan Tugas Akhir.

[Halaman ini sengaja dikosongkan]

BAB II
DASAR TEORI

Bab ini berisi penjelasan tentang permasalahan *Colour Brick Game* dan teori-teori yang digunanakan dalam peyelesaian permasalahan *Colour Brick Game*. Penjelasan ini bertujuan untuk menjelaskan dasar teori yang ada dalam pengerjaan tugas akhir.

2.1. Deskripsi Permasalahan

Colour Brick Game adalah permasalahan berbentuk permainan dimana pemain harus menyusunkan *brick* yang terdiri dari 3 kotak yang masing-masing kotak memiliki nilai warna. Arena dari permainan ini adalah papan permainan berukuran 10 x 20. Pemain akan mendapatkan skor bila terdapat minimal 3 kotak dengan warna yang sama bertetangga secara horizontal, vertikal atau diagonal, terususun pada papan permainan. Kotak yang telah tersusun dengan warna sama akan lenyap. Posisi yang kosong akibat lenyapnya kotak akan mengakibatkan kotak yang berada di posisi atas dari posisi kotak lenyap turun mengisi kekosongan. Bila setelah turunnya kotak pada posisi yang kosong terbentuk 3 atau lebih kotak yang bertetangga dengan warna sama pada papan permainan, maka kotak-kotak tersebut akan lenyap dan kotak pada posisi atas akan mengisi kekosongan. Hal ini akan terjadi hingga pada permainan tidak ada lagi 3 atau lebih kotak yang bertetangga dengan warna sama.

Pemain tidak diberikan informasi akan kombinasi dari nilai warna yang dapat muncul dari *brick* yang akan disusun selanjutnya pada permainan. Permainan akan terhenti bila tidak ada lagi *brick* yang dimasukan atau tidak ada lagi tempat untuk diletakkanya *brick* masukan pada papan permainan. Tujuan dari permasalahan ini adalah menyelesaikan permainan dengan skor semaksimal mungkin dengan batas waktu 10 detik untuk 1 testcase permainan.

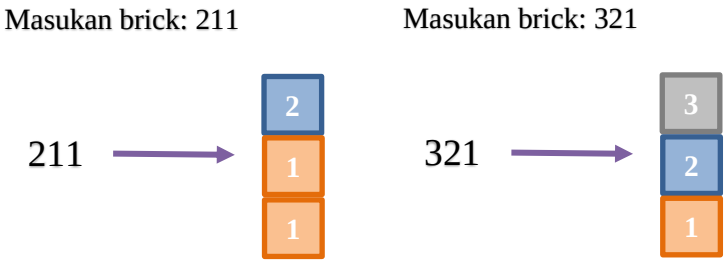
Contoh Masukan dan Keluaran:

Masukan	Keluaran
11 5 211 321 232 233 345 245 451 332 451 332 312	 1 1 2 0 3 0 3 2 4 0 5 0 6 0 6 0 7 0 7 0 8 1

Tabel 2.1 Tabel contoh masukan dan keluaran

2.1.1. Format Masukan

Masukan pada baris pertama adalah integer *N* merupakan banyaknya *brick* dalam permainan dan integer *K* ($3 \leq K \leq 9$) yang merupakan banyaknya jenis warna pada *brick* yang terlibat. Pada *N* baris selanjutnya masukan berupa rangkaian 3 buah integer yang merepresentasikan *brick* masuk sedangkan nilai integer merepresentasikan warna.



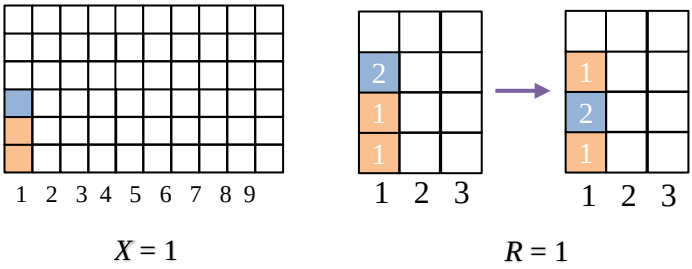
Gambar 2.1 Ilustrasi brick masukan 211 dan 321

Contoh dari masukan dapat dilihat pada *Tabel 2.1*, dimana masukan *N* bernilai 11 dan masukan *K* bernilai 5. Sehingga terdapat 5 jenis warna berpartisipasi dalam permainan, warna tersebut direpresentasikan oleh bilangan 1, 2, 3, 4 dan 5. Kemudian akan dimasukan *brick* yang direpresentasikan oleh 3 integer sebanyak 11 kali yang sesuai dengan *K*. Dimana pada contoh, *brick* pertama tersusun dari 211 dan *brick* kedua masukan tersusun dari 321, kedua masukan ini diilustrasikan pada *Gambar 2.1* .

2.1.2. Format Keluaran

Untuk setiap *brick* yang dimasukkan menghasilkan keluaran berupa integer *X* ($1 \leq X \leq 10$) dan integer *R* ($0 \leq R \leq 2$). Dimana integer *X* adalah posisi kolom dimana *brick* masukan akan diletakkan dan integer *R* adalah banyaknya rotasi yang dilakukan terhadap urutan kotak yang menyusun *brick.*

Keluaran: 1 1 terhadap *brick* 211



Gambar 2.2 Ilustrasi keluaran 1 1 pada brick 211

Contoh dari pengamplikasian keluaran terhadap brick masukan dapat dilihat pada Gambar 2.2. Dimana brick masukan yang diterima adalah 211 dan dikeluarkan aksi 1 1 yang dapat diartikan sebagai *X* bernilai 1 dan *R* bernilai 1. Hal ini menandakan bahwa dilakukan aksi peletakan brick

masukan pada papan permainan di posisi ke-1 dan rotasi sebanyak 1 kali sehingga masukan 211 menjadi 121. Contoh masukan dan keluaran secara menyeluruh dapat dilihat pada *Tabel 2.1*.

2.1.3. Perhitungan Skor

Untuk setiap peletakan *brick* yang menghasilkan minimal 3 kotak bertetangga dengan warna yang sama secara horizontal, vertikal atau diagonal, pemain akan mendapat nilai berupa skor. Skor yang didapatkan akan dihitung dengan menjumlahkan skor langkah ke-1, yaitu lenyapnya kotak akibat peletakan *brick* hingga skor langkah ke-s yang didapatkan akibat reaksi berantai dari lenyapnya kotak pada papan permainan sebanyak s kali.

Rumus perhitungan skor:

$$Skor_i = ((b_1)^2 + (b_2)^2 + \cdots + (b_n)^2) \times n \times i \quad (1)$$

s = Jumlah langkah lenyapnya kotak pada 1 peletakan.
i = Langkah ke-i dari pelenyapan kotak (0 < i < s).
n = Jumlah susunan kotak lenyap yang terjadi.
$b_{1...n}$ = Panjang kotak lenyap ke 1 hingga n.

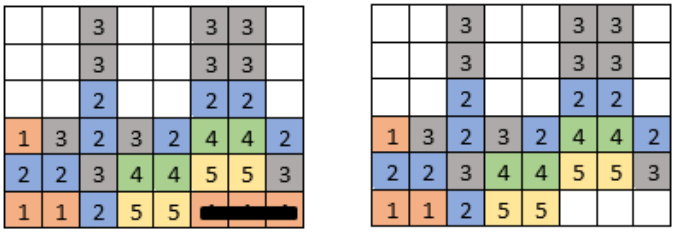
Sehingga skor yang didapatkan untuk peletakan brick:

$$Skor_{brick} = \sum_{i=1}^{s} Skor_i \quad (2)$$

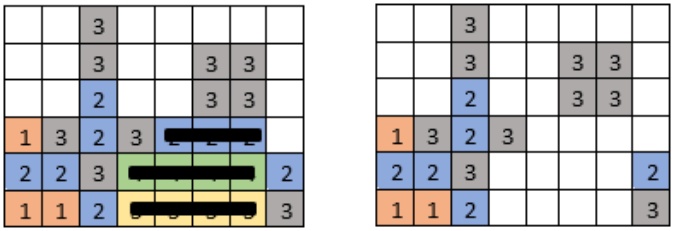
2.1.3.1. Simulasi Perhitungan Skor

Pada bagian ini akan disimulasikan proses perhitungan skor dan pelenyapan kotak pada papan permainan. Simulasi ini mengacu pada masukan dan keluaran pada *Tabel 2.1*. Ilustrasi dari simulasi ini dapat dilihat pada Gambar 2.3.

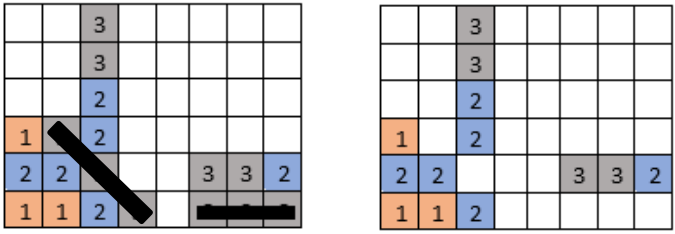
Langkah 1



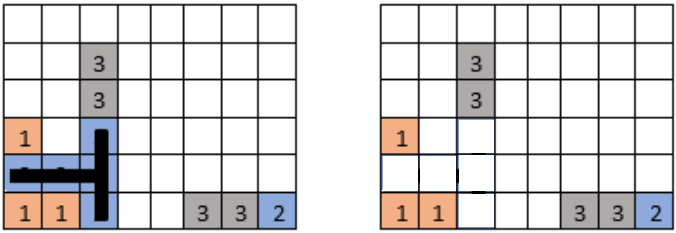
Langkah 2



Langkah 3



Langkah 4



Gambar 2.3 Simulasi pelenyapan kotak dengan s = 4

Dari pelenyapan kotak pada Gambar 2.1, maka dapat dihitung skor menggunakan rumus pada subbab 2.1.3, perhitungannya adalah sebagai berikut:

Langkah 1:

$$Skor_1 = (3^2) \times 1 \times 1 = 9$$

Langkah 2:

$$Skor_2 = (3^2 + 4^2 + 4^2) \times 3 \times 2 = 246$$

Langkah 3:

$$Skor_3 = (3^2 + 3^2) \times 2 \times 3 = 108$$

Langkah 4:

$$Skor_4 = (3^2 + 3^2) \times 2 \times 4 = 144$$

Skor yang didapatkan dari peletakan *brick* 312:

$$Skor_{brick} = 9 + 246 + 108 + 144 = 507$$

2.2. Algoritma Genetika

Algoritma Genetika (GA) adalah algoritma yang terinspirasi dari teori evolusi pada biologi. Algoritma genetika pada umumnya beroperasi secara berulang memperbarui populasi yang terdiri dari kumpulan kromosom yang merepresentasikan variabel atau data yang dioptimasi atau dicari. Dalam setiap perulangannya, populasi akan dievaluasi menggunakan nilai fitness. Kemudian populasi baru akan terbentuk menggunakan hasil operasi seleksi terhadap nilai fitness dimana kromosom terbaik akan disertakan pada populasi untuk generasi selanjutnya sedangkan kromosom yang lain akan digantikan oleh kromosom baru yang berasal dari hasil operasi rekombinasi atau mutasi dari kromosom orang tua. Algoritma genetika akan berhenti melakukan

perulanggan ketika telah mencapai batas iterasi yang ditentukan atau telah mencapai kondisi konvergen.

Salah satu tipe dari algoritma genetika adalah genitor atau algoritma genetika ‘steady-state’ [1]. Kelebihan dari ‘steady-state’ adalah kromosom yang paling baik ditemukan pada pencarian tetap dipertahankan pada populasi, hal ini menghasilkan pencarian yang lebif agresif yang dalam praktiknya umumnya cukup efektif [2].

Pada ‘steady-state’ memiliki tiga perbedaan dari algoritma genetika pada umum, yaitu [2]:

1. Dua kromosom orang tua dipilih untuk bereproduksi dan kromosom anak hasil reproduksi akan langsung ditempatkan kembali pada populasi.
2. Kromosom anak hasil reproduksi tidak menggantikan posisi orang tua, tetapi menggantikan posisi dari krosomom yang tidak fit.
3. *fitness* di tempatkan berdasarkan ranking. Ranking membantu menjaga tekanan selektif yang lebih constant pada pencarian.

Weight yang akan ditraining oleh algoritma genetika berbentuk bilangan real. Sehingga kromosom yang direpresentasikan berbentuk *array* dengan tipe data float. Representasi kromosom secara langsung bernilai float ini memeliki kelebihan lebih cepat, lebih konsisten pada saat berjalan daripada mengubah kromosom dari float menjadi biner [3].

Banyaknya jumlah populasi yang digunakan adalah populasi = 200. Pemilihan populasi dengan besar 200 mengacu pada hasil uji coba perbandingan antara populasi dengan besar 50 hingga 300 pada subbab 5.4.1.

2.2.1. Perhitungan Nilai Fitness

Nilai *fitness* adalah suatu nilai yang digunakan untuk mengetahui seberapa dekat solusi desain untuk mencapai tujuan yang ditetapkan. Nilai *fitness* ini berguna mengarahkan simulasi pada algoritma genetika menuju desain optimal dari solusi. Untuk mengarahkan simulasi dilakukan evaluasi kelayakan dari kromosom pada populasi dengan menghitung nilai *fitness* dan proses ini biasa disebut *fitness function*.

2.2.2. Seleksi

Seleksi adalah operasi memilih kromosom pada populasi untuk dilakukan reproduksi. Kromosom dipilih berdasarkan dari nilai *fitness*. Seleksi dapat terjadi di dua tempat yaitu seleksi dari generasi saat ini untuk menjadi orang tua generasi selanjutnya (*parent selection*) dan seleksi dari kromosom untuk menjadi generasi selanjutnya (*survivor selection*). Mengacu pada subbab 2.2, dimana steady-state' genetic algorithm menempatkan *fitness* berdasarkan ranking. Sehingga seleksi yang digunakan adalah Elitism dan Rank Selection.

2.2.2.1. Elistism

Elitism adalah salah satu tipe dari *survivor selection* yang pada pembuatan generasi selanjutnya menyertakan satu atau beberapa kromosom terbaik. Elitism digunakan untuk mengantisipasi hilangnya kromosom terbaik.

Kromosom	Fitness Score
[1, 2.5, 4, 2]	0.09
[2, 1.5, 3.1]	1.54
[2.5, 1.7, 1.8]	0.12
[1.5, 1.5, 1.7]	1
[0.5, 0.2, 0.1]	0.68
[0.3, 0.7, 0.2]	0.1

Next Generation
[2, 1.5, 3.1]
New Kromosom
New Kromosom
New Kromosom
New Kromosom
New Kromosom

Gambar 2.4 Contoh elitism

2.2.2.2. Rank Selection

Rank selection adalah salah satu tipe *parent selection* dimana pemilihan kromosom dilakukan dengan memberikan peringkat pada kromosom dari populasi berdasarkan hasil fitness function. Kromosom diberikan probabilitas untuk terpilih berdasarkan urutan peringkat. Kromosom orang tua kemudian dipilih berdasarkan pemilihan acak dengan probabilitas hasil peringkat.

Kromosom	Fitness Score
[1, 2.5, 4, 2]	0.09
[2, 1.5, 3.1]	0.54
[0.5, 1.5, 1.2]	0.12

Rank	Kromosom	Fitness Score	Rank Prob
1	[2, 1.5, 3.1]	0.54	0.6
2	[0.5, 1.5, 1.2]	0.12	0.3
3	[1, 2.5, 4, 2]	0.09	0.1

Gambar 2.5 Contoh rank selection

2.2.3. Rekombinasi

Rekombinasi adalah operasi dimana kromosom orang tua bereproduksi dengan menggabungkan kedua atau lebih kromosom orang tua untuk menghasilkan kromosom anak. Pengertian umum tentang rekombinasi memiliki kemiripan dengan *crossover* yang lebih umum digunakan pada algoritma genetika[4]. Sehingga pada buku ini istilah yang digunakan adalah rekombinasi.

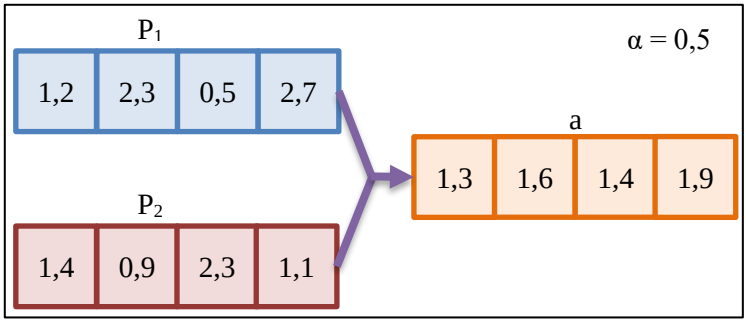
2.2.3.1. Arithmetic Recombination

Arithmetic atau intermediate recombination adalah metode yang hanya bisa digunakan pada variabel bernilai real. Pada metode ini anak yang dihasilkan akan dipilih dari nilai yang berada diantara nilai kedua orang tua.

Rumus dari arithmetic recombination:

$$a = P_1 \times \alpha + (1 - \alpha) \times P_2 \quad (3)$$

α = Presentase rekombinasi.
p1 = Kromosom orang tua ke 1.
p2 = Kromosom orang tua ke 2.
a = Kromosom anak hasil rekombinasi.



Gambar 2.6 Ilustrasi arithmetic recombination

Nilai α yang digunakan pada arithmetic recombination ini tidak konstan tetapi berada di batas antara 0,25 hingga 0,5. Dimana pada awal α = 0,25 dan terus naik hingga 0,5 kemudian turun bertahap kembali ke 0,25. Penggunaan α yang berubah dengan batas antara 0,25 hingga 0,5, dikarenakan hasil dari α yang dinamis lebih baik daripada statis [5].

2.2.4. Mutasi

Mutasi adalah operasi genetika untuk mempertahankan keragaman pada generasi populasi kromosom. Mutasi mengubah satu atau lebih nilai variabel gen pada kromosom dari nilai awalnya. Terjadi atau tidaknya mutasi pada kromosom bergantung pada probabilitas yang ditetapkan. Pada mutasi ini probabilitas yang digunakan adalah 0,3 yang mengacu hasil ujicoba pada subbab 5.4.2.

2.2.4.1. Normal distributed mutation

Normal distributed mutation adalah mutasi yang menggunakan distribusi gaussian atau biasa disebut distribusi normal. Mutasi ini merupakan mutasi yang dapat diterapkan pada variabel bernilai real. Fungsi ini digunakan bila kromosom memiliki ruang pencarian nilai real, tanpa konstrain dan tidak adanya informasi sebelumnya dimana menciptakan kriteria untuk mutasi yang digunakan agar tidak bias. Sehingga digunakannya normal distributed mutation, dimana distribusi normal berasal dari prinsip maksimum entropi, yaitu, tidak bias maksimal [6]. Rumus dari distribusi normal yang menjadi dasar dari mutasi [7]:

$$N(\mu, \sigma^2) = \mu + \sigma N(0,1) \quad (4)$$

Pada mutasi normalnya perubahan pada setiap variabel mengacu pada nilainya saat ini, sehingga untuk skala mutasi hanya menggunakan σ. Sehingga rumus dari normal distributed mutation adalah sebagai berikut [7]:

$$x_i' = x_i + N(0, \sigma^2) = x_i + \sigma N(0,1) \quad (5)$$

x_i' = Mutasi dari x_i
x_i = Nilai variabel saat ini
$N(0,1)$ = Standar distribusi normal
σ = Standar deviasi

2.2.5. Kondisi Konvergen

Kondisi konvergen dapat tercapai bila nilai standar deviasi dari kromosom-kromosom unggulan bernilai dibawah 0,0001. Standar deviasi sendiri adalah ukuran yang digunakan untuk mengukur jumlah dari variasi atau dispersi dari suatu data set. Standar deviasi dicari dengan terlebih dahulu mendapatkan nilai mean atau dari kromosom unggulan.

Mean atau rata-rata adalah hasil jumlah keseluruhan nilai dibagi jumlah variabel yang ditambah. Rumus dari mean:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \quad (6)$$

$\bar{x}$ = populasi mean

x_i = nilai data ke-i

n = jumlah pada populasi

Setelah didapatkan mean, maka dicari nilai standar deviasi mengunakan mean yang telah didapatkan. Rumus untuk mendapatkan standar deviasi adalah sebagai berikut:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}} \quad (7)$$

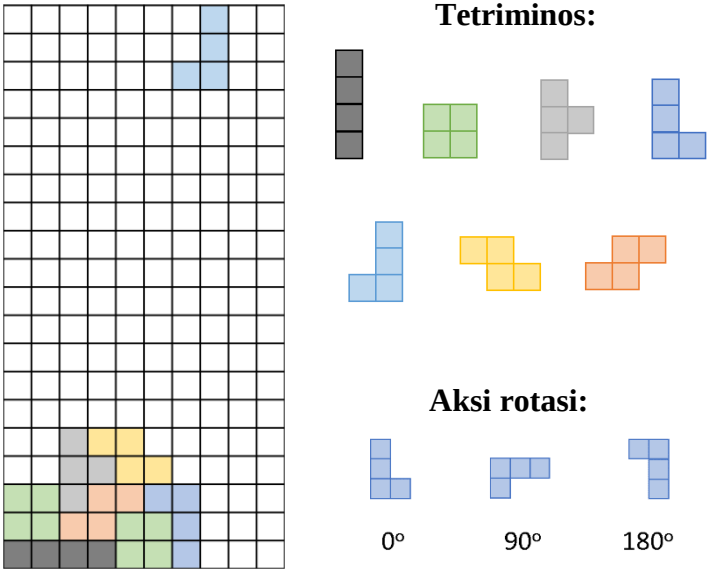
2.3. Tetris

Tetris adalah permainan tile-matching puzzle. Dimana disediakan sebuah papan permainan yang pada umumnya berukuran 10 x 20 kotak. Pemain akan diberikan masukan berupa sebuah tetriminos dengan berbagai bentuk (I, J, L, O, S, T, Z) yang harus disusun pada papan permainan. Gambaran dari permianan tetris dapat dilihat pada Gambar 2.7.

Tujuan dari permainan ini adalah untuk memanipulasi tetriminos masukan dengan menggerakkan masing-masing ke

samping (jika pemain merasa perlu) dan memutarnya dengan 90 derajat unit, dengan tujuan menciptakan garis horizontal dengan panjang sepuluh kotak tanpa celah pada papan permainan. Ketika garis tersebut berhasil dibuat, maka garis akan hancur, dan setiap blok di atas garis yang dihapus akan jatuh. Permainan tetris akan berakhir jika tidak ada lagi tetriminos atau papan permainan telah penuh.

Secara garis besar yaitu dari ukuran papan permainan dan cara bermain, tetris dan colour brick game memiliki persamaan. Dimana kedua permainan tersebut dimainkan dengan menata masukan pada papan permainan dengan aksi perputaran dan menentukan posisi peletakan dengan tujuan untuk menghancurkan kotak dan menghindari penuhnya papan permainan. Sehingga penyelesaian permainan tetris dapat diaplikasikan untuk menyelesaikan *colour brick game*.



Gambar 2.7 Permainan tetris

2.4. Penyelesaian Permainan Tetris

Tetris telah dianalisis secara teori oleh banyak penulis. Pada penyelesainnya, penulis lain menggunakan fitur, weight atau beban dan fungsi evaluasi sebagai penentu pada fungsi keputusan untuk menentukan aksi yang dilakukan [8].

Mengacu pada metode yang telah disebutkan, langkah pertama adalah menentukan set fitur yang dapat mengekstrak informasi yang relevan tentang permainan dan langkah kedua adalah penentuan weight relatif terhadap fitur[9]. Untuk mendapatkan weight relatif terhadap fitur, dilakukan training menggunakan genetic algorithm [10]. Setelah dijalankan genetic algorithm, didapatkan hasil training berupa nilai weight yang optimal. Dipilihnya genetic algorithm untuk mencari weight berdasarkan banyaknya referensi yang menggunakan genetic algorithm untuk mencari weight dan dilihat pada Tabel *2.2*.

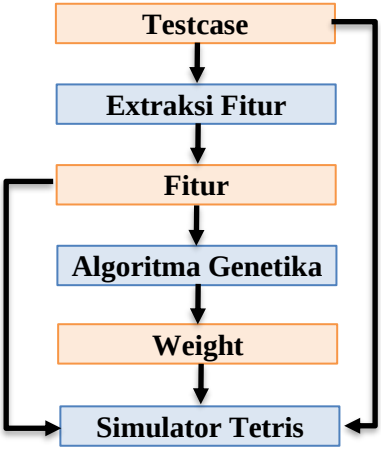
Penulis	Tahun
N. Böhm, G. Kókai, and S. Mandl.	2005
L. Flom and C. Robinson	2005
A. Boumaza	2009
D. Rollison and G. Wagner	2010
J. Lewis	2015
B. Abdelkarim	2016
A. Galavis, M. Noven, J. Spicer, R. Craig	2017

Tabel 2.2 Daftar penulis dan tahun pada referensi penyelesaian tetris yang menggunakan genetic algorithm

Tahap selanjutnya adalah tahap penyelesaian dimana fitur dan weight yang telah didapatkan kemudian diaplikasikan pada fungsi evaluasi dari simulator tetris yang bertugas untuk memainkan tetris. Fungsi evaluasi yang digunakan pada penyelesaian tetris adalah *weighted linear*

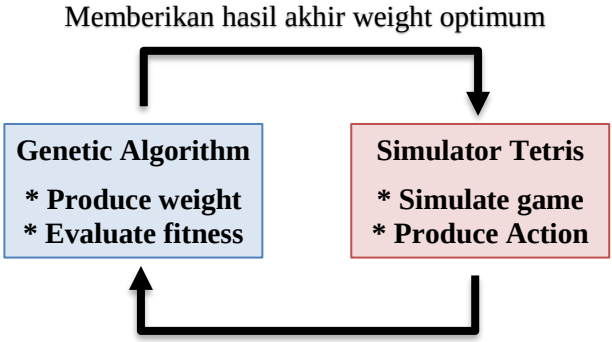
combination of feature [11]. Fungsi evaluasi ini akan melakukan evaluasi pada papan permainan yang merupakan hasil pemetaan aksi dari fungsi pemetaan.

Hasil dari fungsi evaluasi akan digunakan dalam fungsi keputusan. Dimana fungsi keputusan akan memilih hasil dari fungsi evaluasi tertinggi dari kemungkinan aksi peletakan tetriminos pada papan permainan sebagai keputusan yang dipilih. Alur kerja umum dari penyelesaian tetris dapat dilihat pada Gambar 2.8.



Gambar 2.8 Alur kerja umum penyelesaian tetris

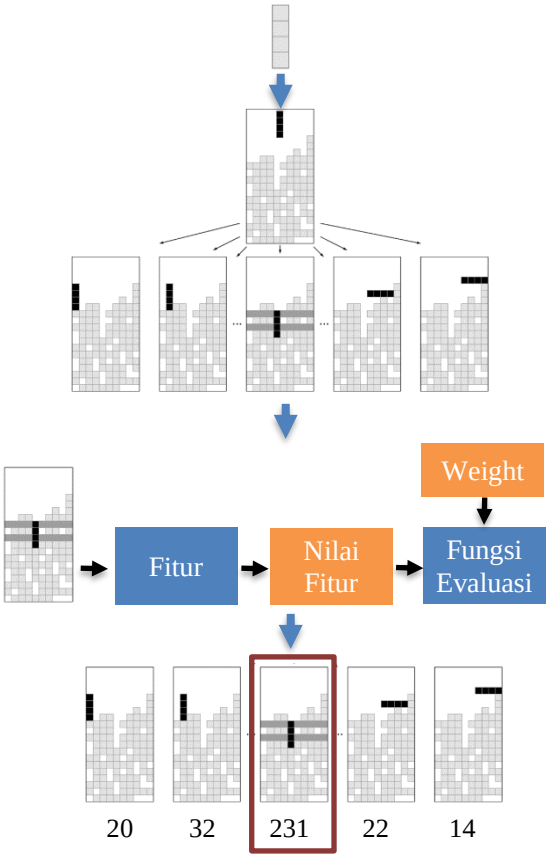
Dalam penerapannya, genetic algoritm yang digunakan untuk menemukan weight bukan merupakan bagian dari simulator tetris, melainkan penerapan genetic algorithm dan simulator tetris adalah dua buah program yang berbeda. Dimana simulator tetris tidak menyertakan genetic algorithm tetapi hanya menyertakan weight hasil dari genetic algorithm, sedangkan genetic algorithm dalam penerapanya mengaplikasikan simulator tetris pada *fitness function* untuk menghasilkan nilai *fitness*. Hubungan dari genetic algorithm dan simulator tetris dapat dilihat pada Gambar 2.9.



Gambar 2.9 Hubungan antara genetic algoritm dan simulator tetris

Alur kerja simulator tetris yang diilustrasikan pada Gambar 2.10 memiliki langkah-langkah sebagai berikut:

1. Tetriminos diterima.
2. Mencoba semua kemungkinan keputusan.
3. Mencari hasil dari fungsi evaluasi pada semua kemungkinan.
4. Memilih keputusan dengan hasil nilai dari fungsi evaluasi tertinggi.



Gambar 2.10 Ilustrasi alur simulator tetris

2.4.1. Ektraksi Fitur pada Tetris

Ekstraksi fitur pada tetris dilakukan dengan menganalisis informasi pada permainan tetris. Analisis dilakukan dengan mengamati papan permainan dan aksi yang dilakukan pemain saat memainkan tetris. Hasil dari ektraksi fitur adalah fitur atau fitur vektor.

2.4.2. Fitur pada Tetris

Fitur pada penyelesaian tetris digunakan sebagai salah satu faktor penentu keputusan. Dimana fitur ini digunakan untuk mendapatkan informasi nilai dari papan permainan yang digunakan pada fungsi evaluasi.

Untuk menyelesaikan Tetris, Böhm menggunakan fitur yang ia temukan dan fitur yang digunakan oleh Fahey dan Dellacherie. Berikut beberapa fitur yang diterapkan oleh Böhm [12]:

a. Pile Height: Posisi baris dari sel/kotak tertinggi yang telah ditempati pada papan permainan.
b. Holes: Banyaknya sel/kotak yang tidak ditempati dimana diatas sel/kotak tersebut ditempati.
c. Connected Holes: Banyaknya hole yang terhubung secara vertikal dengan hole lainnya, dimana kondisi ini hanya terhitung satu hole.
d. Removed Lines: Banyaknya baris yang berhasil dilenyapkan pada langkah terakhir untuk menghasilkan susunan papan permainan saat ini.
e. Maximum Well Depth: : Ukuran paling dalam dari susunan kotak yang membentuk sumur.
f. Landing Height: Posisi tinggi diletakkannya tetriminos.
g. Blocks: Banyaknya kotak ditempati pada papan.

2.4.3. Weight pada Tetris

Weight atau beban adalah pemberat atau pemberi nilai bagi fitur yang akan digunakan. Pada tetris weight digunakan untuk menentukan baik dan buruknya sebuah fitur terhadap kondisi yang diinginkan. Seperti yang disebutkan pada subbab 2.3, untuk mencari weight yang sesuai bagi tiap fitur digunakan maka weight akan dilatih menggunakan algoritma genetika.

2.4.4. Nilai Fitness pada Tetris

Pada tetris, nilai yang ingin dioptimalkan adalah banyaknya baris yang berhasil dilenyapkan. Sehingga mengacu pada subbab 2.2.1, nilai *fitness* dari tetris adalah banyaknya baris yang lenyap. Sehingga fungsi untuk menghitung nilai *fitness* adalah dengan memainkan permainan menggunakan kromosom yang akan dicari nilai *fitness*nya.

2.4.5. Fungsi Pemetaan

Fungsi Pemetaan (γ) adalah fungsi mengaplikasikan keputusan aksi yang diambil pada papan permainan. Pada tetris, fungsi ini menjalankan mekanisme permainan seperti perhitungan skor yang didapatkan dan pelenyapan kotak beserta mekanisme jatuhnya kotak pada posisi atas untuk mengisi kekosongan. Fungsi ini menghasilkan skor dari aksi dan papan permainan dengan keadaan terbaru.

Untuk melenyapkan kotak dan melakukan perhitungan skor maka perlu dilakukan pengecekan satu persatu kotak-kotak pada papan permainan untuk melihat ada tidaknya kotak yang memenuhi syarat untuk dilenyapkan. Bila terdapat maka kotak-kotak akan diberi penanda. Bila seluruh kotak pada papan telah selesai dicek, maka skor akan dikalkulasi dan kotak yang diberi tanda akan dilenyapkan. Kotak yang berada diatas kotak yang lenyap akan turun mengisi kekosongan. Langkah-langkah ini akan terus berulang hingga tidak ditemukan kotak yang memenuhi syarat untuk dilenyapkan.

2.4.6. Fungsi Evaluasi

Fungsi evaluasi dikenal juga sebagai fungsi evaluasi heuristic atau fungsi evaluasi static adalah sebuah fungsi yang digunakan oleh *game-playing* program untuk memperkirakan nilai atau baik tidaknya suatu posisi dalam algoritma terkait.

Pada subbab 2.4, disebutkan bahwa fungsi evaluasi yang digunakan adalah weighted linear combination of

feature. Dimana pada fungsi ini terdapat dua variabel yang berpengaruh yaitu fitur dan beban.

Rumus dari weighted linear combination of feature:

$$V(s) = \sum_{i=1}^{N} w_i f_i(s) \quad (8)$$

Fungsi evaluasi yaitu *V* adalah fungsi yang mengevaluasi terhadap *s* yaitu keadaan papan permainan, dimana hasil evaluasi didapatkan dengan menjumlahkan hasil perkalian nilai fitur f_i yang didapat dari keadaan *s* dengan weight yang berkorespondensi terhadap fitur.

2.4.7. Fungsi Keputusan

Fungsi keputusan atau decission function adalah fungsi yang menentukan pengambilan keputusan. Pada tetris, fungsi penentu keputusan dilakukan dengan mencari nilai maksimum dari hasil fungsi evaluasi terhadap papan permain hasil dari fungsi pemetaan kemungkinan aksi. Rumus dari fungsi keputusan adalah sebagai berikut:

$$d(s,p) = \arg \max_{d_i(p) \in D} V(\gamma(s, d_i(p))) \quad (9)$$

Penjelasan dari rumus diatas adalah sebagai berikut, *d(s, p)* merupakan fungsi keputusan yang mengambil keputusan terhadap tetriminos masukan yang berupa p dan *s* yang berupa keadaan papan permainan. *V* adalah fungsi evaluasi yang didalamnya menerima masukan fungsi pemetaan yaitu γ. Sedangkan γ sendiri adalah fungsi yang memetakan keputusan terhadap *p* pada *s* menggunakan keputusan aksi d_i.

2.5. Perbandingan Tetris dan Colour Brick Game

Parameter	Tetris	Colour Brick Game
Masukan	Tetriminos	Brick
Aksi	Posisi peletakan dan rotasi	Posisi peletakan dan rotasi
Aksi Rotasi	Derajat	Urutan
Papan permainan	Standar(10 x 20)	10 x 20
Penilaian	Banyaknya line yang lenyap	Skor
Kondisi Kalah	Tidak ada tempat untuk meletakkan masukan	Tidak ada tempat untuk meletakkan masukan
Cara Bermain	Menyusun masukan pada papan permainan agar mendapatkan nilai maksimal	Menyusun masukan pada papan permainan agar mendapatkan nilai maksimal
Susunan Lenyap	Line	3 kotak bertetangga sama warna

Tabel 2.3 Perbandingan tetris dan colour brick game

Pada subbab 2.3, dijelaskan bahwa tetris dan *colour brick game* memiliki kesamaan secara garis besar. Mulai dari cara bermain, papan permainan, dan aksi yang dilakukan pada masukan. Sehingga penerapan penyelesaian tetris pada colour brick game yang dijelaskan pada subbab 2.4 dapat dilakukan. Tetapi selain persamaan antara tetris dan *colour brick game* terdapat perbedaan yang membuat implementasi dari penyelesaian tetris pada *colour brick game* perlu dilakukan modifikasi dan adaptasi terhadap permainan *colour brick*

game. Persamaan dan perbedaan antara tetris dan colour brick game dapat dilihat pada Tabel *2.3*.

Dari *Tabel 2.3*, dapat dilihat bahwa terdapat tiga area dimana tetris dan colour brick game berbeda yaitu pada masukan, penilaian dan susunan. Ketiga perbedaan ini mempengaruhi implementasi dari penyelesain tetris pada *colour brick game* sehingga mengakibatkan berubahnya beberapa bagian pada penyelesaian tetris agar dapat diimplementsikan pada penyelesaian *colour brick game*.

Berikut adalah bagian-bagian yang berubah dari penyelesaian tetris dalam implementasi pada *colour brick game*:

a. Berbedanya fitur yang digunakan. Karena pada *colour brick game*, brick disusun sedemikian rupa hingga mendukung optimasi skor. Sehingga fitur yang digunakan sebagai salah satu faktor dalam fungsi evaluasi untuk menentukan keputusan pun berbeda. Maka perlu dilakukan ektraksi fitur untuk *colour brick game* agar mendukung pengoptimalan pada skor.
b. Nilai *fitness* yang berbeda pada algoritma genetika. Bila pada tetris digunakan nilai fitness berupa banyaknya baris yang dihancurkan. Maka pada colour brick game nilai fitness yang digunakan bukanlah banyaknya baris yang dihancurkan.
c. Fungsi keputusan yang diadaptasikan agar mengoptimasi nilai skor. Dikarena berbedanya susunan yang menjadi tujuan, maka fungsi keputusan juga mengalami penyesuain agar dapat menghasilkan keputusan yang mengoptimalkan nilai skor.

2.5.1. Extraksi Fitur pada Colour Brick Game

Masukan	Keluaran
11 5	
211	1 0
321	2 0
232	3 2
233	4 2
345	5 1
245	6 1
451	7 2
332	3 0
451	1 0
332	5 0
312	5 2

Tabel 2.4 Tabel contoh penyelesaian dengan greedy

Mengacu pada subbab 2.5, bahwa fitur yang digunakan pada tetris dan *colour brick game* berbeda maka perlu dilakukan ekstraksi fitur hingga menghasilkan fitur yang mendukung bagi optimasi skor pada *colour brick game*. Sebelum menentukan fitur pada permainan *colour brick game*. Perlu diketahui terlebih dahulu bahwa menyusun masukan untuk mendapatkan skor rantai lenyap seperti pada susunan yang diciptakan dari keluaran pada Tabel *2.1* menghasilkan skor lebih tinggi daripada menggunakan greedy yaitu menyusun masukan untuk mencari skor tertinggi pada saat itu juga. Hal ini ditunjukkan dengan membandingkan hasil skor dari keluaran pada Tabel *2.1* dan Tabel *2.4*. Dengan mengaplikasikan aksi pada keluaran Tabel *2.1*, skor yang berhasil dikumpulkan adalah sebanyak 507. Sedangkan mengaplikasikan aksi pada keluaran Tabel *2.4* hanya berhasil mengumpulkan skor sebanyak 97.

Dari hasil diatas, ektraksi fitur dilakukan dengan melihat susunan yang tersusun dari keluaran Tabel *2.1* sebelum terjadinya mekanisme lenyap, sebagaimana gambar pada Gambar 2.11.

Gambar 2.11 Susunan brick sebelum mekanisme lenyap

Melihat dari Gambar 2.11, yang menyusun terbentuknya susunan tersebut adalah kotak yang berpasangan sama warna secara horizontal, vertikal dan diagonal. Tanpa melihat kotak berwarna yang terdapat pada kolom 8, dapat dihitung bahwa dari 36 kotak yang menyusun susunan, terdapat 10 pasangan secara horizontal, 3 pasangan secara vertikal dan 8 pasangan secara diagonal. Sehingga banyaknya pasangan secara horizontal, vertikal dan diagonal dapat menjadi fitur pada colour brick game.

Hal lain yang perlu dilihat dari susunan pada gambar adalah hampir setiap pasangan memiliki kemungkinan untuk lenyap yaitu dengan adanya minimal 1 warna yang mungkin untuk melengkapi pasangan. Sehingga banyaknya kotak yang memungkinkan untuk lenyap dapat menjadi fitur dalam pertimbangan keputusan.

Fitur lainnya didapatkan dengan melakukan uji coba kombinasi fitur yaitu uji coba pada fitur yang ditemukan dari

colour brick game dan fitur pada tetris. Hasil uji coba kombinasi fitur dapat dilihat pada subbab 5.3, dimana didapatkan 2 fitur dari tetris yaitu Maximum Well Depth dan Blocks.Sehingga ektraksi fitur yang telah dilakukan menghasilkan enam fitur yaitu Horizontal Pair, Vertical Pair, Diagonal Pair, Possible Dissolve, Maximum Well Depth dan Blocks.

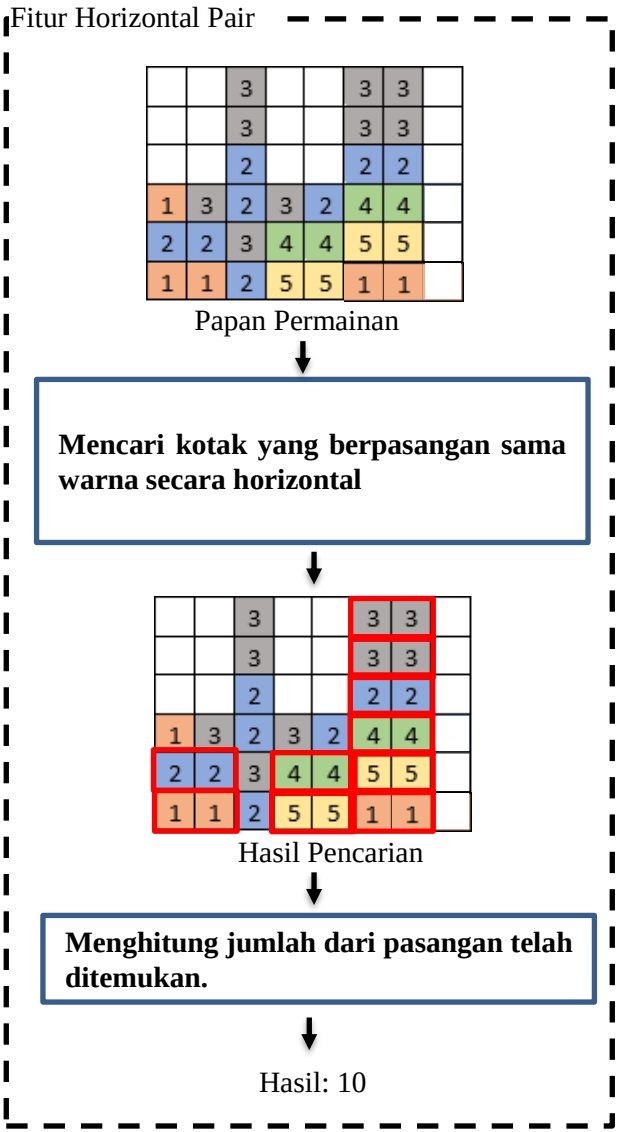
2.5.2. Fitur pada Colour Brick Game

Enam fitur hasil ektraksi yang digunakan dalam penentuan pengambilan keputusan pada colour brick game adalah:

a. Horizontal pair: Banyaknya kotak dengan warna yang sama berpasangan secara horizontal.
b. Vertical pair: Banyaknya kotak dengan warna yang sama berpasangan secara vertikal.
c. Diagonal pair: Banyaknya kotak dengan warna yang sama berpasangan secara diagonal.
d. Possible Dissolve: Banyaknya susunan kotak yang memungkinkan untuk lenyap.
e. Maximum Well Depth: Ukuran paling dalam dari susunan kotak yang membentuk sumur.
f. Blocks: Banyaknya kotak ditempati pada papan.

2.5.2.1. Fitur Horizontal Pair

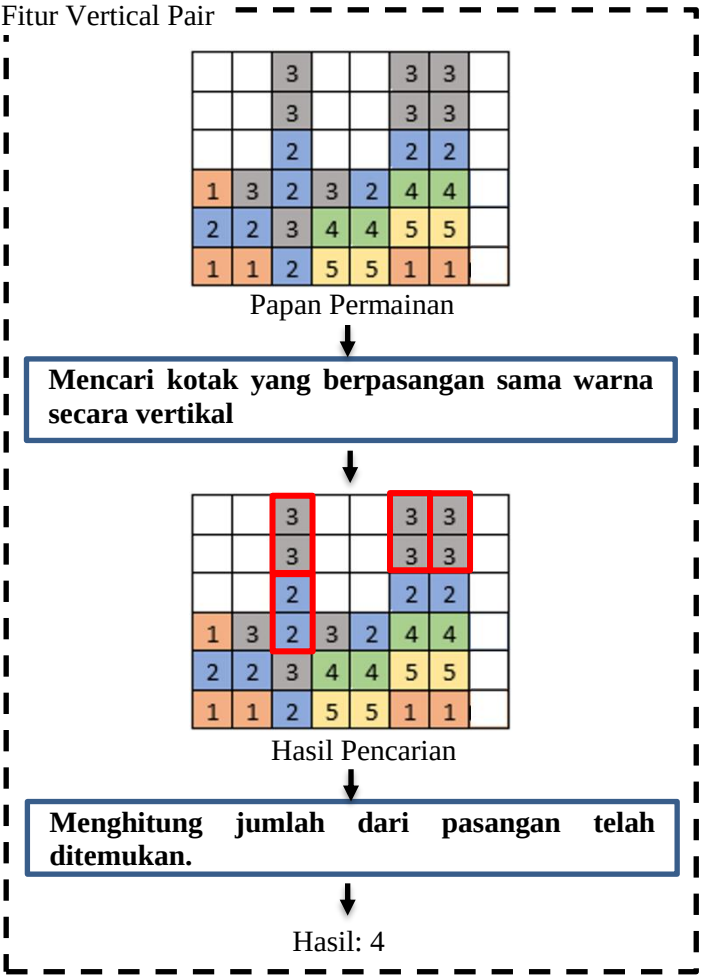
Fitur horizontal pair adalah fitur yang mencari kotak yang berpasangan sama warna secara horizontal dan menghitung jumlah dari banyaknya kotak yang bertetangga dengan sama warna secara horizontal. Alur kerja pada fitur *horizontal pair* dapat dilihat pada Gambar 2.12.



Gambar 2.12 Fitur Horizontal Pair

2.5.2.2. Fitur Vertical Pair

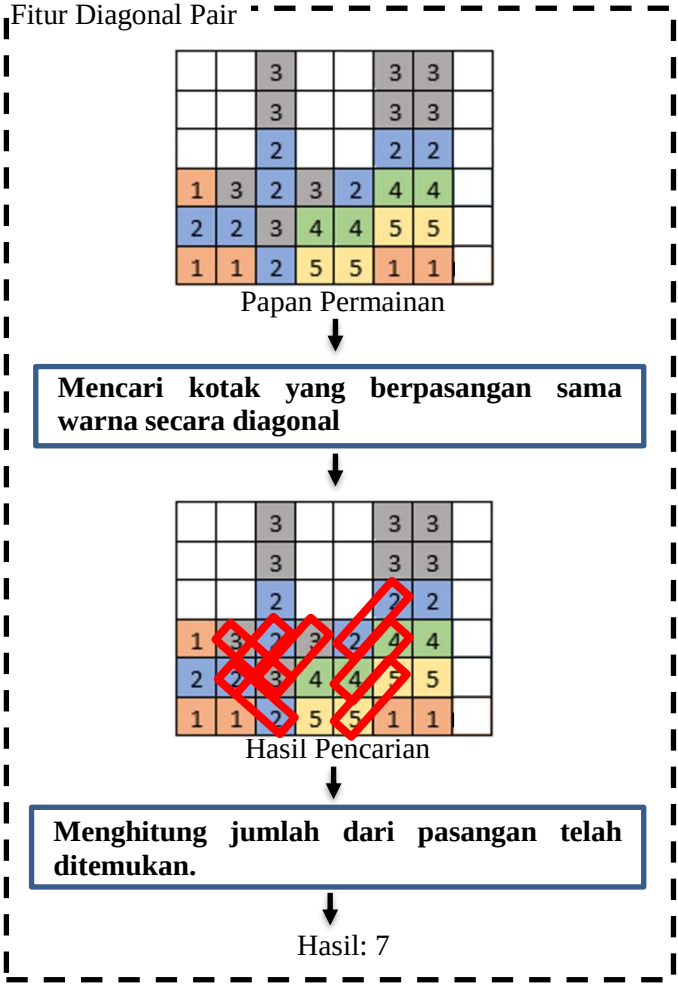
Fitur vertical pair adalah fitur yang mencari kotak berpasangan sama warna secara vertikal dan menghitung jumlah dari pasangan kotak.



Gambar 2.13 Fitur vertical pair

2.5.2.3. Fitur Diagonal Pair

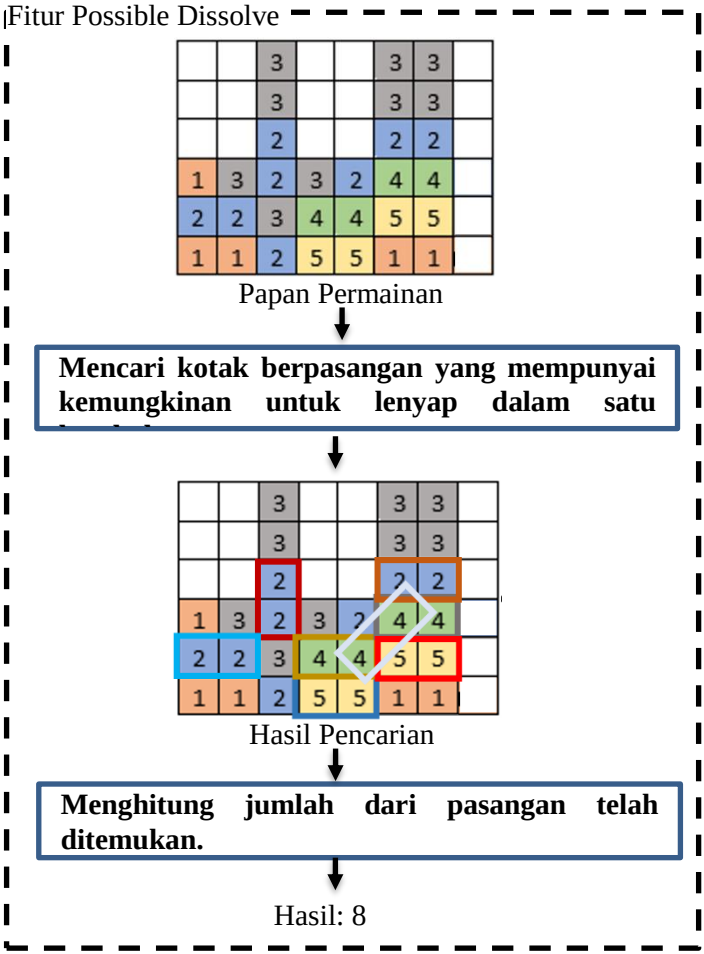
Fitur diagonal pair adalah fitur yang mencari kotak berpasangan sama warna secara diagonal dan menghitung jumlah dari pasangan kotak.



Gambar 2.14 Fitur diagonal pair

2.5.2.4. Fitur Possible Dissolve

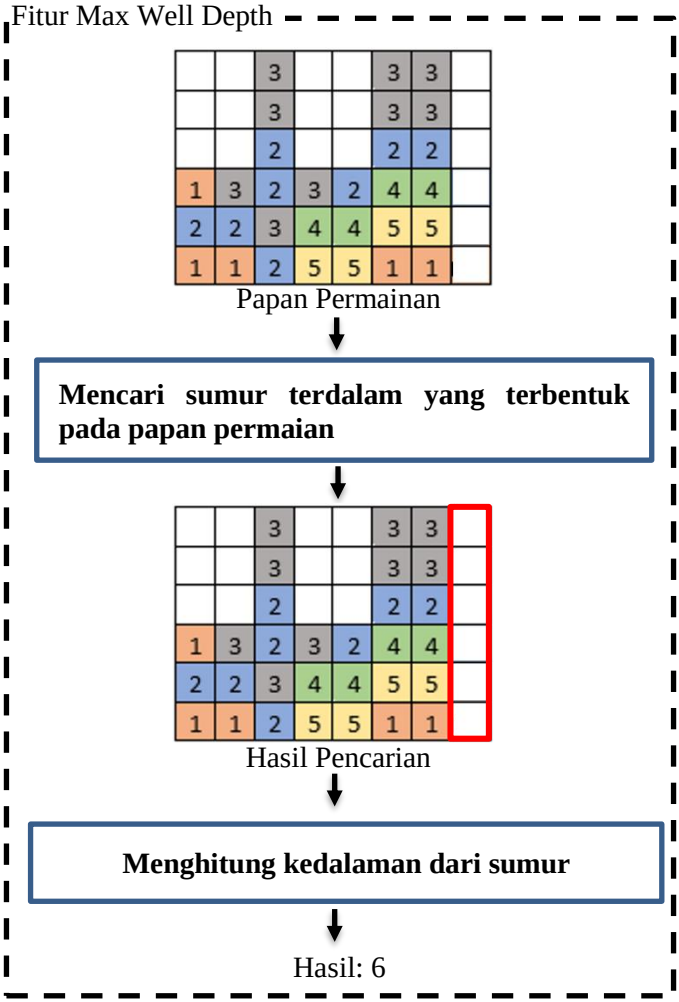
Fitur diagonal pair adalah fitur yang mencari kotak berpasangan sama warna yang memiliki kotak dengan warna sama disekitar yang memungkin terjadi lenyap salam satu langkah dan menghitung jumlah dari pasangan kotak tersebut.



Gambar 2.15 Fungsi possible dissolve

2.5.2.5. Fitur Max Well Depth

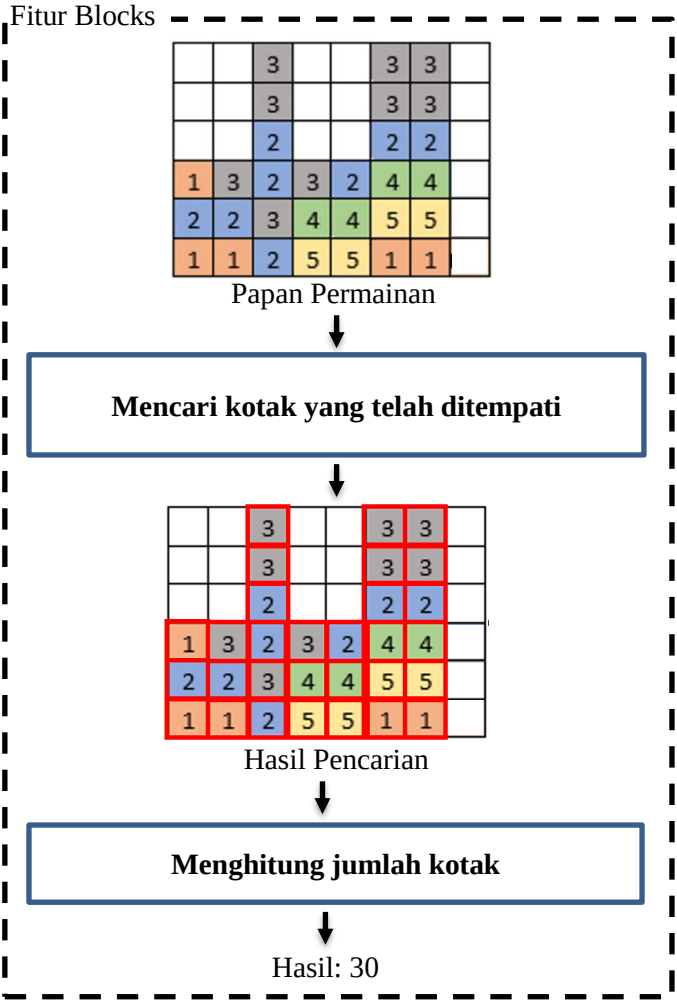
Fitur max well depth adalah fitur yang mencari sumur terdalam yang terbentuk pada papan permaian dan menghitung kedalaman dari sumur tersebut.



Gambar 2.16 Fitur max well depth

2.5.2.6. Fitur Blocks

Fitur block adalah fitur yang mencari kotak yang telah ditempati dan menghitung jumlah kotak tersebut.



Gambar 2.17 Fitur max well depth

2.5.3. Nilai Fitness pada Colour Brick Game

Pada colour brick game, nilai yang dioptimalkan adalah skor pada permainan. Untuk medapatkan skor yang optimal, perlu didapatkan rangkaian weight yang mendukung.. Nilai *fitness* sebagai parameter yang dapat mengarahkan pada optimasi skor dalam *colour brick game* adalah nilai skor itu sendiri.

2.5.4. Fungsi Keputusan pada Colour Brick Game

Pada tetris, fungsi penentu keputusan hanya terfokus pada satu pertimbangan seperti pada subbab 2.4.7. Sedangkan bila pada *colour brick game* hanya menggunakan satu tipe fungsi keputusan yaitu menggunakan aksi yang memiliki nilai maksimum dari fungsi evaluasi maka aksi hanya akan menyusun susunan yang memungkinkan untuk mendapatkan skor yang besar tanpa pernah melenyapkan susunan yang telah disusun. Sehingga diperlukannya fungsi keputusan yang dalam penentuan keputusannya menggunakan nilai maksimum dari skor yang didapatkan. Rumus dari mencari nilai maksimum sebagai fungsi keputusan adalah sebagai berikut:

$$d(s,p) = \arg \max_{d_i(p) \in D} Skor(\gamma(s, d_i(p))) \qquad (10)$$

d(s, p) = Hasil fungsi keputusan dari brick masukan dan kondisi papan saat itu.
d_i = Keputusan aksi i
Skor = Fungsi menghitung skor pada papan permainan hasil dari fungsi pemetaan.
γ = Fungsi pemetaan.

Untuk berpindah dari menggunakan nilai hasil fungsi evaluasi menjadi nilai skor diperlukan suatu kondisi yang menjadi trigger pada fungsi penentu keputusan. Salah satu parameter yang bisa dijadikan kondisi adalah skor

yang didapatkan, tempat tersisah dan langkah tersisah pada papan permainan.

Skor yang didapatkan menjadi kondisi untuk menghindari keputusan yang diambil menghasilkan skor yang tidak optimal. Dimana diperlukannya suatu kondisi batas terendah pada skor yang didapatkan pada kondisi normal permainan. Skor yang digunakan sebagai batas minimum didapatkan melalui uji coba.

Tempat tersisa digunakan sebagai kondisi agar pada saat permainan, program tidak mengalami kehabisan area untuk penempatan masukan pada papan permainan yang mengakibatkan kekalahan. Tempat tersisa ini menjadi trigger agar aksi yang dilakukan dapat melenyapkan kotak sehingga tersedia tempat yang lebih banyak pada papan permainan.

Langkah tersisah menjadi kondisi untuk mengantisipasi bila langkah tersisah memenuhi kondisi minimum. Sehingga keputusan yang diambil tetap berusaha mendapatkan skor semaksimal mungkin pada langkah yang tersisah dari permainan.

Berikut alur kerja dari fungsi keputusan pada *colour brick game*:

1. Brick diterima.
2. Mencoba semua kemungkinan keputusan dan mencari hasil dari fungsi evaluasi dan skor.
3. Memilih keputusan dengan hasil nilai dari fungsi evaluasi tertinggi. Mengulang langkah 1-3 hingga kondisi tepenuhi (skor minimum, langkah tersisah dan tempat tersisa).
4. Memilih keputusan dengan skor tertinggi.

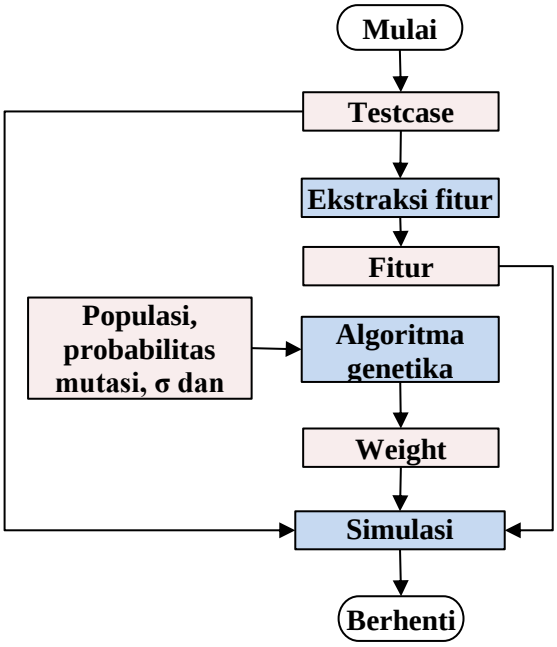
[Halaman ini sengaja dikosongkan]

BAB III
ANALISIS DAN PERANCANGAN

Bab ini berisi tentang penjelasan desain algoritma untuk menyelesaikan permasalahan SPOJ 18703 *Colour Brick Game*.

3.1. Desain Umum Sistem

Pada subbab ini akan membahas tentang rancangan secara umum dari sistem yang digunanakan untuk menyelesaikan permasalahan. Alur kerja untuk menjelaskan jalannya penyelesaian colour brick game secara umum dapat dilihat pada Gambar 3.1.



Gambar 3.1 Alur kerja dari penyelesaian tetris

Seperti yang telah dijelaskan pada Gambar 3.1 dan acuan pada subbab 2.4, dimana penyelesaian *colour brick game* ini berawal dari mengektraksi fitur untuk menentukan fitur yang digunakan. Kemudian fitur yang telah ditemukan dicari weightnya dengan menggunakan program algoritma genetika. Setelah didapatkan weight yang optimal, fitur dan weight tersebut diaplikasikan pada program simulasi yang mengeluarkan aksi dari masukan *brick* sebagai peneyelesaian permasalahan *colour brick game*. Dan sesuai dengan yang ada pada subbab 2.4 bahwa algoritma genetika dan simulasi permainan adalah dua program yang berbeda.

3.2. Ektraksi Fitur

Ektraksi fitur pada colour brick game dilakukan dengan menganalisis papan permainan dan melakukan uji coba kombinasi yang telah dijelaskan pada subbab 2.5.1.

3.3. Desain Fitur

Subbab ini akan membahas desain enam fitur yaituHorizontal Pair, Vertical Pair, Diagonal Pair, Possible Dissolve, Maximum Well Depth dan Blocks yang telah dijelaskan pada subbab 2.5.2.

3.3.1. Desain Fitur Horizontal Pair

Horizontal pair adalah fitur yang berfungsi untuk menghitung banyaknya pasangan sama warna secara horizontal yang terbentuk pada papan permainan. Pada perancangannya, fitur *horizontal pair* mengacu pada subbab 2.5.2.1. Desain pseudocode dari *horizontal pair* dapat dilihat pada Kode Sumber 3.1.

```
1: if j < 9 and GB[i][j] = GB[i][j + 1] then
2:   Increment ph
3: end if
```

Kode Sumber 3.1 Pseudocode *horizontal pair*

3.3.2. Desain Fitur Vertical Pair

Vertical pair adalah fitur yang berfungsi untuk menghitung banyaknya pasangan sama warna secara vertikal yang terbentuk pada papan permainan. Pada perancangannya, fitur *vertical pair* mengacu pada subbab 2.5.2.2. Desain pseudocode dari *vertical pair* dapat dilihat pada *Kode Sumber 3.2*.

```
1: if i > 0 then
2:   if GB[i][j] = GB[i - 1][j] then
3:     Increment pv
4:   end if
5: end if
```

Kode Sumber 3.2 Pseudocode *vertical pair*

3.3.3. Desain Fitur Diagonal Pair

Diagonal pair adalah fitur yang berfungsi untuk menghitung banyaknya pasangan sama warna secara diagonal yang terbentuk pada papan permainan. Pada perancangannya, fitur *diagonal pair* mengacu pada subbab 2.5.2.3. Desain pseudocode dari *diagonal pair* dapat dilihat pada Kode Sumber 3.3.

```
 1: if i > 0 and j < 9 then
 2:   if GB[i][j] = GB[i - 1][j + 1] then
 3:     Increment pd
 4:   end if
 5: end if
 6: if i > 0 and j > 0 then
 7:   if GB[i][j] = GB[i - 1][j - 1] then
 8:     Increment pd
 9:   end if
10: end if
```

Kode Sumber 3.3 Pseudocode *diagonal pair*

3.3.4. Desain Fitur Possible Dissolve

Possible dissolve adalah fitur yang berfungsi untuk menghitung banyaknya pasangan sama warna secara horizontal, vertikal dan diagonal yang memiliki warna sama disekitar pasangan tersebut dan memungkinkan untuk dilenyapkan dalam satu langkap peletakan atau pelenyapan. Pada perancangan fitur *possible dissolve*, perancangan fitu ini mengacu pada subbab 2.5.2.4 yang terdapat gambar alur kerja dari fitur ini. Desain pseudocode dari fungsi ini dapat dilihat pada Kode Sumber 3.4.

```
 1: if j < 9 and GB[i][j] = GB[i][j + 1] then
 2:    if j < 8 and i > 0 and GB[i][j] = GB[i - 1][j + 2] then
 3:       Increment p
 4:    end if
 5:    if j > 0 and i > 0 and GB[i][j] = GB[i - 1][j - 1] then
 6:       Increment p
 7:    end if
 8: end if
 9: if i > 0 then
10:    if GB[i][j] = GB[i - 1][j] then
11:       if i > 1 and GB[i - 2][j] ≠ 0 then
12:          if i > 2 and GB[i][j] = GB[i - 3][j] then
13:             Increment p
14:          else
15             if i < 18 and GB[i][j] = GB[i + 2][j] then
16:               Increment p
17:            end if
18:         end if
19:      else
20:         if i < 18 and GB[i][j] = GB[i + 2][j] then
21:            Increment p
22:         end if
23:      end if
24:   end if
25: end if
26: if i > 0 and j < 9 then
27:    if GB[i][j] = GB[i - 1][j + 1] then
28:       if i > 2 and j < 8 and GB[i][j] = GB[i - 3][j + 2] then
29:          Increment p
30:       end if
31:    end if
32: end if
33: if i > 0 and j > 0 then
34:    if GB[i][j] = GB[i - 1][j - 1] then
35:       if i > 2 and j > 1 and GB[i][j] = GB[i - 3][j - 2] then
36:          Increment p
33:       end if
34:    endif
35: endif
```

Kode Sumber 3.4 Pseudocode fitur *possible dissolve*

3.3.5. Desain Fitur Max Well Depth

Max well depth adalah desain dari fitur *max well depth* yang berfungsi untuk menghitung sumur terdalam yang terbentuk dari susunan kotak. Bagian ini menghitung kedalaman maksimal dari sumur – sumur yang terbentuk. Desain pseudocode dari *max well depth* dapat dilihat pada Kode Sumber 3.5.

```
 1: for i = 0 to 9 do
 2:   Set f to 0
 3:   Set h to 0
 4:   for j = 19 to 0 do
 5:     if GB[i][j] = 0 then
 6:       if((i > 0 and GB[j][i - 1] ≠ 0) or i = 0) then
 7:         if((i < 9 and GB[j] [i + 1] ≠ 0) or i = 9) then
 8:           if f ≠ 1 then
 9:             Set f to 1
10:           end if
11:             Increment h
12:         else
13:           if f = 1 then
14:             Break loop
15:           end if
16:         end if
17:       else
18:         if f = 1 then
19:           Break loop
20:         end if
21:       end if
22:     end if
23:   end for
24:   if f = 1 and h > w then
25:     Set w to h
26:   end if
27: end for
```

Kode Sumber 3.5 Pseudocode dari *max well depth*

3.3.6. Desain Fitur Blocks

Blocks adalah desain dari fitur *block* yang berfungsi untuk menghitung banyaknya kotak yang telah ditempati pada papan permainan. Desain pseudocode dari *blocks* dapat dilihat pada.

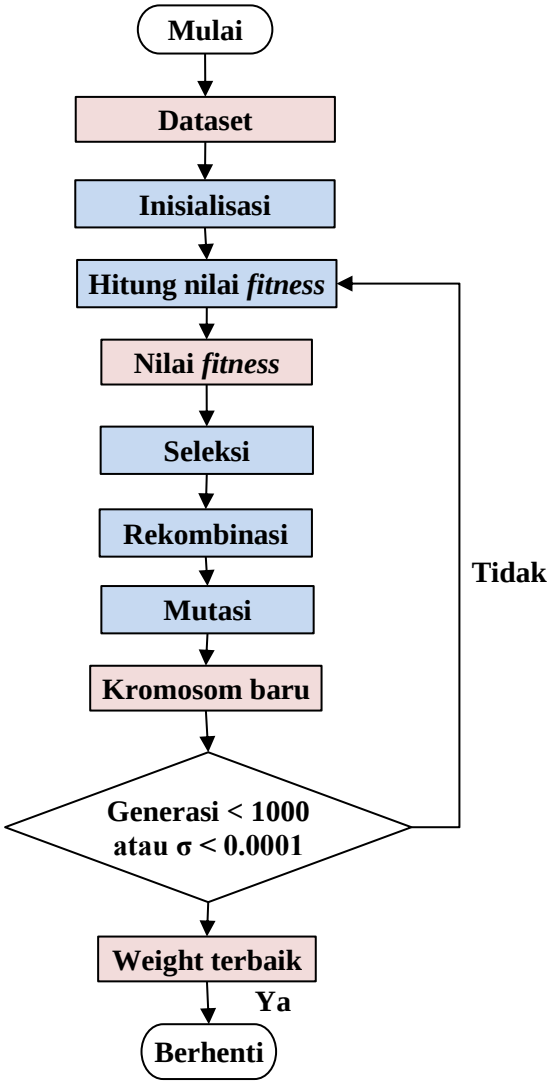
```
1: f ← CountRemainingPlace
2: place ← 200 - f
```

Kode Sumber 3.6 Pseudocode dari *blocks*

3.4. Desain Algoritma Genetika

Sesuai dengan subbab 3.1, dimana fitur yang akan digunakan telah disebutkan pada subbab 2.5.1, sehingga langkah selanjutnya adalah genetic algorithm. Desain fungsi algoritma genetika ini adalah penerapan dari subbab 2.2. Dimana terdapat tiga operasi utama yaitu seleksi, rekombinasi dan mutasi.

Alur berjalannya algoritma genetika adalah algoritma genetika menerima dataset yang berupa kumpulan masukan *brick* sebagai data untuk melatih weight. Kemudian dilakukan perulangan hingga kondisi terpenuhi yaitu standar deviasi dari kromosom lebi rendah dari 0,0001. Didalam perulangan dengan batas maksimum 1000 generasi dilakukan perhitungan *nilai fitness*. Nilai *fitness* yang digunakan adalah skor akhir dari permainan mengacu pada subbab 2.5.3. Selanjutnya dengan nilai *fitness* dilakukan seleksi pada kromosom dengan menggunakan *elitism* dan *rank selection*. Dari hasil seleksi dilakukan rekombinasi/crossover menggunakan nilai alpha dinamis antara 0,25 hingga 0,5 (subbab 2.2.3.1). Setelah dilakukan rekombinasi maka dilakukan mutasi dengan besar standar deviasi untuk mutasi yaitu 200. Hasil akhir dari adalah weight terbaik. Gambaran dari alur berjalannya algoritma genetika dapat dilihat dari pada Gambar 3.2.



Gambar 3.2 Alur kerja algoritma genetika

3.4.1. Desain Fungsi GeneticAlgo

Fungsi ini merupakan badan dari penerapan algoritma genetika dimana fungsi ini memanggil *fitness function*, seleksi, rekombinasi, dan mutasi. Desain pseudocode dari penerapan genetic algorithm dapat dilihat pada *Kode Sumber 3.7*.

```
1: for i to TK do
2:    Run hitung nilai fitness
3: end for
4: Run seleksi
5: for i = il to TK do
6:    p1 = randomize from prob vector
7:    p2 = randomize from prob vector
8:    Run function rekombinasi with p1, p2, and i
9:    Run function mutasi with σ and i
10: end for
11: change α value
```

Kode Sumber 3.7 Pseudocode badan dari algoritma genetika

3.4.2. Desain Fungsi Inisialisasi

Inisialisasi adalah fungsi yang menginisialisasi kromosom pada populasi awal dan dataset. Inisiasi kromsom awal menggunakan *normal distribution* dengan mean = 0. Dimana desain dari fungsi ini hampir sama dengan desain mutasi.

```
1: Set dataset
2: for j to 200 do
3:  for i to 6 do
4:     K[j][0][i] ← σ × N(0, 1)
5:  end for
6: end for
```

Kode Sumber 3.8 Pseudocode fungsi inisialisasi

3.4.3. Desain Hitung Nilai Fitness

Perhitung nilai *fitness* dilakukan dengan memainkan permainan dengan *testcase* yang telah ditentukan. Dimana nilai *fitness* yang didapatkan adalah hasil skor permainan, sehingga desain dari fungsi ini menyerupai subbab 3.5. Untuk rumus perhitungan skor dapat dilihat pada subbab 2.1.3 pada persaman (1) dan (2).

3.4.4. Desain Seleksi

Pada bagian ini akan dijelaskan tentang desain dari operasi seleksi yang rancangannya mengacu pada subbab 2.2.2. Dimana operasi seleksi ini mengaplikasikan 2 jenis seleksi yaitu, elitism dan rank selection. Desain pseudocode dari penerapan operasi seleksi dapat dilihat pada Kode Sumber 3.9.

```
1: sort K
2: for i to TK do
3:  if K[i][1] < FP then
4:     il ← i
5:     Increment sil
6:  end if
7: end for
8: for i to sil do
9:  Set prob vector
10: end for
```

Kode Sumber 3.9 Pseudocode operasi seleksi

3.4.5. Desain Fungsi Rekombinasi

Pada bagian ini akan dijelaskan tentang desain dari fungsi rekombinasi. Dimana fungsi rekombinasi ini mengaplikasikan arithmetic recombination yang mengacu pada subbab 2.2.3.1, dengan nilai awal $\alpha = 0.25$. Desain pseudocode dari penerapan operasi rekombinasi dapat dilihat pada Kode Sumber 3.10.

```
1: Input: parent p1, parent p2, and index j
2: for i to 6 do
3:     K[j][0][i]  = p1 x α + p2 x (1 - α)
4:     K[j + 1][0][i] = p2 x α + p1 x (1 - α)
5: end for
```

Kode Sumber 3.10 Pseudocode operasi rekombinasi

3.4.6. Desain Fungsi Mutasi

Pada bagian ini akan dijelaskan tentang desain dari fungsi mutasi. Dimana fungsi mutasi ini menggunakan distribusi normal untuk mendapatkan nilai acak mengacu pada subbab 2.2.4.1. Desain pseudocode dari penerapan operasi mutasi dapat dilihat pada Kode Sumber 3.11.

```
1: Input: index j
2: for i to 6 do
3:  p ← randomize
4:  if p < 31 then
5:     K[j][0][i] ← K[j][0][i] + σ × N(0, 1)
6:  end if
7: end for
```

Kode Sumber 3.11 Pseudocode operasi mutasi

3.4.7. Desain Kondisi Konvergen

Untuk mennetukan kondisi konvergen dari genetic algorithm yang mengacu pada subbab 2.2.5. Maka diperlukan mendapatkan standar deviasi dari kromosom unggulan dimana perlu terlebih dahulu dicari nilai mean dari kromosom unggulan. Sehingga diperlukan fungsi mean dan fungsi standar deviasi.

3.4.7.1. Desain Fungsi Mean

Pada bagian ini akan dijelaskan tentang desain dari fungsi mean. Dimana fungsi ini mencari rata-rata dari kromosom unggulan. Desain pseudocode dari fungsi mean dapat dilihat pada Kode Sumber 3.12.

```
1: Input: index s and index e
2: Output: mean m
2: Set m to 0
3: for i = s to e do
4:  m ← m + K[j][0][i]
5: end for
6: m ← m ÷ (e - s)
```

Kode Sumber 3.12 Pseudocode fungsi mean

3.4.7.2. Desain Fungsi Standar Deviasi

Pada bagian ini akan dijelaskan tentang desain dari fungsi standar deviasi. Dimana fungsi ini mencari standar deviasi dari kromosom unggulan. Desain pseudocode dari fungsi standar deviasi dapat dilihat pada Kode Sumber 3.13.

1: Input: index s and index e
2: Output: standard deviation sd
2: Set sd to 0
3: **for** i = s **to** e **do**
4: $sd \leftarrow sd + (x_i - \bar{x})^2$
5: **end for**
6: $sd \leftarrow \sqrt{\frac{sd}{(e-s)-1}}$

Kode Sumber 3.13 Pseudocode fungsi standar deviasi

3.4.8. Desain Fungsi Utama pada Algoritma Genetika

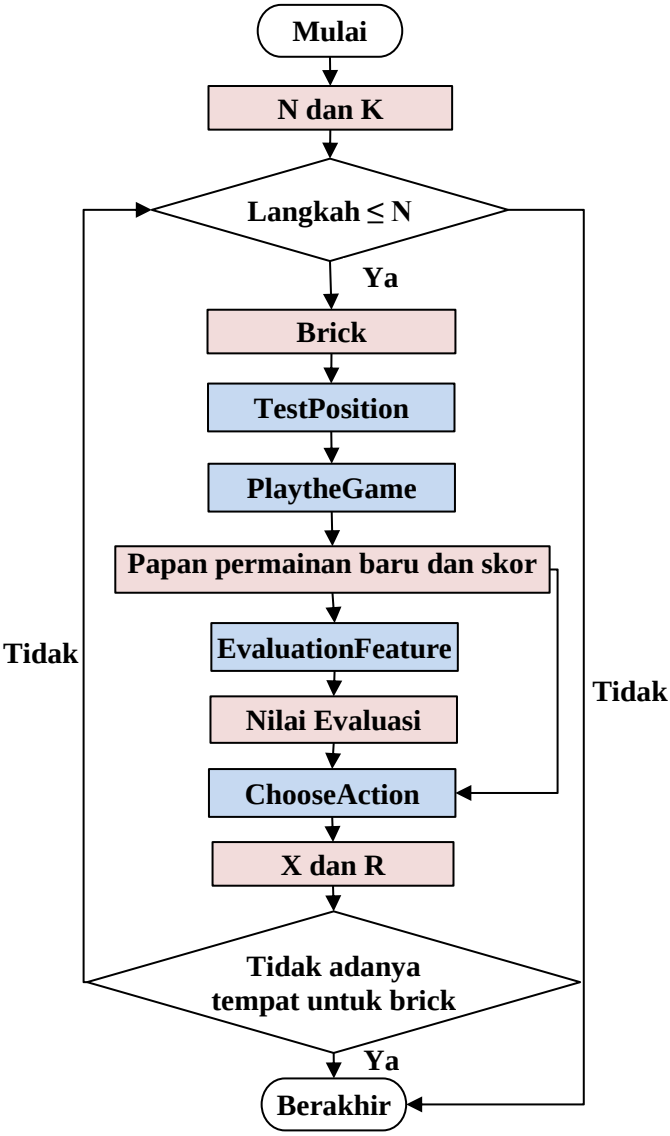
Pada bagian ini dijelaskan tentang fungsi utama pada algoritma genetika. Dimana fungsi utama ini berfungsi untuk mendapatkan masukan, inisiasi variabel yang digunakan, menjalankan perulangan pemanggilan fungsi algoritma genetika dan menentukan apakah kondisi batas populasi atau kondisi konvergen terpenuhi. Desain pseudocode dari fungsi utama pada algoritma genetika dapat dilihat pada Kode Sumber 3.14.

```
1: Get TB
2: Get TC
3: Run Inisialisasi
4: alpha ← 0.25
5: for i = 0 to batas populasi do
6:     Run function GeneticAlgo
7:     m ← mean(1, 10)
8:     sd ← standar deviasi(m, 1, 10)
9:     if sd < 0.0001 then
10:        break
11:    end if
12: end for
```

Kode Sumber 3.14 Pseudocode fungsi utama pada algoritma genetika

3.5. Desain Simulasi Permainan

Pada subbab ini akan dijelaskan tentang desain program untuk penyelesaian permasalahan *colour brick game*. Gambaran dari alur berjalannya simulasi permainan dapat dilihat dari pada Gambar 3.3



Gambar 3.3 Alur kerja simulasi permainan

Dari Gambar 3.3, dapat dilihat bahwa program simulasi permainan pertama kali menerima masukan berupa *N* dan *K*, kemudian dilakukan perulangan sebanyak *N* kali. Pada perulangan, diterima masukan 3 buah *integer* yang merepresentasikan *brick*. Dari brick tersebut dicoba seluruh aksi yang dapat terjadi yaitu 10 posisi peletakan dan 3 kemungkinan perputaran. Semua kemungkinan aksi tersebut akan dicoba satu persatu pada fungsi TestPosition, dimana fungsi tersebut mengecek posisi peletakan memungkinkan atau tidak. Bila tidak akan dilanjutkan ke kemungkinan aksi selanjutnya.

Sedangkan bila memungkinkan maka kemungkinan aksi akan diaplikasikan pada *brick* dan papan permainan dengan PlaytheGame. Hasil dari PlaytheGame adalah papan permainan baru dan skor yang didapatkan. Hasil dari papan permainan tersebut akan dievaluasi dengan fungsi EvaluationFeature, dimana fungsi ini menghasilkan nilai evaluasi. Setelah seluruh kemungkinan aksi telah melalui fungsi EvaluationFeature, dari hasil skor atau hasil nilai evaluasi maksimal akan ditentukan aksi yang ditentukan pada fungsi ChooseAction. Aksi keluaran hasil fungsi ChooseAcrtion berupa integer *X* dan integer *R*.

3.5.1. Desain Fungsi TestPosition

Fungsi TestPosition adalah fungsi yang menjalankan percobaan tiap aksi pada brick. Desain pseudocode dari fungsi TestPosition dapat dilihat pada Kode Sumber 3.15.

```
 1: Input: position X, rotation R, gameboard GB, brick B
 2: Set f to 0
 3: for i = 19 to 2 do
 4:  if GB[i][X] = 0 then
 5:     Set f to 1
 6:     Break loop
 7:  end if
 8: end for
 9: if f = 0 then
10:  ER ← -9999999
11:  TS ← -1
12: else
13:  Run PlaytheGame with GB, X, R, B and i
14:  Run EvaluationFeature with GB
15: end if
```

Kode Sumber 3.15 Pseudocode fungsi TestPosition

3.5.2. Desain Fungsi PlaytheGame

Fungsi PlaytheGame adalah pengaplikasian fungsi pemetaan. Fungsi ini didesain mengacu fungsi pemetaan pada subbab 2.4.5. dan rumus perhitungan skor pada subbab 2.1.3. Fungsi ini dibagi menjadi 6 bagian yaitu badan utama, penempatan brick, cek horizontal, cek vertikal, cek diagonal kanan dan cek diagonal kiri. Desain pseudocode dari fungsi PlaytheGame dapat dilihat pada Kode Sumber 3.16.

```
1: Input: position X, rotation R, position Y, gameboard
   GB, brick B
2: Output: score result TS, modified gameboard GB
3: Run Penempatan Brick
4: Set f to 1 and s to 1
5: while f ≠ 0 do
6:   Set f to 0 and z to 0
7:   Set all CB value to 0
8:   for i = 19 to 0 do
9:     for j = 9 to 0 do
10:      if PP[i][j] = 0 then
11:        Increment z
12:      else
13:        Run Cek Horizontal
14:        Run Cek Vertikal
15:        Run Cek Diagonal Kanan
16:        Run Cek Diagonal Kiri
17:      end if
18:    end for
19:    if z = 10 then
20:      Break loop
21:    end if
22:  end for
23:  if f = 1 then
24:    Run function ApplyChange with CB and GB
25:    TS ← TS + r x l x s
26:    Increment s
27:  end if
28: end while
```

Kode Sumber 3.16 Pseudocode fungsi PlaytheGame

3.5.2.1. Desain Penempatan Brick

Penempatan brick adalah bagian dari fungsi PlaytheGame. Bagian ini berfungsi untuk melakukan aksi rotasi atau perputaran dan penempatan *brick* pada papan permainan. Desain pseudocode dari penempatan *brick* dapat dilihat pada Kode Sumber 3.17.

```
 1: if R = 0 then
 2:   GB[Y][X] ← B[2]
 3:   GB[Y - 1][X] ← B[1]
 4:   GB[Y - 2][X] ← B[0]
 5: else
 6:   if R = 1 then
 7:      GB[Y][X] ← B[1]
 8:      GB[Y - 1][X] ← B[0]
 9:      GB[Y - 2][X] ← B[2]
10:   else
11:      GB[h][X] ←B[0]
12:      GB[h][X] ← B[2]
13:      GB[h][X] ← B[1]
14:   end if
15: end if
```

Kode Sumber 3.17 Pseudocode penempatan *brick*

3.5.2.2. Desain Cek Horizontal

Cek horizontal adalah bagian dari fungsi PlaytheGame. Dimana berfungsi untuk mengecek ada atau tidaknya kotak yang bertetangga secara horizontal sama warna dengan panjang minimal 3 kotak untuk dilenyapkan. Dimana bila ditemukan kotak yang memenuhi kriteria tersebut, validation board CB akan diberikan penanda dengan koordinat i dan j yang sama dengan posisi kotak yang lenyap pada gameboard (papan permainan) GB. Kemudian skor lenyapnya kotak akan langsung dihitung pada variabel r. Desain pseudocode dari cek horizontal dapat dilihat pada Kode Sumber 3.18

```
1: if j < 8 and (j - 1 < 0 or GB[i][j] ≠ GB[i][j - 1]) then
2:  Set t to 1
3:   for o = j + 1 to 10 do
5:      if GB[i][j] ≠ GB[i][o] then
6:         Break loop
7:      else
8:         Increment t
9:      endif
10:   end for
11:   if t > 2 then
12:      for o = 0 to t do
13:         CB[i][j + o] ← x
14:      end for
15:      Increment l
16:      r ← r + t^2
17:      Set f to 1
18:   end if
19: end if
```

Kode Sumber 3.18 Pseudocode cek horizontal

3.5.2.3. Desain Cek Vertikal

Cek vertikal adalah bagian dari fungsi PlaytheGame. Dimana berfungsi untuk mengecek ada atau tidaknya kotak yang bertetangga secara vertikal sama warna dengan panjang minimal 3 kotak untuk dilenyapkan. Bila ditemukan yang memenuhi syarat, maka langkah selanjutnya akan dilakukan langkah sama dengan cek horizontal, yaitu menandai pada validation board CB. Secara garis besar bagian ini memiliki cara kerja yang sama dengan cek horizontal. Desain pseudocode dari cek vertikal dapat dilihat pada Kode Sumber 3.19.

```
 1: if i < 18 and (i - 1 < 0 or GB[i][j] ≠ GB[i -1][j]) then
 2:  Set t to 1
 3:   for o = i + 1 to 10 do
 5:      if GB[i][j] ≠ GB[o][j] then
 6:         Break loop
 7:      else
 8:         Increment t
 9:      endif
10:   end for
11:   if t > 2 then
12:      for o = 0 to t do
13:         CB[i + o][j] ← x
14:      end for
15:      Increment l
16:      r ← r + t²
17:      Set f to 1
18:   end if
19: end if
```

Kode Sumber 3.19 Pseudocode cek vertikal

3.5.2.4. Desain Cek Diagonal Kanan

Cek diagonal kanan adalah bagian dari fungsi PlaytheGame yang berfungsi untuk mengecek ada atau tidaknya kotak yang bertetangga sama warna diagonal kearah kanan dengan panjang minimal 3 kotak untuk dilenyapkan. Bila memenuhi syarat, maka langkah selanjutnya yang sama dengan cek lainnya. Desain pseudocode dari cek diagonal kanan dapat dilihat pada Kode Sumber 3.20

```
1: if i < 18 and j < 8 and (i - 1 < 0 or j - 1 < 0 or GB[i][j]
   ≠ GB[i - 1][j - 1]) then
2:   Set t to 1
2:   b ← j + 1
3:   for o = i + 1 to 19 do
4:     if b > 9 then
5:       Break loop
6:     end if
7:     if GB[i][j] ≠ GB[o][b] then
8:       Break loop
9:     else
10:      Increment t
11:    endif
12:    Increment b
 8:  end for
 9:  if t > 2 then
10:    for o = 0 to t do
11:      CB[i + o][j + o] ← x
12:    end for
13:    r ← t^2
14:    Increment l
15:    Set f to 1
16:  end if
17: end if
```

Kode Sumber 3.20 Pseudocode cek diagonal kanan

3.5.2.5. Desain Cek Diagonal Kiri

Cek diagonal kiri adalah bagian dari fungsi PlaytheGame yang berfungsi untuk mengecek ada atau tidaknya kotak yang bertetangga sama warna diagonal kearah kiri dengan panjang minimal 3 kotak untuk dilenyapkan. Bila memenuhi syarat, maka langkah selanjutnya yang sama dengan cek lainnya. Desain pseudocode dari cek diagonal kiri dapat dilihat pada Kode Sumber 3.21.

```
 1: if i < 18 and j > 1 and (i - 1 < 0 or j + 1 > 9 or
    GB[i][j] ≠ GB[i - 1][j + 1]) then
 2:   Set t to 1
 2:   b ← j - 1
 3:   for o = i + 1 to 19 do
 4:     if b < 0 then
 5:       Break loop
 6:     end if
 7:     if GB[i][j] ≠ GB[o][b] then
 8:       Break loop
 9:     else
10:       Increment t
11:     endif
12:     Decrement b
 8:   end for
 9:   if t > 2 then
10:     for o = 0 to t do
11:       CB[i + o][j - o] ← x
12:     end for
13:     r ← t²
14:     Increment l
15:     Set f to 1
16:   end if
17: end if
```

Kode Sumber 3.21 Pseudocode cek diagonal kiri

3.5.2.6. Desain Fungsi ApplyChange

Fungsi ApplyChange adalah fungsi untuk mengaplikasikan mekanisme kotak lenyap. Dimana pada fungsi PlaytheGame akan dicari kotak-kotak yang memenuhi syarat untuk lenyap pada cek horizontal, xheck dan menandainya pada validation board CB. Kemudian CB akan digunakan untuk penanda bagi kotak yang lenyap pada papan permainan GB. Kemudian nilai kotak yang berada di posisi atas kotak lenyap akan turun menggantikan posisi kotak lenyap. Desain pseudocode dari fungsi ApplyChange dapat dilihat pada *Kode Sumber 3.22*.

```
 1: Input: gameboard GB, validation board CB
 2: Output: modified gameboard GB
 3: for i = 0 to 19 do
 4:   for j = 0 to 9 do
 5:     if CB[i][j] = x then
 6:       for k = i to 1 do
 7:         GB[k][j] ← GB[k - 1][j]
 8:       end for
 9:       GB[k][j] ← 0
10:     end if
11:   end for
12: end for
```

Kode Sumber 3.22 Pseudocode fungsi ApplyChange

3.5.3. Desain Fungsi CountRemainingPlace

Fungsi CountRemainingPlace adalah fungsi untuk menghitung tempat tersisah atau kotak kosong pada papan permainan. Desain pseudocode dari fungsi CountRemainingPlace dapat dilihat pada Kode Sumber 3.23.

```
1: Input: gameboard GB
2: Output: remaining place rp
3: Set rp to 0
2: for i = 0 to 19 do
3:   for j = 0 to 9 do
4:     if GB[i][j] = 0then
5:        Increment rp
6:     end if
7:   end for
8: end for
```

Kode Sumber 3.23 Pseudocode fungsi CountRemainingPlace

3.5.4. Desain Fungsi EvaluationFeature

Fungsi EvaluationFeature adalah fungsi yang bertugas untuk mendapatkan nilai fitur dan melakukan perhitungan evaluasi. Fungsi ini mengacu pada fungsi evaluasi yang terdapat pada subbab 2.4.6. Untuk fitur yang digunakan mengacu pada subbab 2.5.1. Fungsi ini memanggil 6 fitur pada *colour brick game*. Desain pseudocode dari fungsi EvaluationFeature dapat dilihat pada Kode Sumber 3.24.

```
 1: Input: gameboard GB
 2: Output: evaluation result ER
 3: Set w to 0, r to 0 and f to 0
 4: Run Max Well Depth
 5: Set ph to 0 and pd to 0
 6: Set pv to 0 and p to 0
 7: for i = 19 to 0 do
 8:   Set z to 0
 9:   for j = 9 to 0 do
10:     if GB[i][j] = 0 then
11:       Increment z
12:     else
13:       Run Horizontal Pair
14:       Run Vertical Pair
15:       Run Diagonal Pair
16:       Run Possible Dissolve
17:     end if
18:   end for
19:   if z = 10 then
20:     Break loop
21:   end if
22: end for
23: Run Blocks
24: ER ← ww × w + wr × r + wh × ph + wd × pd +
    wv × pv + wp ×  p
```

Kode Sumber 3.24 Pseudocode fungsi EvaluationFeature

3.5.5. Desain Fungsi ChooseAction

Fungsi ChooseAction adalah fungsi keputusan yang mengacu pada subbab 2.5.4. Desain pseudocode dari fungsi ChooseAction dapat dilihat pada Kode Sumber 3.25.

```
 1: Input: brick B
 2: Set me to 0, ms to 0 and ci to 0
 3: for i to 30 do
 4:   PS[2] position X ← i mod 10
 5:   PS[2] rotation R ← Round up i div 10
 6:   Set PS[2] gameboard GB to PGB
 7:   Run function TestPosition with PS[2] position X,
      PS[2] rotation R, PS[2] gameboard GB and B
 8:   if ms < PS[2] score TS then
 9:     ms ← PS[2] score TS
11:     Copy PS[2] to PS[1]
12:   end if
13:   if me < PS[2] evaluation EV then
14:     me ← evaluation EV
15:     Copy PS[2] to PS[0]
16:   end if
17: end for
18: if rs < 3 or rp < 10 and ms > 0 then
19:   ci ← 1
20: else
21:   if ms > sl then
22:     ci ← 1
23:   else
24:     ci ← 0
25:   end if
26: end if
27: Set PGB to PS[ci] gameboard GB
28: Print PS[ci] position X
29: Print PS[ci] rotation R
```

Kode Sumber 3.25 Pseudocode fungsi ChooseAction

3.5.6. Desain Fungsi Utama pada Simulasi Permainan

Pada bagian ini akan dijelaskan tentang desain dari fungsi utama untuk penyelesaian permasalahan *colour brick game*. Desain pseudocode dari fungsi utama dapat dilihat pada Kode Sumber 3.26.

```
1: N ← testcase
2: K ← warna
3: Run function SetParameter with K
4: Set gameboard PGB to 0
4: for i < N do
5:     B ← brick
6:     rs ← N - i
7:     Run function ChooseAction with B
8: end for
```

Kode Sumber 3.26 Pseudocode fungsi utama pada simulasi permainan

[Halaman ini sengaja dikosongkan]

BAB IV
IMPLEMENTASI

4.1. Lingkungan Implementasi

Lingkungan implementasi dan pengembangan yang dilakukan adalah sebagai berikut:

1. Perangkat Keras:
   a. Processor Intel® Core™ i3-M350 CPU @ 2.27GHz
   b. Random Access Memory 4096MB
2. Perangkat Lunak:
   a. Sistem Operasi Windows 7 Home Premium
   b. IDE Dev-C++ 5.11
   c. Bahasa Pemrograman C++
   d. TDM-GCC 4.9.2

4.2. Implementasi Fitur

Implementasi fitur adalah implementasi dari fitur yang digunakan oleh *colour brick game*. Implementasi fitur ini mengacu pada rancangan pada subbab 3.3. Terdapat 5 implementasi fitur dikarenakan fitur possible dissolve diimplementasikan bersama tiga fitur lainnya yaitu fitur horizontal pair, vertical pair dan diagonal pair.

4.2.1. Implementasi Fitur Horizontal Pair

Pada bagian ini mengimplementasikan fitur *horizontal pair* sesuai dengan subbab 3.3.1. Implementasi dari horizontal pair dapat dilihat pada Kode Sumber 4.1

```
if(j < 9 && memory_state[2].board[i][j] ==
memory_state[2].board[i][j + 1]){

  pairhor_point += 1;

}
```

Kode Sumber 4.1 Implementasi *horizontal pair*

4.2.2. Implementasi Fitur Vertical Pair

Pada bagian ini mengimplementasikan fitur *vertical pair* sesuai dengan subbab 3.3.2. Implementasi dari fitur *vertical pair* dapat dilihat pada Kode Sumber 4.2.

```
if(j > 0 && memory_state[2].board[i][j] ==
memory_state[2].board[i - 1][j]){

  pairver_point += 1;

}
```

Kode Sumber 4.2 Implementasi fitur *vertical pair*

4.2.3. Implementasi Fitur Diagonal Pair

Pada bagian ini mengimplementasikan fitur *diagonal pair* sesuai dengan subbab 3.3.3. Implementasi dari fitur *diagonal pair* dapat dilihat pada Kode Sumber 4.3.

```
if(i > 0 && j < 9){

  if(memory_state[2].board[i][j] ==
  memory_state[2].board[i - 1][j + 1]){

    pairdiag_point += 1;

  }

}

if(i > 0 && j > 0){

  if(memory_state[2].board[i][j] ==
  memory_state[2].board[i - 1][j - 1]){

    pairdiag_point += 1;

  }

}
```

Kode Sumber 4.3 Implementasi fitur *diagonal pair*

4.2.4. Implementasi Fitur Possible Dissolve

Pada bagian ini mengimplementasikan fitur *diagonal pair* sesuai dengan subbab 3.3.4. Implementasi dari fitur *diagonal pair* dapat dilihat pada Kode Sumber 4.4..

```
if(j < 9 && memory_state[2].board[i][j] ==
memory_state[2].board[i][j + 1
  if(j < 8 && i > 0 && memory_state[2].board[i][j]
  == memory_state[2].board[i - 1][j + 2]){
    possible += 1;
  } if(j > 0 && i > 0 &&
  memory_state[2].board[i][j] ==
  memory_state[2].board[i - 1][j - 1]){
    possible += 1;
}} if(j > 0 && memory_state[2].board[i][j] ==
memory_state[2].board[i - 1][j]){
  if(i > 1 && (memory_state[2].board[i - 2][j] -
  '0') != 0)){
    if(i > 2 && memory_state[2].board[i][j] ==
    memory_state[2].board[i - 3][j]){
      possible += 1;
  }else if(i < 18 && memory_state[2].board[i][j]
   == memory_state[2].board[i + 2][j]){
    possible += 1;
  }} else if(i < 18 && memory_state[2].board[i][j]
   == memory_state[2].board[i + 2][j]){
    possible += 1;
}} if(i > 0 && j < 9){
  if(memory_state[2].board[i][j] ==
  memory_state[2].board[i - 1][j + 1]){
    if(i > 2 && j < 8 &&
    memory_state[2].board[i][j] ==
    memory_state[2].board[i - 3][j + 2]){
      possible += 1;
}}}
if(i > 0 && j > 0){
  if(memory_state[2].board[i][j] ==
  memory_state[2].board[i - 1][j - 1]){
    if(i > 2 && j > 1 &&
    memory_state[2].board[i][j] ==
    memory_state[2].board[i - 3][j - 2]){
      possible += 1;
}}}
```

Kode Sumber 4.4 Implementasi fitur *possible dissolve*

4.2.5. Implementasi Fitur Max Well Depth

Pada bagian ini mengimplementasikan fitur *max well depth* sesuai dengan subbab 3.3.5. Berikut ini implementasi dari *max well depth*:

```
int height_count = 0; maxwell = 0;
bool found; int i, j;

for(i = 0; i < 10; i++){
  found = false; height_count = 0;
  for(j = 19; j > -1; j--){
    if((memory_state[2].board[j][i] – '0')
    == 0){
      if((i > 0 && memory_state[2].board[j][i
      - 1] != '0') || i == 0)
        if((i < 9 &&
        memory_state[2].board[j][i + 1] !=
        '0') || i == 9){
          if(found) height_count += 1;
          else{
            found = true;
            height_count += 1;
          }
        }else if(found) break;
      else if(found) break;
    }
  }
  if(found && maxwell < height_count)
    maxwell = height_count;
}
```

Kode Sumber 4.5 Implementasi fitur *max well depth*

4.2.6. Implementasi Fitur Blocks

Pada bagian ini mengimplementasikan fitur *blocks* sesuai dengan subbab 3.3.6. Berikut ini implementasi dari *blocks*:

```
block = 200 -
CountRemainingPlace(memory_state[2].board);
```

Kode Sumber 4.6 Implementasi fitur blocks

4.3. Implementasi Algoritma Genetika

Pada bagian ini menjelaskan implementasi dari algoritma genetika sesuai dengan desain pada subbab 3.4. Dimana algoritma ini diimplementasikan untuk melatih weight dari fitur. Dalam pengimplementasian algoritma ini memiliki lima bagian yaitu hitung nilai *fitness*, seleksi, fungsi rekombinasi, fungsi mutasi, dan kondisi konvergen yang dibagi menjadi dua, fungsi mean dan fungsi standar deviasi. Kelima bagian tersebut berpartisipasi dalam berjalannya algoritma genetika. Terdapat satu fungsi utama dari program algoritma genetika yang berfungsi untuk menjalankan fungsi algoritma genetika.

4.3.1. Penggunaan Library pada Algoritma Genetika

Program ini menggunakan beberapa library yang ada pada C++. Pada library algorithm, program ini menggunakan fungsi sort yang tersedia pada library tersebut.

```
#include <iostream>
#include <fstream>
#include <array>
#include <algorithm>
#include <random>
#include <string>
#include <cmath>
#include <ctime>
using namespace std;
```

Kode Sumber 4.7 Deklarasi library pada algoritma genetika

4.3.2. Deklarasi Variabel pada Algoritma Genetika

Berikut deklarasi variabel yang digunakan pada program algoritma genetika:

```
class vector_weight{
  public:
    array<double, 6> weight;
    long long score;
};
class ar_result{
  public:
    array<double, 6> chromosome_one;
    array<double, 6> chromosome_two;
};
const int N = 200;
vector_weight chromosomes[N];
char allbrick[33333][3];
long long fitness_param;
bool alpha_up;
double alpha;
string filename = "Nama file";
default_random_engine gen;
ofstream resultfile;
```

Kode Sumber 4.8 Deklarasi variabel pada algoritma genetika

4.3.3. Implementasi fungsi GeneticAlgo

Bagian ini menjelaskan tentang implementasi badan dari algoritma genetika yang mengacu desain pada subab 3.4.1. Implementasi dari fungsi GeneticAlgo dapat dilihat pada Kode Sumber 4.9.

```
void GeneticAlgo (){
  int j, I; index_limit = -1;
  for(j = 0; j < N; j++){
    set_the_weight(j);
    chromosomes[j].score = game_run();
  }
  // Implementasi seleksi
  fitness_param = chromosomes[0].score;
  int randpar_x, randpar_y;
  ar_result result;

  for(i = index_limit; i < N; i += 2){
    randpar_x = prob[rand() % SIL];
    randpar_y = prob[rand() % SIL];
    result =
    recombination(chromosomes[randpar_x].weigh
    t, chromosomes[randpar_y].weight);
    chromosomes[i].weight =
    mutation(pair_sigma,
    result.chromosome_one);
    if(i + 1 < N) chromosomes[i + 1].weight =
    mutation(pair_sigma,
    result.chromosome_two);
  }
  if(alpha >= 0.5) alpha_up = false;
  else if(alpha <= 0.25) alpha_up = true;
  if(alpha_up) alpha += 0.05;
  else alpha -= 0.05;
}
```

Kode Sumber 4.9 Implementasi fungsi GeneticAlgo

4.3.4. Implementasi Fungsi Inisialisasi

Implementasi dari fungsi ini memiliki kesamaan sumber kode dengan fungsi mutasi sehingga implementasi dapat *Kode Sumber 4.10* dilihat pada dengan nilai kromosom awal adalah 0.

```
void init_parameter (){

  cin >> total_color;
  cin >> total_step;

  for(int i = 0; i < total_step; i++)
    cin >> allbrick[i][0] >> allbrick[i][1]
    >> allbrick[i][2];

  array<double, 6> sgm = {100, 100, 100,
  100, 100, 100};

  for(int i = 0; i < N; i++)
    for(int j = 0; j < 6; j++){

      normal_distribution<double>
      gauss_dist(0, 1);

      chromosomes[i].weight[j] = sgm [j] *
      gauss_dist(gen);

    }
}
```

Kode Sumber 4.10 Implementasi fungsi inisialisasi

4.3.5. Implementasi Hitung Nilai Fitness

Hitung nilai fitness diimplementasika pada fungsi game_run. Dimana implementasinya memiliki sumber kode yang sama dengan simulasi permainan.

4.3.6. Implementasi Seleksi

Implementasi seleksi dilakukan dengan mengacu pada subbab 3.4.4. Dimana seleksi ini mengaplikasikan 2 jenis seleksi. Implementasi dari fungsi GeneticAlgo dapat dilihat pada Kode Sumber 4.11.

```
sort(chromosomes, chromosomes + N,
best_score);
for(j = 0; j < N; j++)
  if(chromosomes[j].score < fitness_param){
    index_limit = j;
    break;
  }
if(index_limit < 10) index_limit = 10;
else if(index_limit > N/10)
  index_limit = N/10;
int SIL, cp = 0;
for(i = 0; i < index_limit; ++i)
  SIL += i;
int prob[SIL]; j = 0;
for (i = 0; i < SIL; ++i){
  prob[i] = j; cp += 1;
  if(cp == index_limit - j){
    j += 1; cp = 0;
  }
}
```

Kode Sumber 4.11 Implementasi seleksi

4.3.7. Implemetasi Fungsi Rekombinasi

Pada bagian ini akan menjelaskan implementasi dari fungsi rekombinasi. Dalam implementasinya mengacu pada desain dari subbab 3.4.5. Fungsi ini pada implementasi diberikan nama recombination.

```
ar_result recombination(array<double, 6>
chromosome_one, array<double, 6>
chromosome_two){

  ar_result result;

  for(int i = 0; i < 6; i++){
     result.new_chromosome[0][i] = (alpha *
     chromosome_one[i]) + ((1 - alpha) *
     chromosome_two[i]);

     result.new_chromosome[1][i] = (alpha *
     chromosome_two[i]) + ((1 - alpha) *
     chromosome_one[i]);

  }

  return result;

}
```

Kode Sumber 4.12 Implementasi fungsi rekombinasi

4.3.8. Implementasi Fungsi Mutasi

Pada bagian ini akan menjelaskan implementasi dari fungsi mutasi. Dalam implementasinya mengacu pada desain dari subbab 3.4.6. Fungsi ini pada implementasi diberikan nama mutation.

```
array<double, 6> mutation(array<double, 6>
data_sigma, array<double, 6>
past_chromosome){

  array<double, 6> new_chromosome;
  int prob;

  for(int i = 0; i < 6; i++){

    prob = (rand() % 100) + 1;

    if(prob < 31){
      normal_distribution<double>
      gauss_dist(0, 1);

      new_chromosome[i] += data_sigma[i] *
      gauss_dist(gen);
    }else{
        new_chromosome[i] =
        past_chromosome[i];
    }

  }

  return new_chromosome;

}
```

Kode Sumber 4.13 Implementasi fungsi mutasi

4.3.9. Implementasi Fungsi Mean

Implementasi fungsi mean ini mengacu pada desain dari subbab 3.4.7.1. Implementasi dari fungsi ini dapat dilihat pada Kode Sumber 4.14.

```
array<double, 6> mean(int start, int end){
  array<double, 6> result;
  for(int i = 0; i < 6; i++){
    result[i] = 0.0;
    for(int j = start; j < end; j++)
      result[i] += chromosomes[j].weight[i];
    result[i] /= end;}
  return result;
}
```

Kode Sumber 4.14 Implementasi fungsi mean

4.3.10. Implementasi Fungsi Standar Deviasi

Implementasi fungsi standar deviasi ini mengacu pada desain dari subbab 3.4.7.2. Implementasi dari fungsi ini dapat dilihat pada Kode Sumber 4.15.

```
array<double, 6> SD(int start, int end){
  array<double, 6> result;
  for(int i = 0; i < 6; i++){
    result[i] = 0.0;
    for(int j = start; j < end; j++)
      result[i] +=
      pow((chromosomes[j].weight[i] -
      data_mean[i]), 2);
    result[i] = sqrt(result[i]/(end - 1));}
  return result;
}
```

Kode Sumber 4.15 Implementasi fungsi standar deviasi

4.3.11. Implementasi Fungsi Utama pada Algoritma Genetika

Implementasi fungsi utama ini mengacu pada desain dari subbab 3.4.8. Implementasi dari fungsi ini dapat dilihat pada Kode Sumber 4.16.

```
int main(){
  cin >> total_color >> total_step;
  int end_flag;
  array<double, 6> pair_mean;
  array<double, 6> pair_sigma;
  init_parameter();
  alpha = 0.25;
  alpha_up = true;

  for(int i = 0; i < 1000; i++){
    end_flag = 0;
    same_score = 0;
    GeneticAlgo();
    pair_mean = mean(0, index_limit);
    pair_sigma = sigma(0, index_limit,
    pair_mean);
    for(int l = 0; l < 6; l++)
      if(pair_sigma[l] < 0.0001)
        end_flag += 1;
    if(end_flag == 6) break;
  }
}
```

Kode Sumber 4.16 Implementasi fungsi utama program GA

4.4. Implementasi Simulasi Permainan

Pada bagian ini menjelaskan implementasi program utama. Program ini digunakana untuk menyelesaikan permsalahan colour brick game.

4.4.1. Penggunaan Library pada Simulasi Permainan

Program ini menggunakan beberapa library yang ada pada C++. Pada library algorithm, program ini menggunakan fungsi sort yang tersedia pada library tersebut.

```
#include <iostream>
using namespace std;
```

Kode Sumber 4.17 Deklarasi library pada simulasi permainan

4.4.2. Deklarasi Variabel pada Simulasi Permainan

Berikut deklarasi variabel yang digunakan pada program simulasi permainan:

```
float weightHor, weightDiag, weightVer,
      weightPos, weightMaxWell, weightBlock;
bool goExplode = false;
int total_step, total_color, game_score;
int pairhor_point, pairdiag_point,
    pairver_point, possible, maxwell, block;
int step_counter, remaining_place,
    score_limit, remaining_step;
char brick[3], check_board[20][10];
char playing_board[20][10];
typedef struct state {
  char board[20][10];
  int point;
  int position;
  int direction;
  float weight_score;
} state;
state memory_state[3];
```

Kode Sumber 4.18 Deklarasi variabel pada simulasi permainan

4.4.3. Implementasi Fungsi TestPosition

Pada subbab ini akan mengimplementasikan fungsi TestPosition sesuai dengan subbab 3.5.1. Berikut implementasi dari fungsi TestPosition:

```
void TestPosition(int move){
  memory_state[2].direction = move < 10 ?
  0 : (move < 20 ? 1 : 2);
  memory_state[2].position = ((move + 1) %
  10) == 0 ? 9 : (((move + 1) % 10 ) - 1);
  memory_state[move].point = 0;
  int position = memory_state[2].position, i;

  bool found = false;
  for(i = 19; i > 1; i--){
  if(memory_state[2].board[i][position] ==
  '0'){
     found = true;
     break;
    }
  }

  if(!found){
    memory_state[2].point = -1;
    memory_state[2].weight_score = -999999;
  }else{
    play_the_game(memory_state[2].position,
    memory_state[2].direction, i);
    count_weight();
  }
}
```

Kode Sumber 4.19 Implementasi TestPosition

4.4.4. Implementasi Fungsi PlaytheGame

Pada bagian ini akan menjelaskan tentang fungsi PlaytheGame pada program utama. Fungsi ini diimplementasi sesuai rancangan yang telah ada pada subbab 3.5.2. Berikut implementasi dari fungsi PlaytheGame:

```
void PlaytheGame(int position, int direction,
int placing_height){
  // bagian penempatan brick
  int step = 1, reward = 0, line = 0,
  zero_counter = 0, temp_val = 0;
  bool eroded_found = true;
  int i, j, o, b;
  while(eroded_found){
    eroded_found = false; reward = 0; line = 0;
    for(i = 0; i < 20; ++i)
      for(j = 0; j < 10; ++j)
        check_board[i][j] = '0';
    for(i = 19; i > -1; --i){
      zero_counter = 0;
      for(j = 9; j > -1; --j){
        if((memory_state[2].board[i][j] - '0')
        == 0) zero_counter += 1;
        else{
          // bagian pengecekan
        }
      if(zero_counter == 10) break;
    }
    if(eroded_found){
      ApplyChange(check_board);
      memory_state[2].point += reward * line *
      step;
      step += 1;
    } else break;
  }
}
```

Kode Sumber 4.20 Implementasi PlaytheGame

4.4.4.1. Implementasi Penempatan Brick

Pada bagian ini mengimplementasikan rancangan peletakan brick pada papan permainan sesuai dengan subbab 3.5.2.1. Berikut ini implementasi dari penempatan brick:

```
if(direction == 0){

  memory_state[2].board[placing_height][pos
  ition] = brick[2];
  memory_state[2].board[placing_height -
  1][position] = brick[1];
  memory_state[2].board[placing_height -
  2][position] = brick[0];

}else if(direction == 1){

  memory_state[2].board[placing_height][pos
  ition] = brick[1];
  memory_state[2].board[placing_height -
  1][position] = brick[0];
  memory_state[2].board[placing_height -
  2][position] = brick[2];

}else{

  memory_state[2].board[placing_height][pos
  ition] = brick[0];
  memory_state[2].board[placing_height -
  1][position] = brick[2];
  memory_state[2].board[placing_height -
  2][position] = brick[1];
}
```

Kode Sumber 4.21 Implementasi penempatan brick

4.4.4.2. Implementasi Cek Horizontal

Pada bagian ini mengimplementasikan pengecekan secara horizontal sesuai dengan subbab 3.5.2.2. Bagian ini terletak pada fungsi PlaytheGame posisi yang berisi komentar 'bagian pengecekan'. Berikut ini implementasi dari cek horizontal:

```
if(j < 8 && (j - 1 < 0 ||
memory_state[2].board[i][j] !=
memory_state[2].board[i][j - 1])){

temp_val = 1;
 for(o = j + 1; o < 10; ++o){

   if(memory_state[2].board[i][j] !=
   memory_state[2]. board[i][o])
     break;
   else temp_val += 1;

 }

 if(temp_val > 2){

   for(b = 0; b < temp_val; ++b)
     check_board[i][j + b] = 'x';
   line += 1;
   eroded_found = true;
   reward += pow(temp_val, 2);

 }
}
```

Kode Sumber 4.22 Implementasi cek horizontal

4.4.4.3. Implementasi Cek Vertikal

Pada bagian ini mengimplementasikan pengecekan secara vertikal sesuai dengan subbab 3.5.2.3. Bagian ini terletak pada fungsi PlaytheGame posisi yang berisi komentar 'bagian pengecekan'. Berikut adalah implementasi dari cek vertikal:

```
if(i < 18 && (i - 1 < 0 ||
memory_state[2].board[i][j] !=
memory_state[2].board[i - 1][j])){

  temp_val = 1;
  for(o = i + 1; o < 20; ++o){

    if(memory_state[2].board[i][j] !=
    memory_state[2]. board[o][j])
      break;
    else temp_val += 1;

  }

  if(temp_val > 2){

    for(b = 0; b < temp_val; ++b)
      check_board[i + b][j] = 'x';
    line += 1;
    eroded_found = true;
    reward += pow(temp_val, 2);

  }
}
```

Kode Sumber 4.23 Implementasi cek vertikal

4.4.4.4. Implementasi Cek Diagonal Kanan

Pada bagian ini mengimplementasikan pengecekan secara diagonal kearah kanan sesuai dengan subbab 3.5.2.4. Bagian ini terletak pada fungsi PlaytheGame posisi yang berisi komentar 'bagian pengecekan'. Berikut adalah implementasi dari cek diagonal kanan:

```
if(i < 18 && j < 8 && (i - 1 < 0 || j - 1 <
0 || memory_state[2].board[i][j]
!= memory_state[2].board[i - 1][j - 1])){
 temp_val = 1;
 b = j + 1;

 for(o = i + 1; o < 20; ++o){
  if(b > 9) break;
  else if(memory_state[2].board[i][j] !=
  memory_state[2]. board[o][b])
   break;
  else temp_val += 1;
  b += 1;
 }

 if(temp_val > 2){
  for(b = 0; b < temp_val; ++b)
   check_board[i + b][j + b] = 'x';
  line += 1;
  eroded_found = true;
  reward += pow(temp_val, 2);
 }
}
```

Kode Sumber 4.24 Implementasi cek diagonal kanan

4.4.4.5. Implementasi Cek Diagonal Kiri

Pada bagian ini mengimplementasikan pengecekan secara diagonal kearah kiri sesuai dengan subbab 3.5.2.5. Bagian ini terletak pada fungsi PlaytheGame posisi yang berisi komentar 'bagian pengecekan'. Berikut adalah implementasi dari cek diagonal kiri:

```
if(i < 18 && j > 1 && (i - 1 < 0 || j + 1 >
9 || memory_state[2].board[i][j] !=
memory_state[2].board[i - 1][j + 1])){
  temp_val = 1;
  b = j - 1;

  for(o = i + 1; o < 20; ++o){
    if(b < 0) break;
    else if(memory_state[2].board[i][j] !=
    memory_state[2]. board[o][b])
      break;
    else temp_val += 1;
    b -= 1;
  }

  if(temp_val > 2){

    for(b = 0; b < temp_val; ++b)
      check_board[i + b][j - b] = 'x';

    line += 1;
    eroded_found = true;
    reward += pow(temp_val, 2);
  }
}
```

Kode Sumber 4.25 Implementasi cek diagonal kiri

4.4.4.6. Implementasi Fungsi ApplyChange

Pada bagian ini akan menjelaskan implementasi pada fungsi ApplyChange. Pengimplementasian fungsi ini mengacu pada subbab 3.5.2.6. Berikut ini implementasi dari fungsi ApplyChange:

```
void ApplyChange(char board[20][10], int
state_position){
  int i, j, k;

  for(i = 0; i < 20; ++i){
    for(j = 0; j < 10; ++j){
      if(board[i][j] == 'x'){
        for(k = i; k > 0; --k){

          memory_state[2].board[k][j] =
          memory_state[2].board[k - 1][j];

        }

         memory_state[2].board[0][j] = '0';
      }
    }
  }
}
```

Kode Sumber 4.26 Implementasi ApplyChange

4.4.5. Implementasi Fungsi EvaluationFeature

Pada bagian ini akan menjelaskan tentang fungsi EvaluationFeature pada program utama. Fungsi ini diimplementasi sesuai rancangan yang telah ada pada subbab 3.5.4. Berikut implementasi dari fungsi EvaluationFeature:

```
void EvaluationFeature (){
  // Max well depth
  int zero_counter;
  pairhor_point = 0; pairdiag_point = 0;
  pairver_point = 0; possible = 0;
  for(i = 19; i > -1; i--){
    zero_counter = 0;
    for(j = 9; j > -1; j--){
      if((memory_state[2].board[i][j] -
      '0') != 0){
        // bagian Horizontal pair
        // bagian Vertical pair
        // bagian Diagonal pair
        // bagian Possible dissolve
      }else zero_counter += 1;
    }
    if(zero_counter == 10) break;
  }
  // bagian Blocks
  memory_state[2].weight_score =
  weightMaxWell * maxwell + weightBlock *
  block + weightHor * pairhor_point +
  weightDiag * pairdiag_point + weightVer *
  pairver_point + weightPos * possible;
}
```

Kode Sumber 4.27 Implementasi fungsi EvaluationFeature

4.4.6. Implementasi Fungsi CountRemainingPlace

Pada subbab ini akan mengimplementasikan fungsi CountRemainingPlace sesuai dengan subbab 3.5.3. Berikut implementasi dari fungsi CountRemainingPlace:

```
int CountRemaingingPlace(char
board[20][10]){
  remaining_place = 0;
  for(int j = 0; j < 20; j++)
    for(int k = 0; k < 10; k++)
      if((board [j][k] - '0') == 0)
      remaining_place += 1;
      return remaining_place;
}
```

Kode Sumber 4.28 Implementasi CountRemainingPlace

4.4.7. Implementasi Fungsi ChooseAction

Pada subbab ini akan mengimplementasikan fungsi ChangeAction sesuai dengan subbab 3.5.5. Berikut implementasi dari fungsi ChooseAction:

```
bool ChooseAction(){
  int max_eval = -999999, max_score = -1, ci
  = 0;
  for(int i = 0; i < 30; i++){
    memory_state[2].point = 0;
    memory_state[2].weight_score = 0;
    memory_state[2].direction = 0;
    memory_state[2].position = 0;
    // Bagian mengecek posisi
  }
  if((remaining_step < 3 || remaining_place
  < 10) && max_score > 0) ci = 0;
  else if(max_score > score_limit) ci = 0;
  else ci = 1;
  // Bagian menampilkan hasil
}
```

Kode Sumber 4.29 Implementasi fungsi ChooseAction

Berikut ini adalah implementasi bagian mengecek posisi dari tiap kemungkinan aksi dari ChooseAction.

```
for(int j = 0; j < 20; j++)
  for(int k = 0; k < 10; k++)
    memory_state[i].board[j][k] =
    playing_board[j][k];
    if(max_score < memory_state[2].point){
      max_score = memory_state[2].point;
      copy_memory_state(0, 2); }
    if(max_eval <
    memory_state[2] .weight_score){
      max_eval = memory_state[2].weight_score;
      copy_memory_state(1, 2);
    }
```

Kode Sumber 4.30 Implementasi mengecheck posisi

Berikut ini adalah implementasi bagian menampilkan hasil dari ChooseAction.

```
if(memory_state[ci].point < 0)
  return true;
else{
  for(int j = 0; j < 20; j++)
    for(int k = 0; k < 10; k++)
      playing_board[j][k] =
      memory_state[ci].board[j][k];
  cout << memory_state[ci].position + 1 << " "
    << memory_state[ci].direction << "\n";
  game_score += memory_state[ci].point;
  remaining_place =
  CountRemainingPlace(playing_board);
  return false;
}
```

Kode Sumber 4.31 Implementasi menampilkan hasil

4.4.8. Implementasi Fungsi Utama pada Simulasi Permainan

Pada subbab ini akan mengimplementasikan fungsi utama dari sesuai dengan subbab 3.5.6. Berikut implementasi dari fungsi CountRemainingPlace:

```
int main(){
  int step, position = 1, rotation = 0;
  bool gameOver;
  ios::sync_with_stdio(false);
  cin >> total_step >> total_color;
  set_variable();
  for(int i = 0; i < total_step; ++i){
    remaining_step = total_step - (i + 1);
    cin >> brick[0] >> brick[1] >> brick
[2];
    remaining_step = total_step - i;
    gameOver = ChooseAction();
    step_counter += 1;
    if(gameOver) break;
  }
}
```

Kode Sumber 4.32 Implementasi fungsi utama

[Halaman ini sengaja dikosongkan]

BAB V
UJI COBA DAN HASIL

Bab ini berisi tentang penjelasan hasil uji coba dan pembahasan solusi dari permasalahan SPOJ 18073 Colour Brick Game.

5.1. Lingkungan Uji Coba

Uji coba dilakukan adalah menggunakan perankat dengan spesifikasi sebagai berikut:

1. Perangkat Keras:
   a. Processor Intel® Core™ i3-M350 CPU @ 2.27GHz
   b. Random Access Memory 4096MB
2. Perangkat Lunak:
   1. Sistem Operasi Windows 7 Home Premium

5.2. Skenario Uji Coba

Pada subbab ini dijelaskan skenario uji coba yang dilakukan pada pengerjaan program penyelesaian. Uji coba akan dilakukan terhadap beberapa parameter untuk menentukan parameter yang optimal. Uji coba dilakukan pada tiga daerah yaitu fitur, algoritma genetika, dan uji coba pada program penyelesaian itu sendiri.

Skenario uji coba yang dilakukan pada fitur adalah sebagai berikut:

1. Uji Coba Kombinasi Fitur

Berikut adalah skenario uji coba terhadap algoritma genetika untuk mendapatkan parameter yang tepat:

1. Uji Coba Populasi
2. Uji Coba Probabilitas Mutasi
3. Uji Coba Nilai σ

Untuk uji coba terhadap program penyelesaian dari permasalahan, terdapat 3 skenario uji coba yang diujikan pada online judge SPOJ yaitu:

1. Uji Coba Kebenaran
2. Uji Coba Optimasi Skor pada SPOJ
3. Uji Coba tiap Testcase

5.3. Uji Coba Kombinasi Fitur

Uji coba fitur dilakukan untuk menentukan kombinasi fitur terbaik untuk digunakan dalam fungsi evaluasi (subbab 2.5). Pada uji coba ini akan memadukan fitur hasil analisis dan fitur pada tetris. Fitur Tetris yaitu Connected Holes, Holes, Removed Lines tidak dapat digunakan karena tidak dapat diaplikasikan sedangkan Landing Height tidak berpengaruh pada penyusunan.

Uji coba ini dilakukan pada testcase dengan kondisi K = 9 dan N = 20000 dengan parameter pada algoritma genetika berupa populasi = 200, standar deviasi = 200 dan probabilitas mutasi = 0,3. Berikut hasil dari percobaan:

No	Kombinasi Fitur	Skor Tertinggi	Generasi Terhenti
1	HP + VP + DP + PD	6909	13
2	HP + VP + DP + PD + PH	10119	13
3	HP + VP + DP + PD + MW	9118	13
4	HP + VP + DP + PD + B	9585	18
5	HP + VP + DP + PD + PH + B	5484	11
6	HP + VP + DP + PD + PH + MW	7474	13
7	HP + VP + DP + PD + B + MW	13168	8

Tabel 5.1 Hasil percobaan kombinasi fitur

HP = Horizontal Pair
VP = Vertical Pair
DP = Diagonal Pair
PD = Possible Dissolve
MW = Maximum Well Depth
B = Block
PH = Pile Height

Dari hasil uji coba pada Tabel *5.1* dapat dilihat bahwa walau pada awal percobaan kombinasai, fitur pile height menghasilkan hasil tinggi tetapi fitur tersebut tidak dapat dikombinasikan dengan fitur lain dari tetris karena menurunya skor yang didapatkan. Sedangkan kombinasi dari fitur tetris lainnya menghasilkan skor lebih baik yaitu kombinasi fitur Maximum Well Depth dan Block pada no. 7, sehingga no. 7 adalah kombinasi fitur paling optimal.

5.4. Uji Coba pada Algortima Genetika

Uji coba yang dilakukan pada algoritma genetika memiliki tujuan untuk menentukan parameter yang tepat untuk digunakan dalam training beban.

5.4.1. Uji Coba Populasi

Uji coba populasi pada algoritma genetika bertujuan untuk menentukan besar populasi yang tepat untuk digunakan. Uji coba ini dilakukan pada testcase dengan kondisi K = 9 dan N = 20000 dengan parameter pada algoritma genetika berupa standar deviasi = 200 dan probabilitas mutasi = 0,3. Hasil uji coba dapat dilihat pada Tabel *5.2*.

No	Populasi	Skor Tertinggi	Generasi Terhenti
1	50	8327	24
2	100	8327	8
3	200	13168	8
4	300	11001	11

Tabel 5.2 Hasil uji coba populasi

Dari hasil pada Tabel *5.2* maka dapat dilihat nilai populasi yang membuat skor optimal adalah populasi bernilai 200.

5.4.2. Uji Coba Probabilitas Mutasi

Uji coba probabilitas mutasi pada algoritma genetika bertujuan untuk menentukan besar probabilitas terjadinya mutasi pada kromosom. Uji coba ini dilakukan pada testcase dengan kondisi K = 9 dan N = 20000 dengan parameter pada algoritma genetika berupa populasi = 200 dan standar deviasi = 200. Berikut hasil dari percobaan:

No	Probabilitas	Skor Tertinggi	Generasi Terhenti
1	0,05	9065	10
2	0,1	9629	9
3	0,2	10627	12
4	0,3	13168	8

Tabel 5.3 Hasil uji coba probabililtas mutasi

Dari hasil pada Tabel *5.3* maka dapat dilihat nilai probabilitas mutasi yang membuat skor optimal adalah probabilitas mutasi bernilai 0,3.

5.4.3. Uji Coba Nilai σ

Uji coba nilai σ (standar deviasi) pada algoritma genetika bertujuan untuk menentukan besar σ yang tepat untuk digunakan pada mutasi. Uji coba ini dilakukan pada testcase dengan kondisi K = 9 dan N = 20000 dengan parameter pada algoritma genetika berupa populasi = 200 dan probabilitas mutasi = 0,3. Berikut hasil dari percobaan:

No	σ	Skor Tertinggi	Generasi Terhenti
1	50	8184	9
2	100	10962	15
3	200	13168	8
4	300	12223	9

Tabel 5.4 Hasil uji coba nilai σ

Dari hasil pada Tabel *5.4* maka dapat dilihat *σ* mutasi yang membuat skor optimal adalah *σ* bernilai 200.

5.5. Uji Coba Kebenaran

Uji coba ini bertujuan untuk membuktikan bahwa implementasi solusi dapat berjalan pada online judge SPOJ. Berikut bukti bahwa implementasi solusi dapat berjalan:

Gambar 5.1 Hasil submit solusi pada SPOJ

Dari hasil yang mendapatkan warna hijau menandakan sumber kode dapat berjalan dengan baik dan kebenaran dari sumber kode terbukti.

5.6. Uji Coba Optimasi Skor pada SPOJ

Uji coba ini bertujuan untuk melihat performa dari implementasi solusi pada optimasi skor. Hasil uji coba dapat dilihat pada Gambar 5.2.

RANK	DATE	USER	RESULT	TIME	MEM	LANG
1	2018-07-20 04:50:45	Destiana	**24100379**	9.28	2.8M	C++ 4.3.2
2	2018-07-20 12:11:44	Rully Soelaiman	**24100379**	9.18	2.8M	C++ 4.3.2
3	2014-06-09 20:25:36	Tata Dule	**23041164**	16.00	17M	C++ 4.3.2
4	2014-05-25 21:51:38	Aleksandar Ogrizovic	**22790095**	15.80	17M	C++ 4.3.2
5	2014-05-25 20:40:52	Karol Konaszyński	**17037860**	27.37	3.0M	C++ 4.3.2
6	2014-05-25 16:38:19	Karol Marković	**15964448**	14.51	11M	C++ 4.3.2
7	2014-05-25 18:40:58	Dragan Marković	**15287744**	19.70	6.0M	C++ 4.3.2
8	2014-05-25 18:49:29	SaĹĄa VuÄ□koviÄ□	**15066028**	11.82	3.1M	C++ 4.3.2
9	2014-05-25 21:49:06	4DarkLady	**15055338**	11.56	3.1M	C++ 4.3.2
10	2014-05-23 20:34:41	Nikola SmiljkoviÄ□	**14386073**	21.47	5.5M	C++ 4.3.2

Gambar 5.2 Hasil ranking pada SPOJ 18073

Dari hasil ranking pada Gambar 5.2, penyelesaian yang digunakan pada Tugas Akhir ini paling optimal dalam optimasi skor. Dalam optimasi running time, penyelesaian tugas akhir ini paling optimal diantara ranking 3 – 10. Dalam optimasi memori, penyelesaian tugas akhir ini lebih optimal diantara penyelesaian yang menghasilkan skor diatas 20 juta.

5.7. Uji Coba Tiap Testcase

Uji coba ini memperlihatkan hasil dari tiap testcase pada SPOJ beserta weight yang digunakan. Dimana skor dan banyaknya move akan dibandingkan dengan hasil pada Microsoft Bubble Cup Problem Set Booklet 2014.

Berikut hasil Booklet 2014 untuk K = 3 dan K = 4:

	K = 3		K = 4	
	N = 3333	N = 33333	N = 2500	N = 25000
Correct Move	3333	33333	2498	25000
Score	1.333.534	11.997.210	646.033	6.168.573

Tabel 5.5 Hasil Booklet 2014 untuk K = 3 dan K = 4

Berikut hasil dan parameter dari solusi tugas akhir ini untuk K = 3 dan K = 4:

	K = 3		K = 4	
	N = 3333	N = 33333	N = 2500	N = 25000
Correct Move	3333	33333	2500	25000
Score	1.375.599	12.787.295	661.602	5.931.373
Nilai Parameter				
Horizontal Pair	40,1596	40,1596	107,047	107,047
Vertical Pair	-191,458	-191,458	-143,271	-143,271
Diagonal Pair	47,1979	47,1979	129,176	129,176
Possible Dissolve	65,654	65,654	116,011	116,011
Max Well	-47,1867	-47,1867	137,186	137,186
Block	0,777783	0,777783	-57,8758	-57,8758
Score Limit	17750	18500	8600	6750

Tabel 5.6 Hasil dan parameter implementasi solusi untuk K = 3 dan K = 4

Berikut hasil Booklet 2014 untuk K = 5 hingga K = 7:

	K = 5		K = 6	K = 7
	N = 2000	N = 20000	N = 16666	N = 14285
Correct Move	2000	20000	16664	14285
Score	204.065	1.685.123	521.511	215.244

Tabel 5.7 Hasil Booklet 2014 untuk K = 5 hingga K = 7

Berikut hasil dan parameter dari solusi tugas akhir ini untuk K = 5 hingga K = 7:

	K = 5		K = 6	K = 7
	N = 2000	N = 20000	N = 16666	N = 2500
Correct Move	2000	20000	16666	14285
Score	231.030	2.031.750	566.462	324.145
Nilai Parameter				
Horizontal Pair	153,296	153,296	106.149	301,569
Vertical Pair	-212,85	-212,85	-5,82133	-52,8532
Diagonal Pair	233,036	233,036	73,8042	275,494
Possible Dissolve	120,275	120,275	33,941	102,024
Max Well	175,776	175,776	105,196	265,767
Block	-219,832	-219,832	-106,672	-350,452
Score Limit	6250	4000	5500	2000

Tabel 5.8 Hasil dan parameter implementasi solusi untuk K = 5 hingga K = 7

Berikut hasil pada Booklet 2014 untuk K = 8 dan K = 9:

	K = 8	K = 9
	N = 12500	N = 11111
Correct Move	1007	370
Score	14.740	4.062

Tabel 5.9 Hasil Booklet 2014 untuk K = 8 dan K = 9

Berikut hasil dan parameter dari solusi tugas akhir ini untuk K = 8 dan K = 9:

	K = 8	K = 9
	N = 12500	N = 11111
Correct Move	12500	582
Score	183.980	7.143
Nilai Parameter		
Horizontal Pair	95,0478	247,712
Vertical Pair	80,8678	80,5575
Diagonal Pair	232,788	383,78
Possible Dissolve	206,558	234,719
Max Well	23,8948	26,2254
Block	-326,756	-488,369
Score Limit	100	50

Tabel 5.10 Hasil dan parameter implementasi solusi untuk K = 8 dan K = 9

Perbandingan secara keseluruhan antara solusi pada Booklet 2014 dan solusi pada tugas akhir ini:

RANK	DATE	USER	RESULT	TIME	MEM	LANG
1	2018-07-20 04:50:45	Destiana	**24100379**	9.28	2.8M	C++ 4.3.2
4	2014-05-25 21:51:38	Aleksandar Ogrizovic	**22790095**	15.80	17M	C++ 4.3.2

Gambar 5.3 Perbandingan menyeluruh solusi Booklet 2014 dan tugas akhir

Dari perbandingan tersebut dapat dilihat bahwa penyelesaian yang diusulkan pada tugas akhir ini memiliki hasil skor yang lebih optimal dari pada penyelesaian yang digunakan pada Booklet 2014. Dimana problem solver untuk permasalahan colour brick game pada Microsoft Bubble Cup Problem Set Booklet 2014 adalah Alexandar Ogrizovic dengan posisi 4 pada ranking SPOJ.

Melihat dari alur jalannya algoritma pada Booklet 2014, dapat dilihat bahwa terdapat beberapa kemungkinan yang mengakibatkan penyelesaian tugas akhir menghasilkan skor yang lebih optimal. Salah satu kemungkinan alasan yang mengakibatkan hasil skor dari solusi tugas akhir ini lebih optimal adalah kombinasi fitur yang digunakan lebih optimal sehingga mengakibat weight yang digunakan lebih optimal. Alasan mengapa waktu dan memori solusi pada tugas akhir ini lebih optimal adalah penyelesaian pada Booklet menyimpan pada array seluruh kondisi peletakan *brick* beserta skor dan kondisi papan dan menggunakannya array tersebut untuk pengambilan keputusan setelah kondisi terpenuhi sehingga mengakibatkan diperlukannya waktu dan memori yang besar.

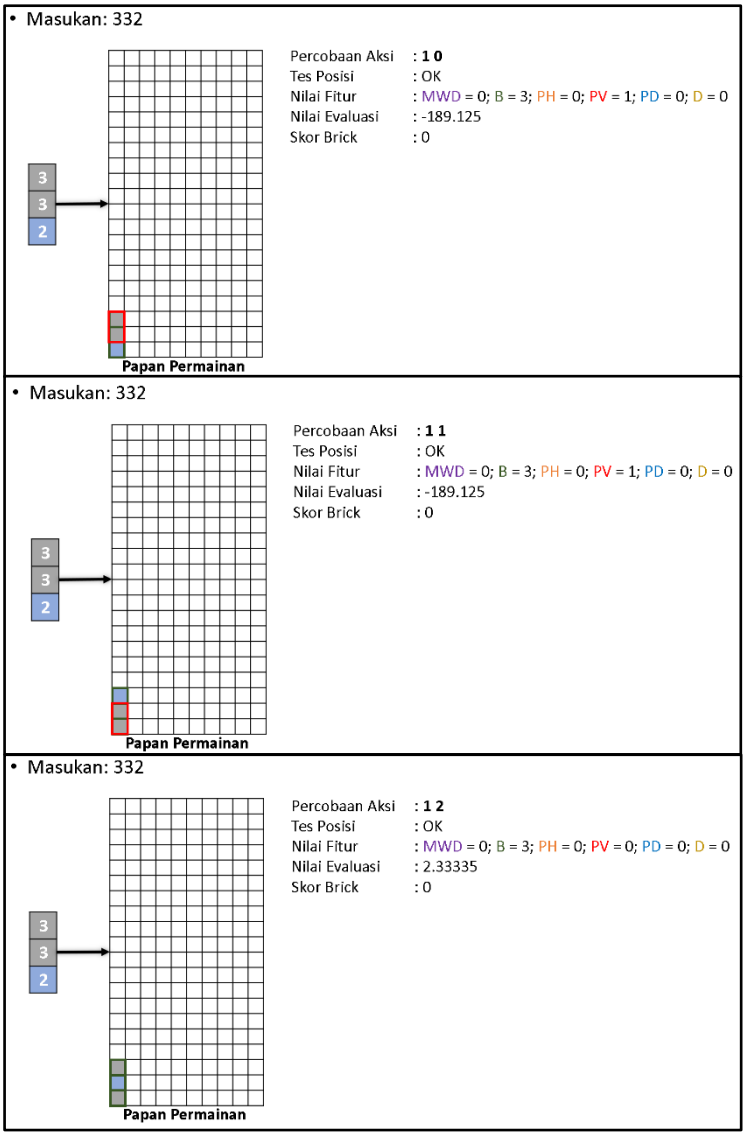
5.8. Ilustrasi Penyelesaian Colour Brick Game

Pada subbab ini akan diberikan ilustrasi penyelesain colour brick game dari tugas akhir ini. Pada ilustrasi ini nilai N = 10 dan K = 3 dengan masukan dan keluarana hasil penyelesaian tugas akhit ini dapat dilihat pada Tabel 5.11.

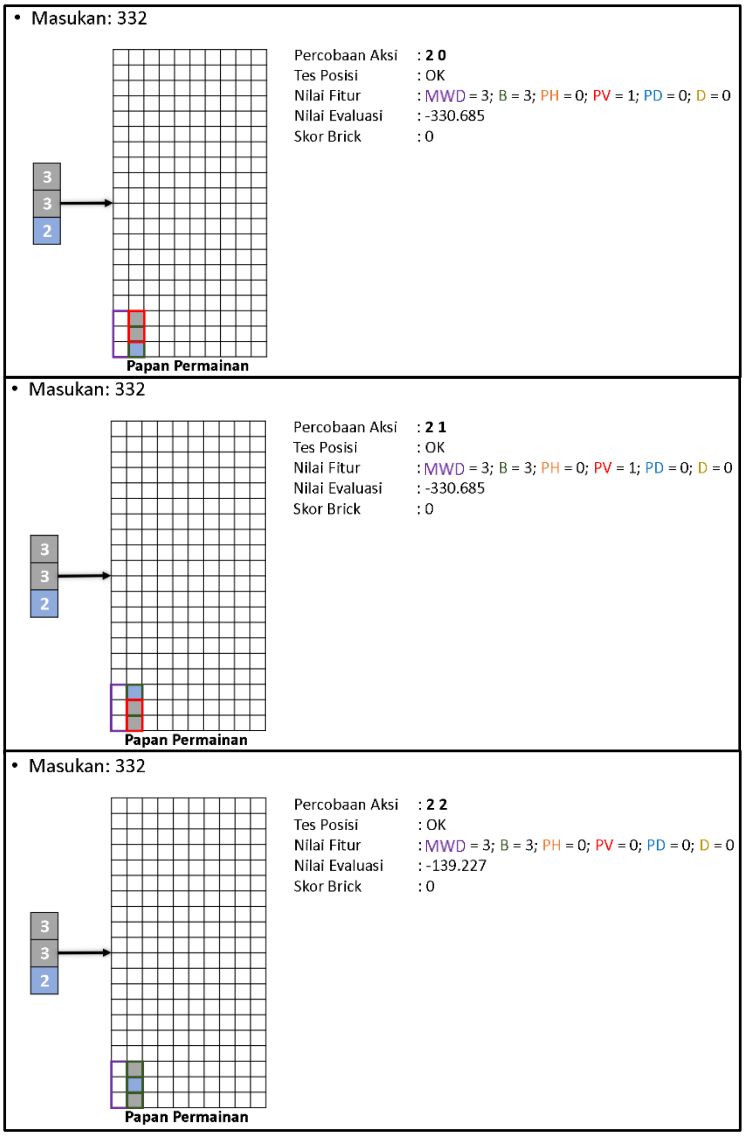
Masukan	**Keluaran**
10 3	
332	1 2
232	2 0
112	3 2
333	2 0
212	6 0
323	3 0
111	1 0
131	7 0
221	4 2
333	2 0

Tabel 5.11 Masukan dan keluaran untuk ilustrasi

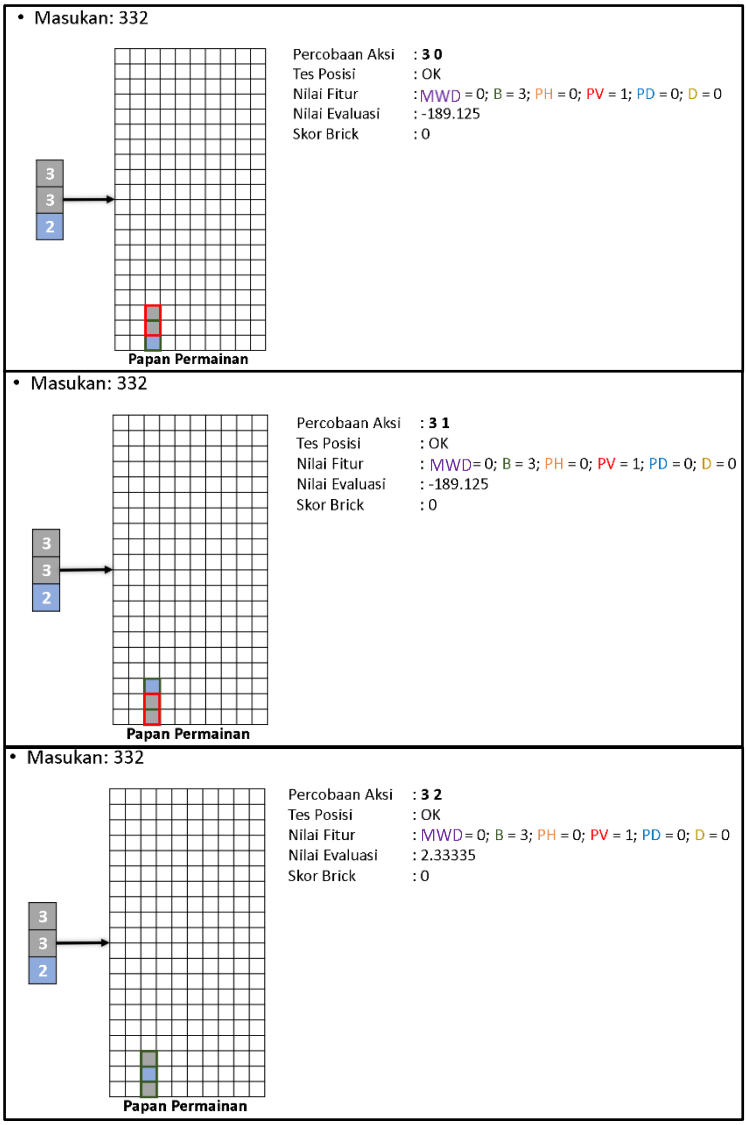
Pada masukan *brick* pertama setelah masukan N dan K, ilustrasi yang memperlihatkan proses TestPosition hingga EvaluationFeature untuk kemungkinan aksi dari masukan *brick* 332 dapat dilihat dari Gambar 5.4 hingga Gambar 5.6. Hasil nilai evaluasi dan skor pada possisi 4 hingga 8 memiliki nilai yang sama dengan posisi 3 sehingga terwakili oleh ilustrasi pada Gambar 5.6. Ilustrasi pada posisi 9 memiliki nilai evaluasi dan skor yang sama dengan posisi 2 sehingga terwakili oleh ilustrasi pada Gambar 5.5. Ilustrasi pada posisi 10 memiliki nilai evaluasi dan skor yang sama dengan posisi 1 sehingga terwakili oleh ilustrtasi pada Gambar 5.4. Untuk ilustrasi dari ChooseAction dari masukan *brick* kesatu dapat dilihat pada Gambar 5.7. Dimana hasil dari ChooseAction adalah aksi 1 2.



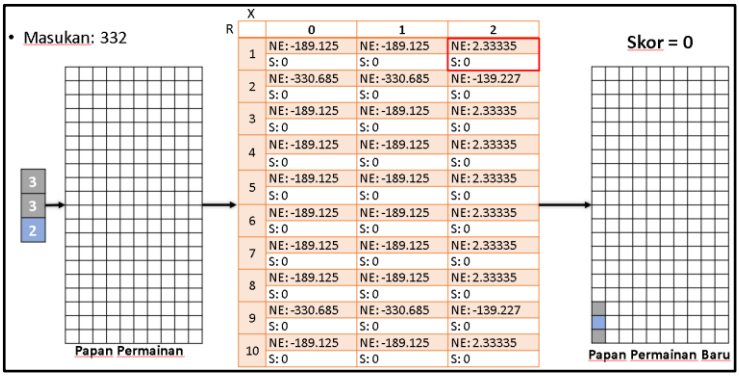
Gambar 5.4 Ilustrasi kemungkinan aksi pada posisi 1 dengan masukan *brick* pertama



Gambar 5.5 Ilustrasi kemungkinan aksi pada posisi 2 dengan masukan *brick* pertama

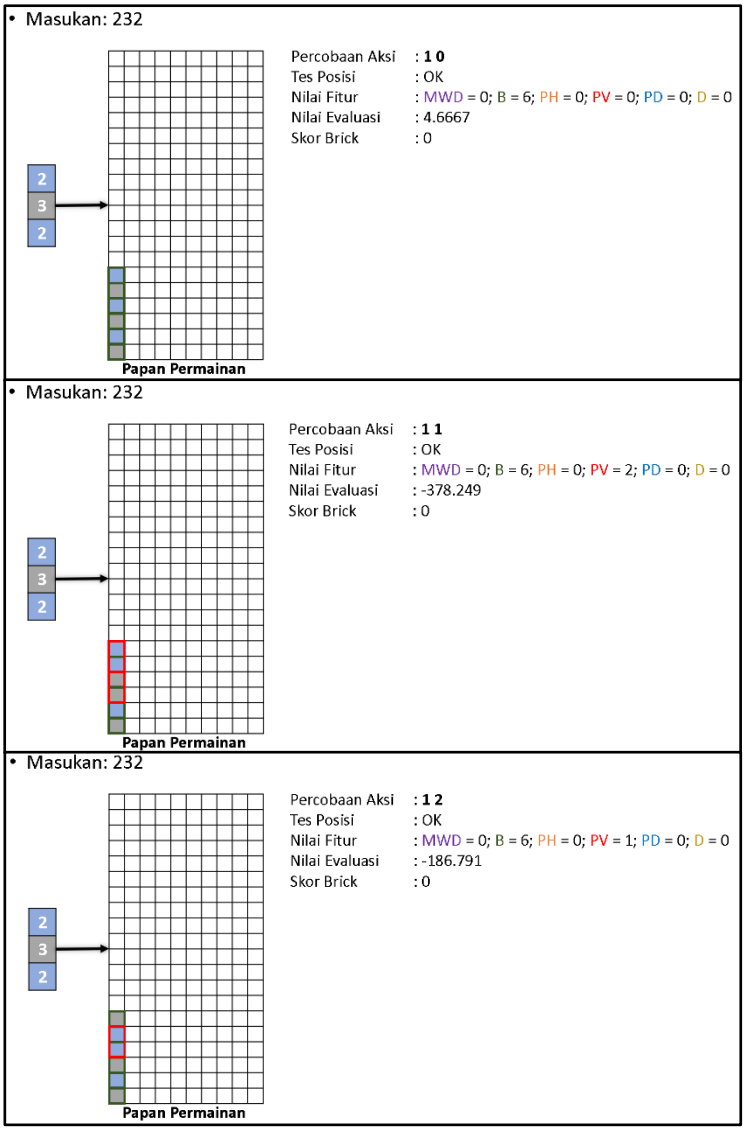


Gambar 5.6 Ilustrasi kemungkinan aksi pada posisi 3 dengan masukan *brick* pertama

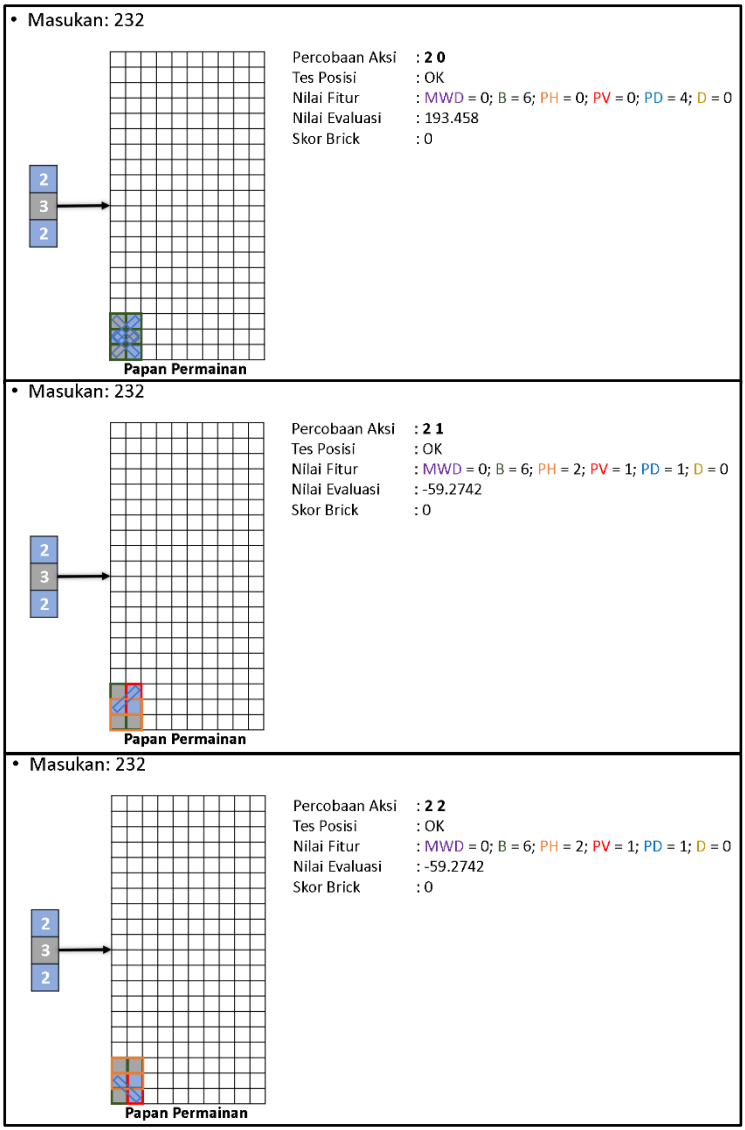


Gambar 5.7 Ilustrasi nilai evaluasi dan skor dari seluruh kemungkinan aksi pada *brick* pertama

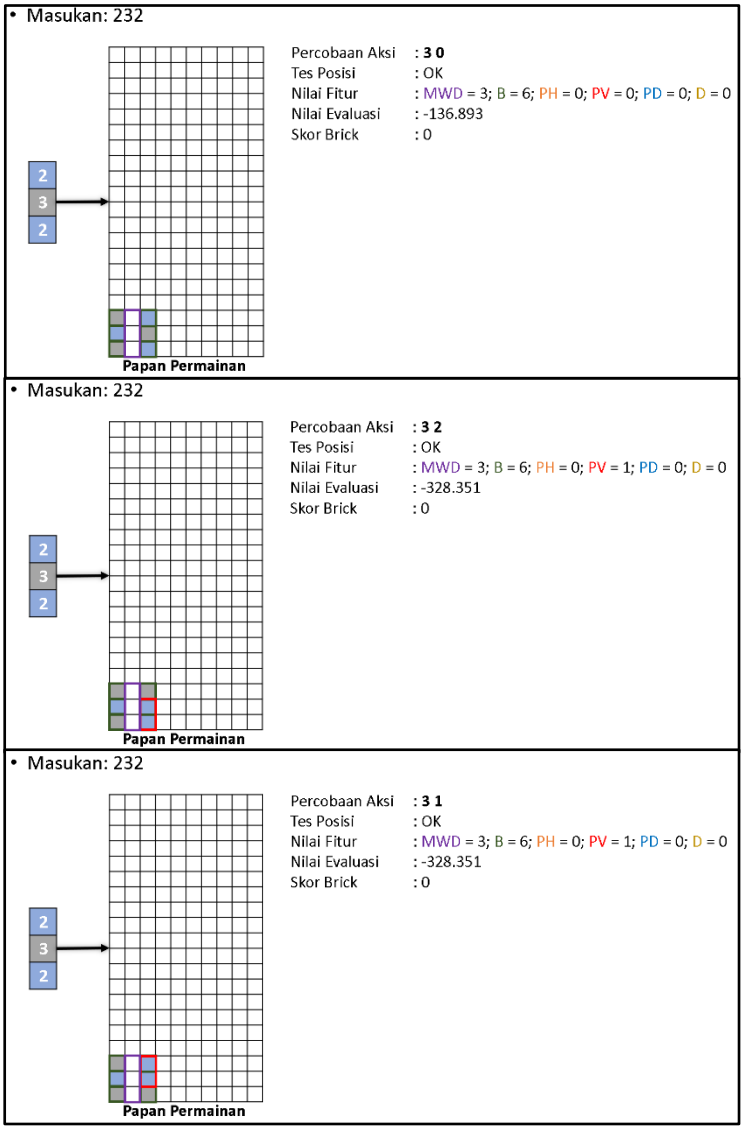
Pada masukan kedua dari *brick*, ilustrasi yang memperlihatkan proses TestPosition hingga EvaluationFeature untuk kemungkinan aksi dari masukan *brick* 232 dapat dilihat dari Gambar 5.8 hingga Gambar 5.11. Hasil nilai evaluasi dan skor pada possisi 5 hingga 8 dan 10 memiliki nilai yang sama dengan posisi 4 sehingga terwakili oleh ilustrasi pada Gambar 5.11. Ilustrasi pada posisi 9 memiliki nilai evaluasi dan skor yang sama dengan posisi 3 sehingga terwakili oleh ilustrasi pada Gambar 5.10. Untuk ilustrasi dari ChooseAction dari masukan *brick* kedua dapat dilihat pada Gambar 5.12. Dimana hasil dari ChooseAction adalah aksi 2 0.



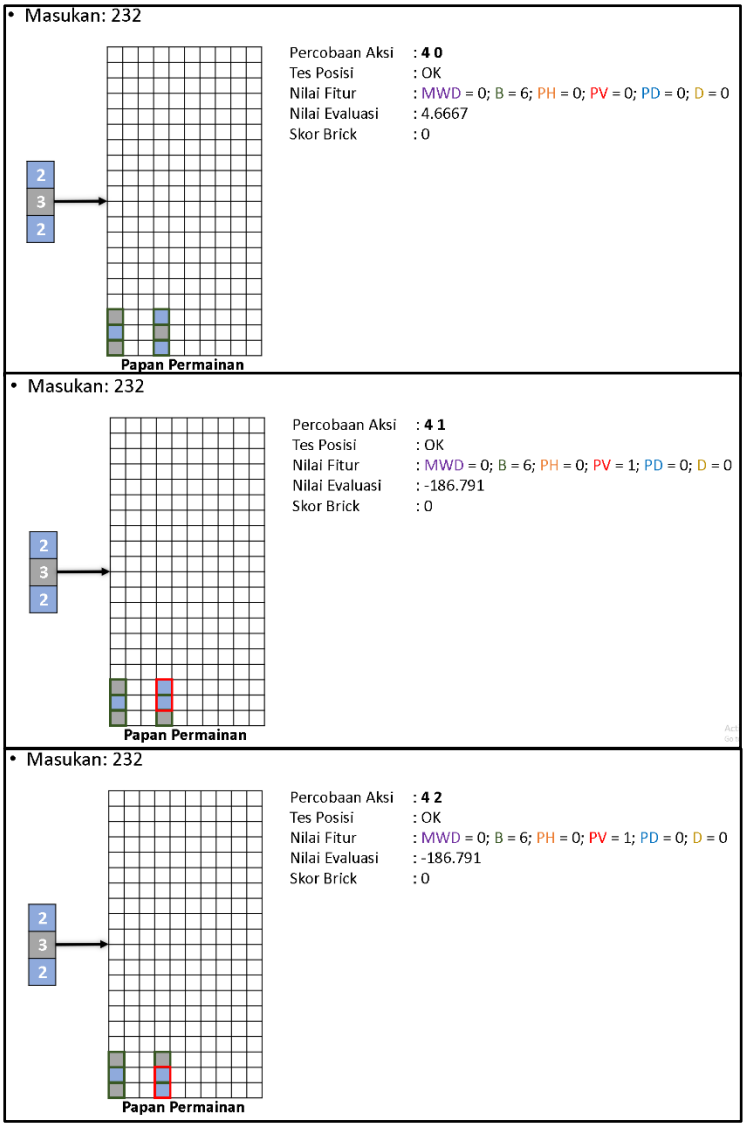
Gambar 5.8 Ilustrasi kemungkinan aksi pada posisi 1 dengan masukan *brick* kedua



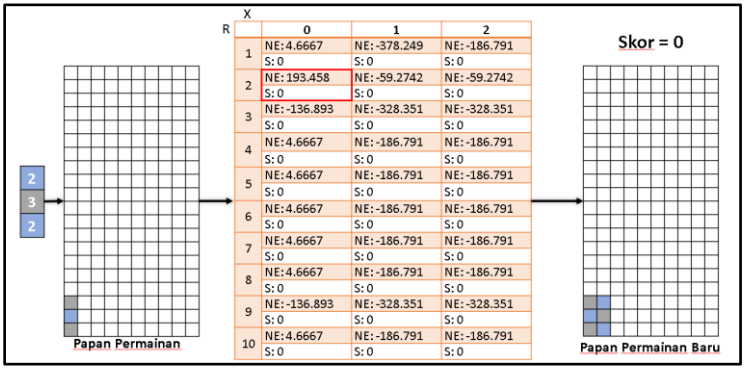
Gambar 5.9 Ilustrasi kemungkinan aksi pada posisi 2 dengan masukan *brick* kedua



Gambar 5.10 Ilustrasi kemungkinan aksi pada posisi 3 dengan masukan *brick* kedua

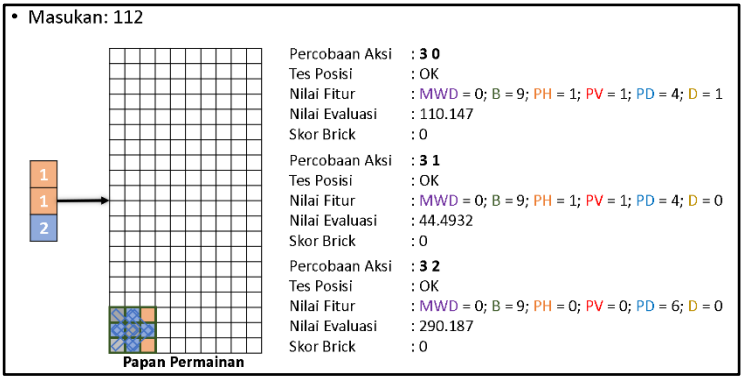


Gambar 5.11 Ilustrasi kemungkinan aksi pada posisi 4 dengan masukan *brick* kedua

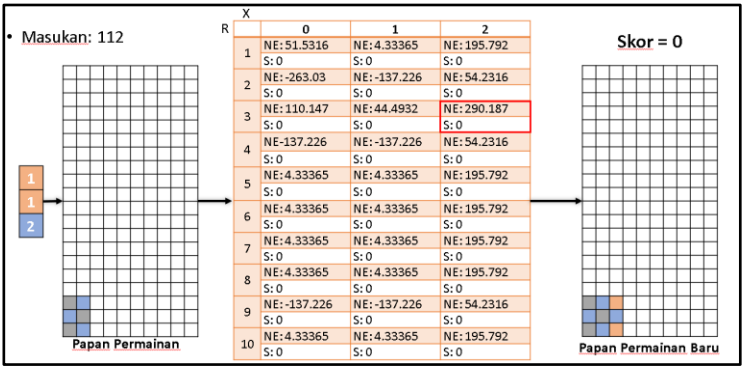


Gambar 5.12 Ilustrasi nilai evaluasi dan skor dari seluruh kemungkinan aksi pada *brick* kedua

Pada masukan ketiga dari *brick*, diilustrasikan proses TestPosition hingga EvaluationFeature untuk aksi yang menjadi keluaran dengan posisi 3 dan rotasi 2 dari masukan *brick* 112 pada Gambar 5.13. Untuk ilustrasi dari ChooseAction dari masukan ketiga dapat dilihat pada Gambar 5.14. Dimana hasil dari ChooseAction adalah aksi 3 2.



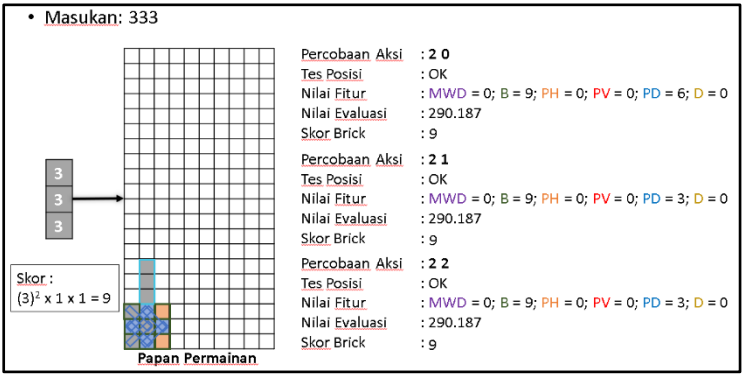
Gambar 5.13 Ilustrasi kemungkinan aksi pada posisi 3 dan rotasi 2 dari masukan *brick* ketiga



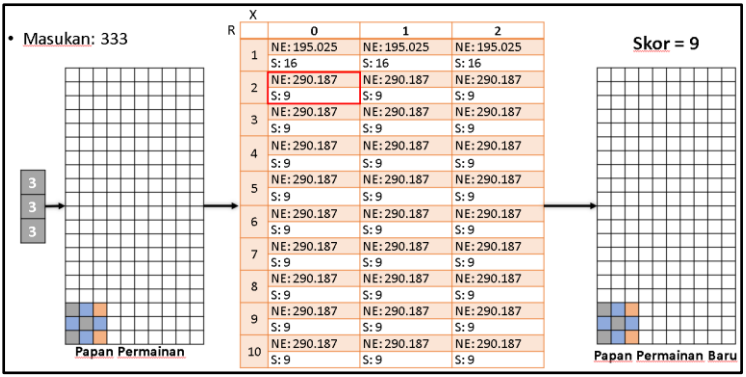
R \ X	0	1	2
1	NE:51.5316 S:0	NE:4.33365 S:0	NE:195.792 S:0
2	NE:-263.03 S:0	NE:-137.226 S:0	NE:54.2316 S:0
3	NE:110.147 S:0	NE:44.4932 S:0	NE:290.187 S:0
4	NE-137.226 S:0	NE:-137.226 S:0	NE:54.2316 S:0
5	NE:4.33365 S:0	NE:4.33365 S:0	NE:195.792 S:0
6	NE:4.33365 S:0	NE:4.33365 S:0	NE:195.792 S:0
7	NE:4.33365 S:0	NE:4.33365 S:0	NE:195.792 S:0
8	NE:4.33365 S:0	NE:4.33365 S:0	NE:195.792 S:0
9	NE:-137.226 S:0	NE:-137.226 S:0	NE:54.2316 S:0
10	NE:4.33365 S:0	NE:4.33365 S:0	NE:195.792 S:0

Gambar 5.14 Ilustrasi nilai evaluasi dan skor dari seluruh kemungkinan aksi pada *brick* ketiga

Pada masukan keempat dari *brick*, diilustrasikan proses TestPosition hingga EvaluationFeature untuk aksi dengan posisi 2 dari masukan *brick* 333 pada Gambar 5.15. Untuk ilustrasi dari ChooseAction dari masukan keempat dapat dilihat pada Gambar 5.16. Dimana hasil dari ChooseAction adalah aksi 2 0.

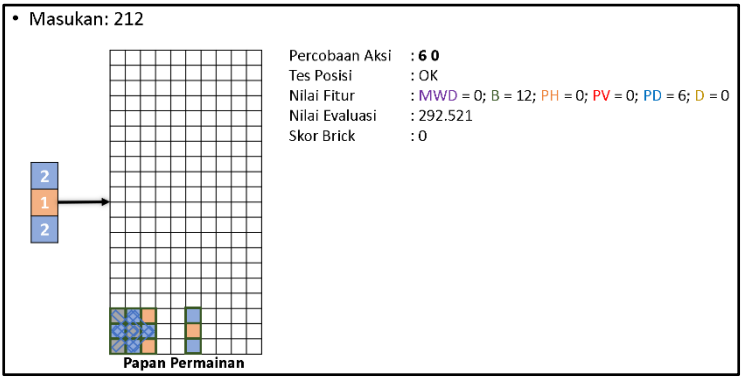


Gambar 5.15 Ilustrasi kemungkinan aksi pada posisi 3 dari masukan *brick* keempat

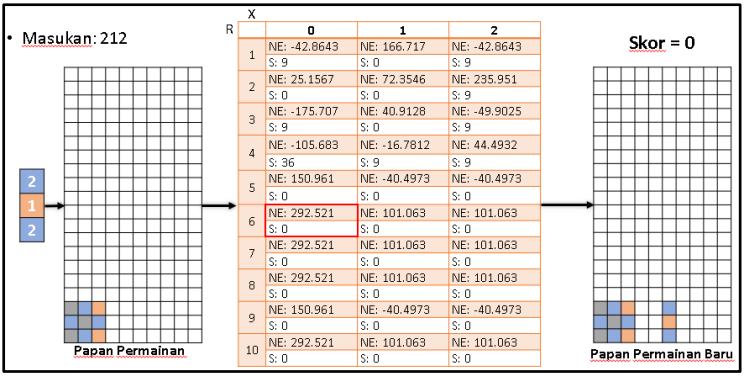


Gambar 5.16 Ilustrasi nilai evaluasi dan skor dari seluruh kemungkinan aksi pada *brick* keempat

Pada masukan kelima dari *brick*, diilustrasikan proses TestPosition hingga EvaluationFeature untuk aksi yang menjadi keluaran dengan posisi 6 dan rotasi 0 dari masukan *brick* 212 pada Gambar 5.17. Untuk ilustrasi dari ChooseAction dari masukan kelima dapat dilihat pada Gambar *5.18*. Dimana hasil dari ChooseAction adalah aksi 6 0.



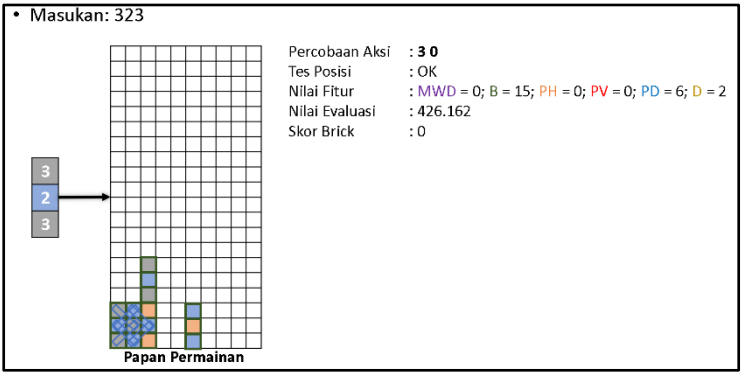
Gambar 5.17 Ilustrasi kemungkinan aksi pada posisi 6 dan rotasi 0 dari masukan *brick* kelima



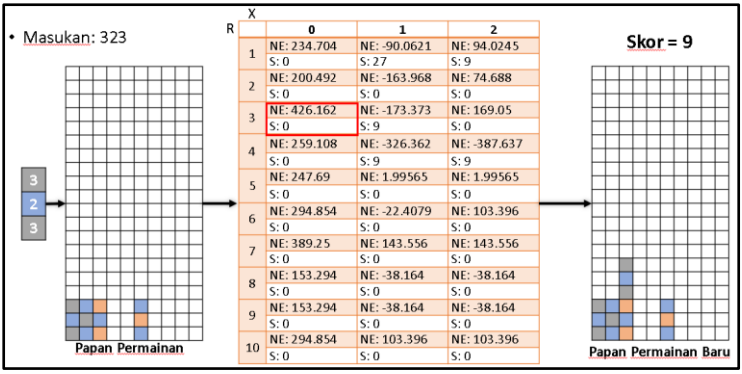
R \ X	0	1	2
1	NE: -42.8643 S: 9	NE: 166.717 S: 0	NE: -42.8643 S: 9
2	NE: 25.1567 S: 0	NE: 72.3546 S: 0	NE: 235.951 S: 9
3	NE: -175.707 S: 9	NE: 40.9128 S: 0	NE: -49.9025 S: 9
4	NE: -105.683 S: 36	NE: -16.7812 S: 9	NE: 44.4932 S: 9
5	NE: 150.961 S: 0	NE: -40.4973 S: 0	NE: -40.4973 S: 0
6	NE: 292.521 S: 0	NE: 101.063 S: 0	NE: 101.063 S: 0
7	NE: 292.521 S: 0	NE: 101.063 S: 0	NE: 101.063 S: 0
8	NE: 292.521 S: 0	NE: 101.063 S: 0	NE: 101.063 S: 0
9	NE: 150.961 S: 0	NE: -40.4973 S: 0	NE: -40.4973 S: 0
10	NE: 292.521 S: 0	NE: 101.063 S: 0	NE: 101.063 S: 0

Gambar 5.18 Ilustrasi nilai evaluasi dan skor dari seluruh kemungkinan aksi pada *brick* kelima

Pada masukan keenam dari *brick*, diilustrasikan proses TestPosition hingga EvaluationFeature untuk aksi yang menjadi keluaran dengan posisi 3 dan rotasi 0 dari masukan *brick* 323 pada Gambar 5.19. Untuk ilustrasi dari ChooseAction dari masukan keenam dapat dilihat pada Gambar 5.20. Dimana hasil dari ChooseAction adalah aksi 3 0.



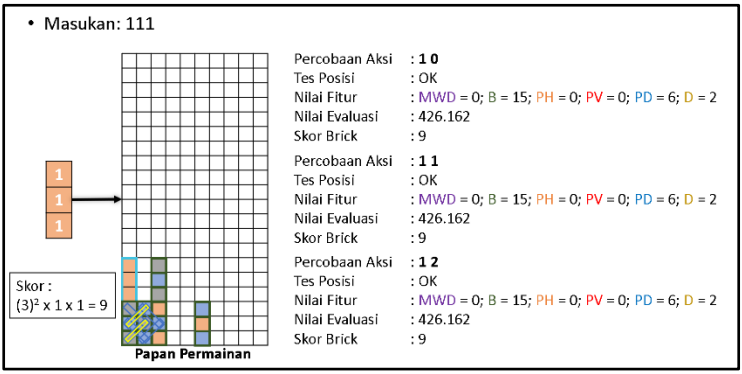
Gambar 5.19 Ilustrasi kemungkinan aksi pada posisi 3 dan rotasi 0 dari masukan *brick* keenam



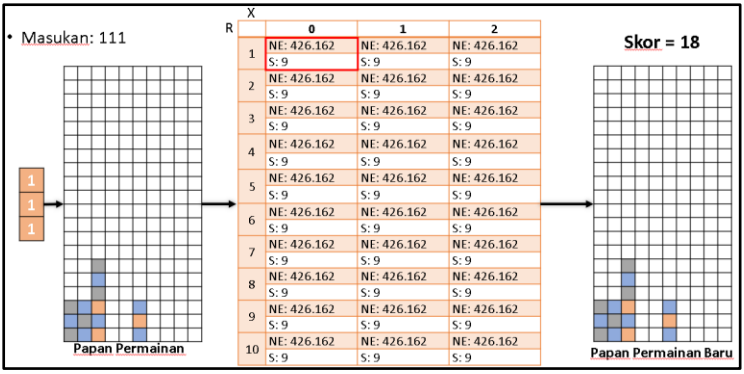
R \ X	0	1	2
1	NE: 234.704 S: 0	NE: -90.0621 S: 27	NE: 94.0245 S: 9
2	NE: 200.492 S: 0	NE: -163.968 S: 0	NE: 74.688 S: 0
3	NE: 426.162 S: 0	NE: -173.373 S: 9	NE: 169.05 S: 0
4	NE: 259.108 S: 0	NE: -326.362 S: 9	NE: -387.637 S: 9
5	NE: 247.69 S: 0	NE: 1.99565 S: 0	NE: 1.99565 S: 0
6	NE: 294.854 S: 0	NE: -22.4079 S: 0	NE: 103.396 S: 0
7	NE: 389.25 S: 0	NE: 143.556 S: 0	NE: 143.556 S: 0
8	NE: 153.294 S: 0	NE: -38.164 S: 0	NE: -38.164 S: 0
9	NE: 153.294 S: 0	NE: -38.164 S: 0	NE: -38.164 S: 0
10	NE: 294.854 S: 0	NE: 103.396 S: 0	NE: 103.396 S: 0

Gambar 5.20 Ilustrasi nilai evaluasi dan skor dari seluruh kemungkinan aksi pada *brick* keenam

Pada masukan ketujuh dari *brick*, diilustrasikan proses TestPosition hingga EvaluationFeature untuk aksi yang dengan posisi 1 dari masukan *brick* 111 pada Gambar 5.21. Untuk ilustrasi dari ChooseAction dari masukan ketujuh dapat dilihat pada Gambar 5.22. Dimana hasil dari ChooseAction adalah aksi 1 0.

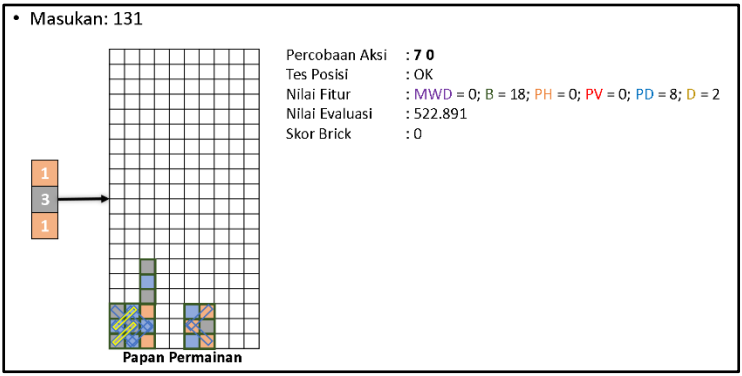


Gambar 5.21 Ilustrasi kemungkinan aksi pada posisi 1 dari masukan *brick* ketujuh

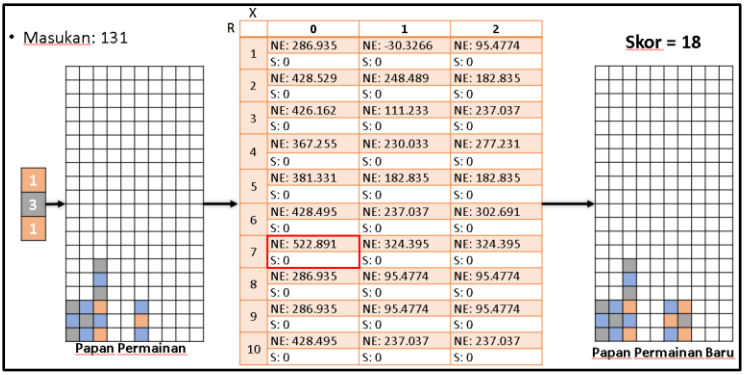


Gambar 5.22 Ilustrasi nilai evaluasi dan skor dari seluruh kemungkinan aksi pada *brick* ketujuh

Pada masukan kedelapan dari *brick*, diilustrasikan proses TestPosition hingga EvaluationFeature untuk aksi yang menjadi keluaran dengan posisi 7 dan rotasi 0 dari masukan *brick* 131 pada Gambar 5.23. Untuk ilustrasi dari ChooseAction dari masukan kedelapan dapat dilihat pada Gambar 5.24. Dimana hasil dari ChooseAction adalah aksi 7 0.



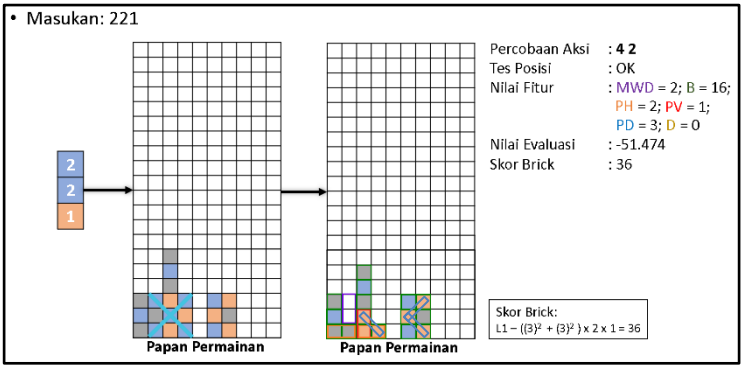
Gambar 5.23 Ilustrasi kemungkinan aksi pada posisi 7 dan rotasi 0 dari masukan *brick* kedelapan



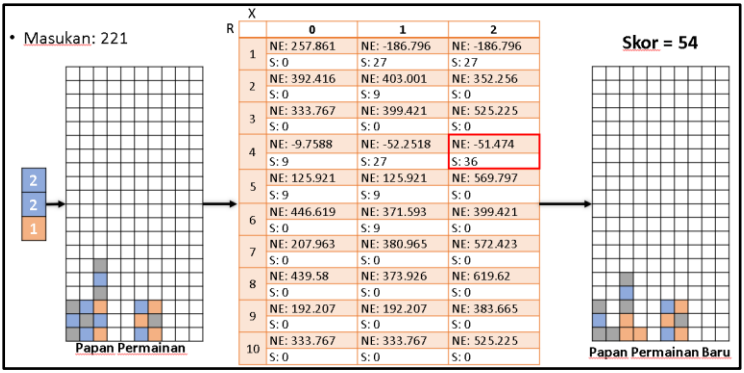
R \ X	0	1	2
1	NE: 286.935 S: 0	NE: -30.3266 S: 0	NE: 95.4774 S: 0
2	NE: 428.529 S: 0	NE: 248.489 S: 0	NE: 182.835 S: 0
3	NE: 426.162 S: 0	NE: 111.233 S: 0	NE: 237.037 S: 0
4	NE: 367.255 S: 0	NE: 230.033 S: 0	NE: 277.231 S: 0
5	NE: 381.331 S: 0	NE: 182.835 S: 0	NE: 182.835 S: 0
6	NE: 428.495 S: 0	NE: 237.037 S: 0	NE: 302.691 S: 0
7	NE: 522.891 S: 0	NE: 324.395 S: 0	NE: 324.395 S: 0
8	NE: 286.935 S: 0	NE: 95.4774 S: 0	NE: 95.4774 S: 0
9	NE: 286.935 S: 0	NE: 95.4774 S: 0	NE: 95.4774 S: 0
10	NE: 428.495 S: 0	NE: 237.037 S: 0	NE: 237.037 S: 0

Gambar 5.24 Ilustrasi nilai evaluasi dan skor dari seluruh kemungkinan aksi pada *brick* kedelapan

Pada masukan kesembilan dari *brick*, diilustrasikan proses TestPosition hingga EvaluationFeature untuk aksi yang menjadi keluaran dengan posisi 4 dan rotasi 2 dari masukan *brick* 221 pada Gambar 5.25. Untuk ilustrasi dari ChooseAction dari masukan kesembilan dapat dilihat pada Gambar 5.26. Dimana ChooseAction pada langkah kesembilan beralih dari menggunakan nilai evaluasi menjadi nilai skor setelah kondisi sisa *brick* masukan kurang dari 3 terpenuhi dengan hasil dari ChooseAction adalah aksi 4 2.



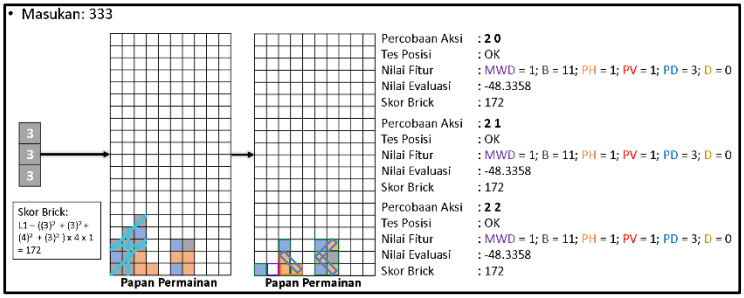
Gambar 5.25 Ilustrasi kemungkinan aksi pada posisi 4 dan rotasi 2 dari masukan *brick* kesembilan



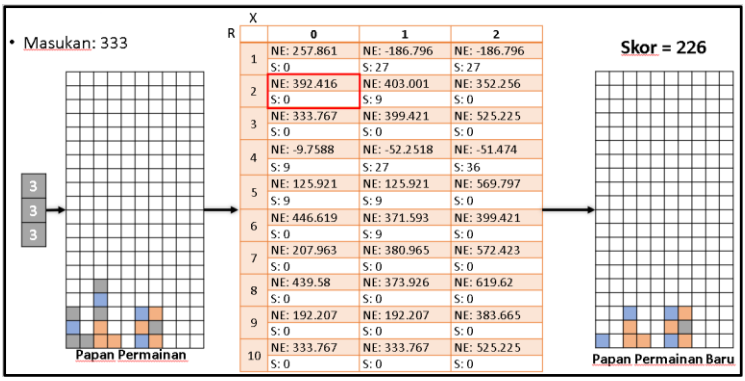
R \ X	0	1	2
1	NE: 257.861 S: 0	NE: -186.796 S: 27	NE: -186.796 S: 27
2	NE: 392.416 S: 0	NE: 403.001 S: 9	NE: 352.256 S: 0
3	NE: 333.767 S: 0	NE: 399.421 S: 0	NE: 525.225 S: 0
4	NE: -9.7588 S: 9	NE: -52.2518 S: 27	NE: -51.474 S: 36
5	NE: 125.921 S: 9	NE: 125.921 S: 9	NE: 569.797 S: 0
6	NE: 446.619 S: 0	NE: 371.593 S: 9	NE: 399.421 S: 0
7	NE: 207.963 S: 0	NE: 380.965 S: 0	NE: 572.423 S: 0
8	NE: 439.58 S: 0	NE: 373.926 S: 0	NE: 619.62 S: 0
9	NE: 192.207 S: 0	NE: 192.207 S: 0	NE: 383.665 S: 0
10	NE: 333.767 S: 0	NE: 333.767 S: 0	NE: 525.225 S: 0

Gambar 5.26 Ilustrasi nilai evaluasi dan skor dari seluruh kemungkinan aksi pada *brick* kesembilan

Pada masukan kespuluh dari *brick*, diilustrasikan proses TestPosition hingga EvaluationFeature untuk aksi dengan posisi 2 dari masukan *brick* 333 pada Gambar 5.27. Untuk ilustrasi dari ChooseAction dari masukan kesepuluh dapat dilihat pada Gambar 5.28. Dimana ChooseAction pada langkah kesepuluh beralih dari menggunakan nilai evaluasi menjadi nilai skor setelah kondisi sisa *brick* masukan kurang dari 3 terpenuhi dengan hasil dari ChooseAction adalah aksi 2 0.



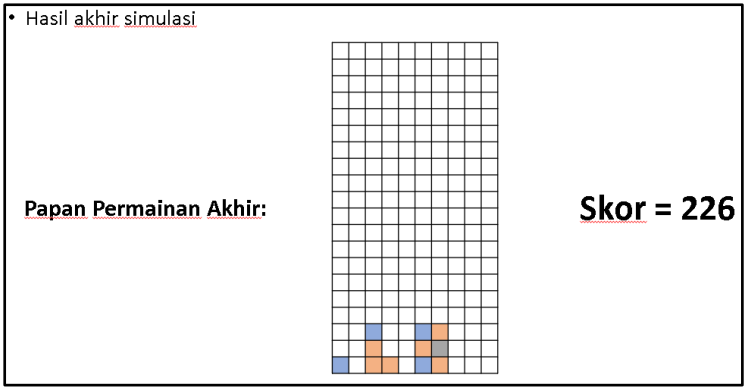
Gambar 5.27 Ilustrasi kemungkinan aksi pada posisi 2 dari masukan *brick* kesepuluh



R \ X	0	1	2
1	NE: 257.861 S: 0	NE: -186.796 S: 27	NE: -186.796 S: 27
2	NE: 392.416 S: 0	NE: 403.001 S: 9	NE: 352.256 S: 0
3	NE: 333.767 S: 0	NE: 399.421 S: 0	NE: 525.225 S: 0
4	NE: -9.7588 S: 9	NE: -52.2518 S: 27	NE: -51.474 S: 36
5	NE: 125.921 S: 9	NE: 125.921 S: 9	NE: 569.797 S: 0
6	NE: 446.619 S: 0	NE: 371.593 S: 9	NE: 399.421 S: 0
7	NE: 207.963 S: 0	NE: 380.965 S: 0	NE: 572.423 S: 0
8	NE: 439.58 S: 0	NE: 373.926 S: 0	NE: 619.62 S: 0
9	NE: 192.207 S: 0	NE: 192.207 S: 0	NE: 383.665 S: 0
10	NE: 333.767 S: 0	NE: 333.767 S: 0	NE: 525.225 S: 0

Gambar 5.28 Ilustrasi nilai evaluasi dan skor dari seluruh kemungkinan aksi pada *brick* kesepuluh

Dari hasil penyelesain permainan colour brick game yang diilustrasikan didapatkan hasil akhir berupa skor 226 dan keadaan papan permainan akhir. Ilustrasi dari hasil akhir dan papan permainan dapat dilihat pada Gambar 5.29.



Gambar 5.29 Ilustrasi hasil akhir dari aksi pada Tabel 5.11

BAB VI
KESIMPULAN DAN SARAN

Pada bab ini membahas mengenai kesimpulan yang diperoleh berdasarkan hasil uji coba yang telah dilakukan. Selain itu juga terdapat saran yang ditujukan untuk pengembangan dari penyelesain permasalahan lebih lanjut.

6.1. Kesimpulan

Dari pengamatan terhadap uji coba dan hasil pada bab 5, maka dapat disimpulkan sebagai berikut:

1. Menetukan fitur yang digunakan pada colour brick game dengan cara menganalisis cara bermain dan papan permain serta menguji coba dengan fitur lainnya.
2. Kombinasi fitur yang digunakan untuk menyelesaikan permasalahan *colour brick game* berpengaruh pada optimal atau tidaknya skor hasil dari penyelesaian.
3. Weight pada colour brick game dapat dicari dengan menggunakan algoritma genetika.
4. Besar dari weight yang digunakan untuk tiap *testcase* berbeda. Hal ini menandakan bahwa setiap kondisi dan aluran permainan yang berbeda pada *colour brick game* membutuhkan nilai weight yang berbeda.

6.2. Saran

Saran-saran yang dapat diberikan dalam penyelesaian permasalah *colour brick game* adalah sebagai berikut:

1. Pada pengembangan penyelesaian selanjutnya dapat melakukan penggabungan antara fitur yang telah ada dalam buku ini dengan lebih banyak fitur yang terdapat pada tetris. Atau pengembangan lainnya dengan menemukan fitur yang lebih mendukung untuk didapatkannya nilai optimal.
2. Memodifikasi algoritma yang terdapat pada buku ini sehingga lebih optimal atau menggunakan alur penyelesaian yang berbeda dari yang diterapkan pada buku tugas akhir ini dan tidak mengacu pada penyelesaian tetris.
3. Menggunakan algoritma yang berbeda untuk training weight tiap fitur yang digunakan pada buku ini untuk penyelesaian *colour brick game*.

DAFTAR PUSTAKA

[1] L. Flom and C. Robinson. *Using a genetic algorithm to weight an evaluation function for Tetris*, July, 2005. Available: ResearchGate, https://www.researchgates.net/publication/228610689_Using_a_genetic_algorithm_to_weight_an_evaluation_function_for_Tetris [Accessed: 23 August 2017]

[2] D. Whitley. "A Genetic Algorithm Tutorial," *Statistic and Computing*, Jun, vol. 4, no. 2, 1994.

[3] C. Z. Janikow and Z. Michalewicz. "An Experimental Comparison of Binary and Floating Point Representation in Genetic Algorithm," presented at *International Computer Games Association* 1991, pp. 31 – 36, 1991.

[4] H.Mühlenbein and D.Schlierkamp-Voosen. "Predictive Models for the Breeder Genetic Algorithm" *Journal of Evolutionary Computation*, vol. 1, no. 1, pp. 25 - 49, 1993.

[5] G. Gopal, D. Rathee and J. Rathee. "Effect of changing alpha in arithmetic crossover of GA for solving optimization functions," *International Journal of Computer Science & Communication*, vol. 4, no. 2, Sep, pp. 191 - 197, 2013.

[6] HG. Beyer and HS. Schwefel. "Evolution strategies: A comprehensive introduction" *International Journal of Natural Computing*, vol. 1, no. 1, March, pp. 12 - 18, 2002.

[7] X. Yu and M. Gen. *Introduction to Evolution Algorithm*. London: Springer, 2010, pp. 54 – 58.

[8] Thiery and B. Scherrer. "Building Controllers for Tetris," *International Computer Games Association (ICGA 2009)*, 32, pp. 3 – 4, 2009.

[9] J. B. Amundsen, “A comparison of feature functions for Tetris strategies,” Master thesis, Department of Computer and Information Science, Norwegian University of Science and Technology, Trondheim, Norwegia, 2014.

[10] J. Lewis. *Playing Tetris with Genetic Algorithm*, 2015. Available: Semantic Scholar, https://www.semanticscholar.org/paper/Playing-Tetris-with-Genetic-Algorithms-Lewis [Accessed: 23 August 2017].

[11] A. Boumaza. “On the evolution of artifical Tetris players,” *The IEEE Symposium on Computational Intellegence and Games*, Sep 2009, Milan, Italy, pp. 387 – 389, 2009.

[12] N. Böhm, G. Kókai, and S. Mandl."An Evolutionary Approach to Tetris," *The Sixth Metaheuristics International Conference*, MIC2005, Aug 2005, Vienna, Austria, p. 2, 2005.

BIODATA PENULIS

Penulis dengan nama lengkap **Destiana Nurliasari**, lahir di Malang pada 27 Desember 1996. Penulis menempuh pendidikan sekolah dasar di SD Negeri Sooko I, Kabupaten Mojokerto. Penulis melanjutkan pendidikan sekolah menengah pertama di SMP Negeri 1 Kota Mojokerto dan selanjutnya di SMA Negeri 1 Sooko Kabupaten Mojokerto. Selanjutnya penulis menempuh pendidikan sarjana di Departemen Informatika, Fakultas Teknologi Informasi dan Komunikasi, Institut Teknologi Sepuluh Nopember Surabaya.

Pada perkuliahan penulis memiliki minat pada bidang Interaksi Grafika dan Seni (IGS). Selain bidang minat pada perkuliahan, penulis memiliki minat dibidang seni gambar. Penulis memiliki kegemaran menggambar karakter kartun secara digital maupun tradisional. Hasil gambar penulis diunggah pada galeri daring.

Dalam menyelesaikan pendidikan S1, penulis mengambil bidang minat Algoritma dan Pemrograman (AP). Penulis dapat dihubungi melalui email: naru_anna@yahoo.co.id